

CE

CONRAD

Alle Versuche im Überblick

Internet of Things Advent Calendar 2017	3	12th Day	27
Source codes and additional information.....	3	In the Advent calendar today.....	27
Background knowledge on the components.....	3	Setting the running light speed by App.....	27
LEDs.....	3	The Sketch.....	27
Resistors and their colour codes.....	4	The App.....	27
1st Day	5	13th Day	28
In the Advent calendar today.....	5	In the Advent calendar today.....	28
Configuring the IoT-board.....	5	Adjusting RGB via a slider in the App.....	28
Step 1: Installation of the driver for the IoT-board.....	5	The Sketch.....	28
Step 2: Installation of the Arduino IDE.....	5	The App.....	29
Update of the firmware on the IoT-board.....	6	14th Day	30
Testing the IoT-board.....	7	In the Advent calendar today.....	30
Making the on-board-LED flash.....	7	Modelling clay contact.....	30
2nd Day	9	This is how sensor contacts work:.....	30
In the Advent calendar today.....	9	The Sketch.....	30
Measuring analogue values.....	9	The App.....	31
The program.....	9	15th Day	32
How the program works.....	9	In the Advent calendar today.....	32
3rd Day	10	Differentiable modelling clay contacts.....	32
In the Advent calendar today.....	10	The Sketch.....	32
Flashing light.....	10	The App.....	32
The program.....	10	16th Day	33
How the program works.....	10	In the Advent calendar today.....	33
4th Day	11	Controlling the flashing LED with the App.....	33
In the Advent calendar today.....	11	The Sketch.....	33
Alternating flash.....	11	The App.....	33
The program.....	11	17th Day	34
How the program works.....	11	In the Advent calendar today.....	34
5th Day	12	Display of the resistor value.....	34
In the Advent calendar today.....	12	The Sketch.....	34
Traffic light.....	12	The App.....	34
Installing Snap! and preparing the IoT-board.....	12	18th Day	35
Implementing a program in Snap!.....	12	In the Advent calendar today.....	35
6th Day	13	RGB-running light.....	35
In the Advent calendar today.....	13	The Sketch.....	35
Connection to the IoT-board.....	13	The App.....	35
Installing the App for control.....	13	19th Day	36
The program.....	13	In the Advent calendar today.....	36
7th Day	15	App to select hardware Apps.....	36
In the Advent calendar today.....	15	The Sketch.....	36
Controllable running light.....	15	The App.....	36
The program.....	15	20th Day	37
How the program works.....	15	In the Advent calendar today.....	37
8th Day	16	Heat sensor in the App.....	37
In the Advent calendar today.....	16	The Sketch.....	37
Outputting sounds through the App.....	16	The App.....	37
Development environment for the Apps.....	16	21st Day	38
Your first App with AI2.....	17	In the Advent calendar today.....	38
Controlling the piezo with the App.....	20	Measuring brightness and darkness in the App.....	38
Function of the App.....	20	The Sketch.....	38
Testing the App.....	22	The App.....	38
9th Day	23	22nd Day	39
In the Advent calendar today.....	23	In the Advent calendar today.....	39
RGB-LEDs.....	23	Moisture measurement.....	39
Changing the colour of an RGB-LED with the App.....	23	The Sketch.....	39
The Sketch for the IoT-board.....	23	The App.....	39
The App.....	24	23rd Day	40
10th Day	25	In the Advent calendar today.....	40
In the Advent calendar today.....	25	Code breaker.....	40
Displaying the push of a button.....	25	The Sketch.....	40
The Sketch.....	25	The App.....	40
Displaying the reaction of the IoT-board.....	25	24th Day	42
11th Day	26	In the Advent calendar today.....	42
In the Advent calendar today.....	26	Reaction game.....	42
LED echo by App.....	26	The Sketch.....	42
The Sketch.....	26	The App.....	42
The App.....	26		

Internet of Things Advent Calendar 2017

If Cisco has its way, more than 50 billion linked devices will be used by 2020; even more optimistically, Intel expects 200 billion devices¹. Each of these devices - or "things" - has a unique address and communicates with the outside world through the internet or through some other interfaces, such as Bluetooth: From the coffee machine to the fridge, from the car to the train, from the production machine to the bracelet: everything can be programmed and can communicate with other things. This topic is more than just a hype. It is becoming the state of the art, and we should all be dealing with it. Use the Advent season and get into the Internet of Things (IoT). 24 experiments will introduce you to the subject and let you program your own thing. Have fun!

Source codes and additional information

In the next 24 days, you will learn many new things about the subject of IoT (Internet of Things) and implement exciting projects. So that you won't have to type down the programs, some of which can be rather large, we procured the entire source codes and any additional information for download. Go to <http://www.buch.cd> and enter the code **15007-3**. You will find an archive for download there. The archive contains a separate directory for every day. For detailed information, read the file **Liesmich.pdf** in the archive.

Some of the programs from day 8 onwards are very large and not printed in full here. Only those parts of the project that you must have to understand and implement it are described in the handbook. Projects that are controlled with the dedicated Smartphone app are available for download in full. If you want to make any changes to the program, always have a look at the finished program first; then make a copy of it and modify the copy. If anything goes wrong, you can get the initial files from <http://www.buch.cd>.

Updates in the download area

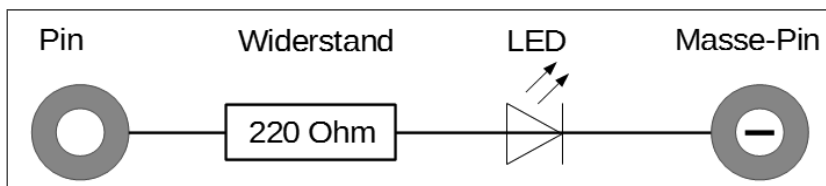
The advent calendar was created long before Advent and is based on program versions from that time. If any greater changes occur before Advent, you can find an update of the affected projects in the download area.

Background knowledge on the components

Every compartment contains one component. Here, we will provide the most important information on the components.

LEDs

LEDs (light-emitting diodes) are lit when power runs through them in the flow direction. LEDs are illustrated in circuits with an arrow-shaped triangle that indicates the current direction from the plus to the minus pole or to the ground line. An LED lets through nearly any amount of power in the flow direction, and has only a very small resistance. In order to limit the flow current and to prevent the LED from burning through, one usually must install a 220 Ohm dropping resistor between the connection pin used and the anode of the LED or between the cathode and the ground pin. This dropping resistor also protects the output of the included IoT-board from too-high currents. The LEDs in the Advent calendar have the dropping resistor already installed, and therefore can be connected directly to the pins of the IoT board.



Circuit diagram of an LED with a dropping resistor

¹ Source: <https://www.fool.com/investing/general/2016/01/18/internet-of-things-in-2016-6-stats-everyone-should.aspx>

In which direction is the LED connected?

The two connection wires of an LED are differently long. The longer one is the plus pole, the anode, the shorter one the cathode. It's easy to remember: The plus has one dash more than the minus and therefore also makes the wire a bit longer. Most LEDs are also flattened on the minus side, like a minus sign. It's easy to remember: Cathode = short = edge.

Resistors and their colour codes

Colour	Resistance in Ohm			
	1. Ring (tens)	2. Ring (ones)	3. Ring (multiplier)	4. Ring (tolerance)
Silver			$10^{-2} = 0,01$	$\pm 10 \%$
Gold			$10^{-1} = 0,1$	$\pm 5 \%$
Black		0	$10^0 = 1$	
Brown	1	1	$10^1 = 10$	$\pm 1 \%$
Red	2	2	$10^2 = 100$	$\pm 2 \%$
Orange	3	3	$10^3 = 1.000$	
Yellow	4	4	$10^4 = 10.000$	
Green	5	5	$10^5 = 100.000$	$\pm 0,5 \%$
Blue	6	6	$10^6 = 1.000.000$	$\pm 0,25 \%$
Violet	7	7	$10^7 = 10.000.000$	$\pm 0,1 \%$
Grey	8	8	$10^8 = 100.000.000$	$\pm 0,05 \%$
White	9	9	$10^9 = 1.000.000.000$	

Resistors are used to limit current at sensitive electronic components and as dropping resistors for LEDs. The unit of measurement for resistors is Ohm. 1,000 Ohm make one Kiloohm, in short: kOhm. 1,000 kOhm make one Megaohm, in short: MOhm. The unit Ohm is often represented by the Omega character \times .

The coloured rings on the resistors indicate the resistance. They are much easier to recognise than tiny figures that can still be found only on very old resistors. With a little practice, you can "translate" the values from the colour codes quite quickly.

Most resistors have four such colour rings. The first two colour rings represent the digits, the third one is a multiplier and the fourth the tolerance. This tolerance ring is usually golden or silver - colours that do not appear in the first two rings. This makes the

reading direction clear. The tolerance value is hardly relevant in digital electronics. The table shows the meaning of the coloured rings on resistors.

It doesn't matter in which direction the resistor is installed. The installation direction of LEDs, in contrast, is relevant.

1st Day

In the Advent calendar today

- 1 x IoT Bluetooth board²

Today, you will get to know the board with which you will implement the projects of the next 24 days. To prepare for the next days, you will install the driver for the USB connection, install the Arduino IDE and finally create your first program for the board.

1. Day

Chipset on the IoT-board

The IoT-board comes with two chipsets. For program execution, the board has an ATmega328P. This micro controller communicates with an HC-05 via a serial interface. The HC-05 is responsible for the wireless connection (Bluetooth). The module supports Bluetooth V2.0+EDR. You will need a Smartphone with Android for App communication.

Configuring the IoT-board

In order to take the IoT-board into operation, you need a computer with Linux, Mac OS X or Windows and a Micro-USB-cable. This connection cable serves power supply and connection of the IoT-board to the PC in order to program it. You do not need to buy such a cable. You probably have one already - almost all modern Smartphones use this plug type.

Select the proper USB-port at the PC

Connect the cable to a USB 2.0 connection of your PC if you can. USB-3.0 connections may have connection problems. You can often recognise a USB-3.0-connection by the blue socket.

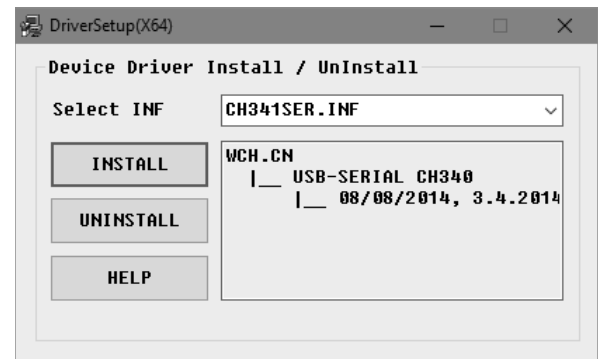
Before connecting the board to a PC, complete the following steps:

- Step 1: Installation of the driver for the IoT-board
- Step 2: Installation of the Arduino IDE

Step 1: Installation of the driver for the IoT-board

The USB port on the IoT-board is connected to a CH340G-chipset. In order to use this chipset for a USB connection, you need to install the matching driver for your operating system. Perform the following four steps for this:

- 1 Download the example programs and the drive drivers from <http://www.buch.cd>. Enter the code **15007-3** there and follow the instructions on the screen.
- 2 Unpack the ZIP archive into any folder under your user folder.
- 3 Connect the IoT-board via the USB-cable and start the driver installation with the file `CH341SER.EXE`. You may need to confirm a query from the Windows user account control for installation.
- 4 In the installation dialogue, click **Install** and wait until driver installation is confirmed.



Installation of the device driver

Step 2: Installation of the Arduino IDE

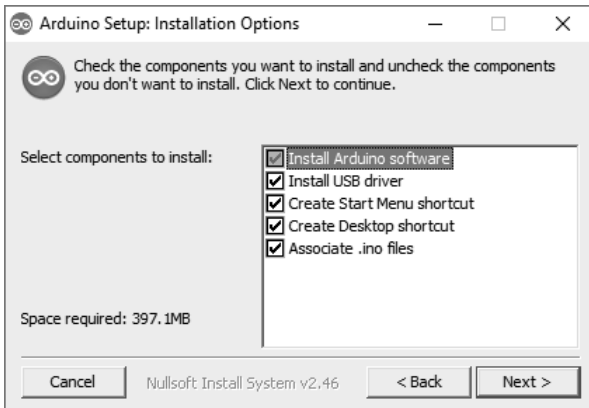
The IoT-board is compatible with the Arduino Nano and can be programmed with the Arduino IDE3. In the Arduino IDE, you can write the programs in the programming language C and transfer them directly to the IoT-board. After transfer, the program will run without any connection to the PC, which means that you may disconnect the board.

Switching off the IoT-board

The IoT-board has no off-switch. You just need to disconnect the USB-cable from the computer PC or the mains unit, and the IoT-board will switch off. The last saved program will start automatically when the board is switched on again. The same happens if you push the reset button.

² Called the IoT-board below.

³ The Arduino IDE in version 1.8.2 has been used for the projects. If a newer version is available by the time the Advent calendar is sold, you can use it instead. If there are any distinctive changes, you will find a corresponding note in the download area.



Installation of the Arduino IDE

Download the Windows Installer for the current version of the Arduino IDE from www.arduino.cc/en/Main/Software or use the file `arduino-windows.exe` from the downloads for the Advent calendar. Under Windows 10, you can also download the Arduino IDE from the Windows Store and install it from there.

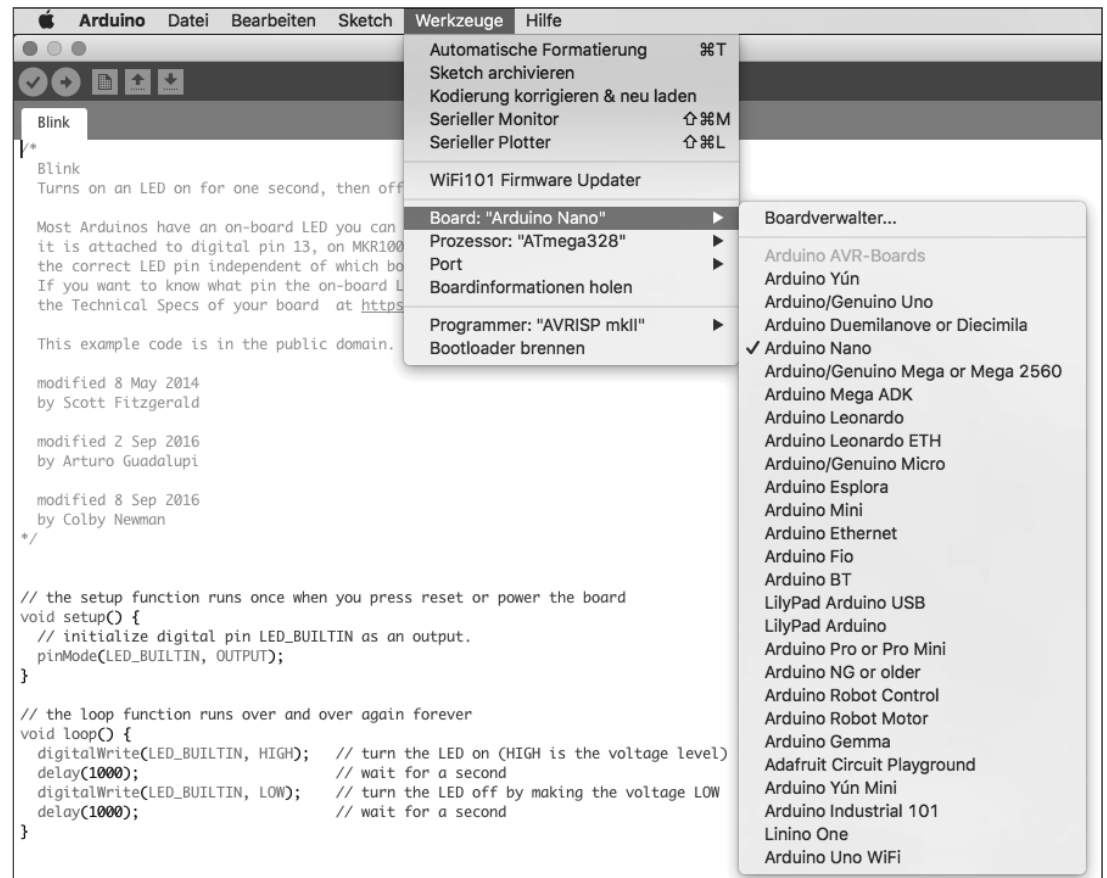
Make sure that all boxes are checked in the dialogue field **Installation Options**. Depending on the Windows configuration, the user account control must be confirmed.

Update of the firmware on the IoT-board

To enable you to reproduce all experiments with your IoT-board, first update the firmware on your board. For this, first open the previously installed Arduino IDE and choose **Tools/**

Port in the Arduino IDE menu. Only a single serial port is usually displayed here. Check this.

Then use the menu item **Tools/Board** to select the **Arduino Nano** if it has not been recognised automatically. **ATmega328** must be chosen as CPU.



Choose the right board in the Arduino IDE



Quick start of the firmware download onto the board

After starting the Arduino IDE and selecting the board, open the firmware **Firmware_V1.2b.ino** now via the menu item **File/Open....** It can be found in the download archive, directory **Firmware_V1.2b**. Before you install the firmware, ensure that the jumper on the IoT board is set to **AT**. Load the firmware onto the board via **Sketch/Upload**. You can also use the icons in the editor for this instead: Use the arrow to the right to load the firmware⁴ onto the board.

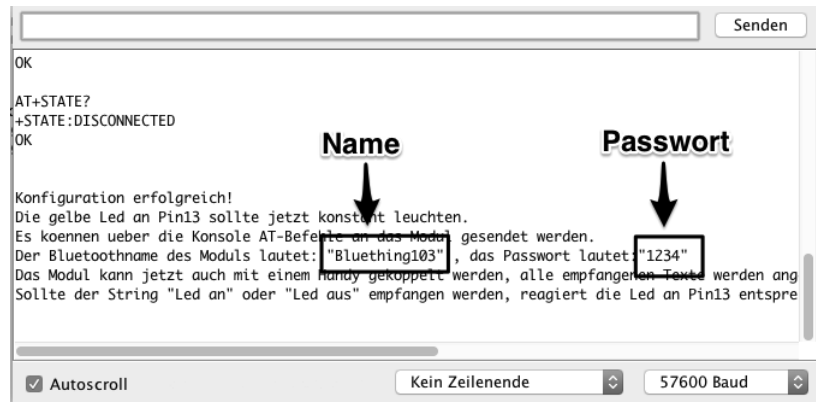
After a short time, the status line of the editor should show the message **Upload complete**. Now the IoT-board has been updated.

⁴ The firmware is a regular Arduino-program. Such an Arduino-program is called a Sketch.

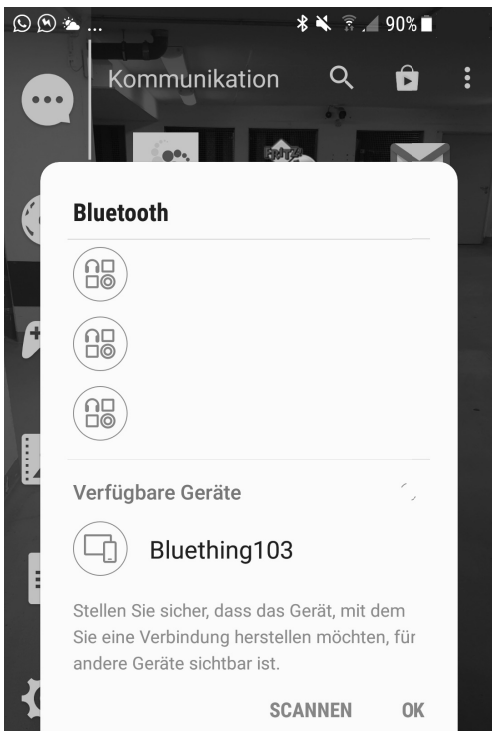
Testing the IoT-board

After updating the firmware, you can now test the board. For this, open the serial monitor of the Arduino IDE via **Tools/Serial Monitor**. Set the data transmission to **57600 Baud**. Now you should see legible text in the output window. Finally, there will be an output saying **Configuration successful!**, followed by information on the wireless network.

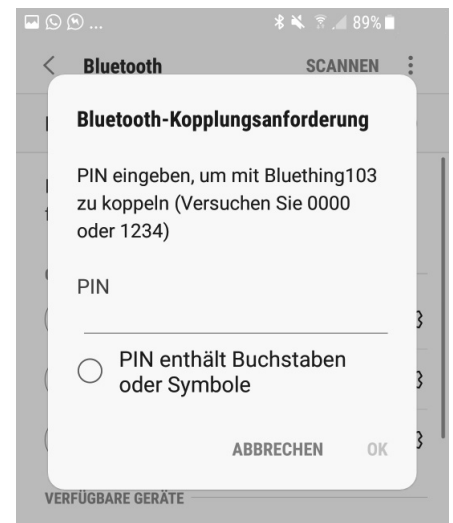
Now you can use your Android Smartphone to test whether you can see the wireless network. For this, switch on Bluetooth on your Smartphone; the wireless network should appear after a short time. Now select the wireless network (in the following screen shot: **Bluething103**) and confirm your selection with **OK**.



The name of the network in this example is **Bluething103** and the password is **1234**.



On your system, the name of the IoT-board will also start with Bluething, but may end on a different number. The number is generated individually based on the MAC address of your board.



You must enter a password to connect to the IoT-board.

You must enter a password for establishing the connection; in this case, it is **1234**.

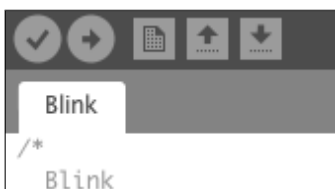
After entering the proper password, the network will appear in **CONNECTED DEVICES**. Now your Smartphone can communicate with your IoT-board.

On your system, the name of the IoT-board will also start with Bluething, but may end on a different number. The number is generated individually based on the MAC address of your board.

Making the on-board-LED flash

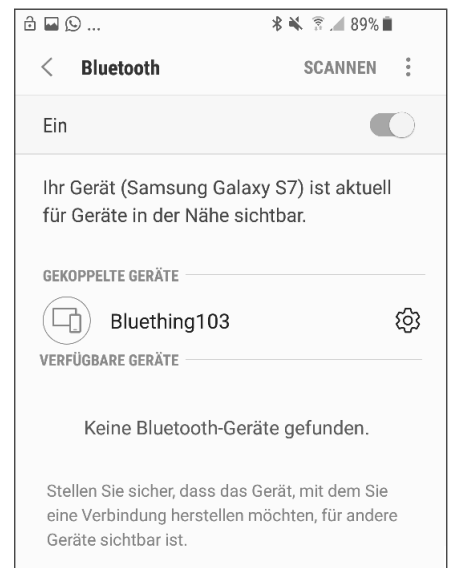
Now we want to test programming of the IoT-board. We use the flash Sketch integrated into the Arduino IDE as an example. For this, choose **Datei/Beispiele/01.Basics/Blink** in the menu.

Now click the round icon with the arrow at the upper left, in order to upload the program to the connected IoT-board, which is also called flashing.

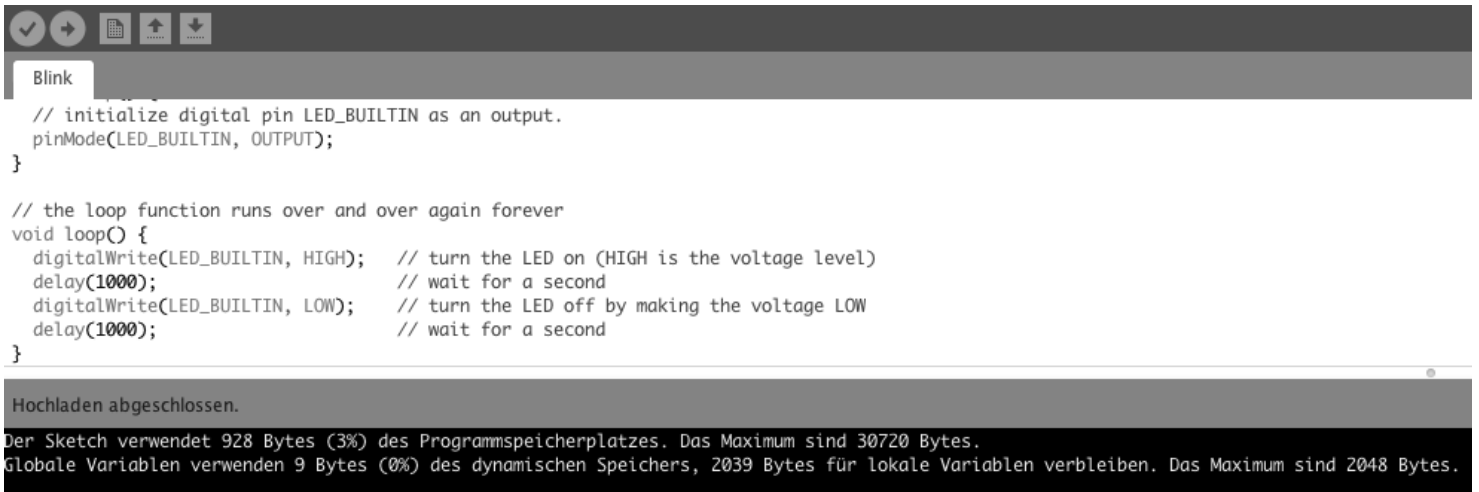


Use the check mark to compile the data; the arrow to the right uploads the program to the IoT-board.

After the upload is complete, the LED D2 (next to connection D13) on the IoT-board will flash. Now the board is ready for the next days.



Once a device has been connected to a Smartphone, the connection can be established without a new scan.



```

Blink
// initialize digital pin LED_BUILTIN as an output.
pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000); // wait for a second
  digitalWrite(LED_BUILTIN, LOW); // turn the LED off by making the voltage LOW
  delay(1000); // wait for a second
}

```

Hochladen abgeschlossen.

Der Sketch verwendet 928 Bytes (3%) des Programmspeicherplatzes. Das Maximum sind 30720 Bytes.
 Globale Variablen verwenden 9 Bytes (0%) des dynamischen Speichers, 2039 Bytes für lokale Variablen verbleiben. Das Maximum sind 2048 Bytes.

Below the source code window, you can see the outputs of the Arduino IDE when compiling and uploading.

If the LED does not flash, look at the error messages in the Arduino IDE.



Problem beim Hochladen auf das Board. Hilfestellung dazu unter <http://www.arduino.cc/en/Guide/Troubleshooting#upload>.

Der Sketch verwendet 928 Bytes (3%) des Programmspeicherplatzes. Das Maximum sind 30720 Bytes.
 Globale Variablen verwenden 9 Bytes (0%) des dynamischen Speichers, 2039 Bytes für lokale Variablen verbleiben. Das Maximum sind 2048 Bytes.

```

avrduide: stk500_recv(): programmer is not responding
avrduide: stk500_getsync() attempt 1 of 10: not in sync: resp=0x00
avrduide: stk500_getsync() attempt 10 of 10: not in sync: resp=0x00

```

Problem beim Hochladen auf das Board. Hilfestellung dazu unter <http://www.arduino.cc/en/Guide/Troubleshooting#upload>.

The connection to the IoT-board has failed here. In this case, you have chosen the wrong port; this can be fixed quickly in the menu item **Tools/Port**.

Always use the IoT-board in AT mode

The IoT-board has a Jumper. This jumper must be set to **AT** for all projects in this calendar.

2nd Day

In the Advent calendar today

- 1 x board (SYB 46)
- 1 x jumper cable

Measuring analogue values

Today, you will program a Sketch in the Arduino IDE to read out the values of an analogue input. The values will be graphically displayed by text output.

Components: 1 x board, 1 x jumper cable (male - male)

The program

The program for this day is called `Tag02.ino` and located in directory `Tag02` within the download archive.

```
int analogValue = 0;
int analogPin = A0;

void setup() {
  pinMode(analogPin, INPUT);
  Serial.begin(9600);
}
```

```
void loop() {
  analogValue = analogRead(analogPin);
  Serial.println(analogValue);
  delay(1000);
}
```

How the program works

```
int analogPin = A0;
```

First, the pin number used is specified in the variable `analogPin`. All programs in the Arduino IDE are made up at least of the two functions `setup` and `loop`:

```
void setup() {
  pinMode(analogPin, INPUT);
  Serial.begin(9600);
}
```

The function `setup` runs once initially and is usually used for configuration. The analogue pin is specified as the input. `Serial.begin(9600)` starts serial communication in order to display values on the serial monitor and the serial plotter. The parameter in this function is the Baud rate.

```
void loop() {
  ...
}
```

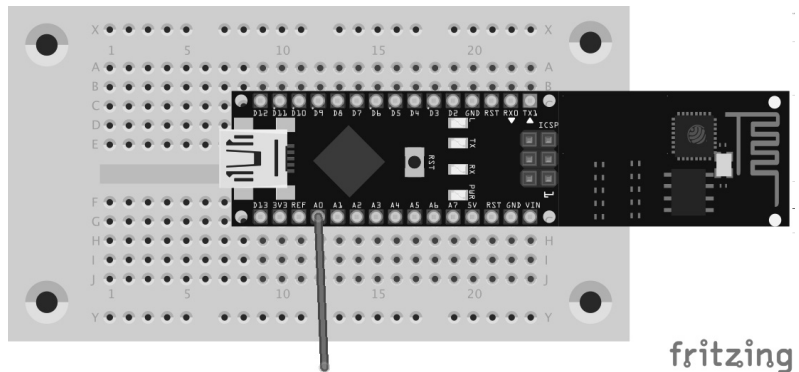
The function `loop` is repeated until the power supply is disconnected or the reset button pushed.

```
void loop() {
  analogValue = analogRead(analogPin);
  Serial.println(analogValue);
  delay(1000);
}
```

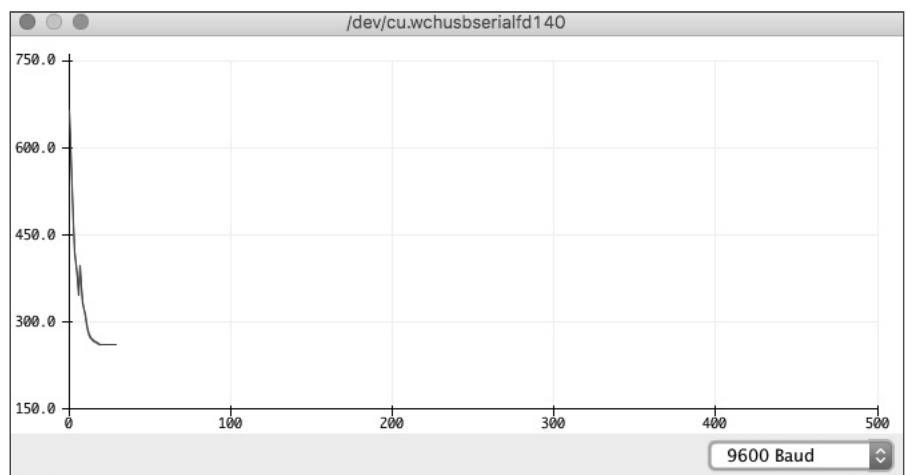
`analogRead` saves the value of the analogue input in the variable `analogValue` and outputs it via `Serial.println`. `delay(1000)` makes the program wait for 1000 milliseconds.

Now, open the serial monitor via **Tools/Serial Monitor**. You can see the measured value there. You can also have the values displayed graphically. For this, connect the previously opened serial monitor and open the serial plotter via **Tools/Serial plotter**.

2. Day



The jumper cable serves as an antenna. Try measurements without its cable as well, to see if the cable makes a difference to your measuring environment.



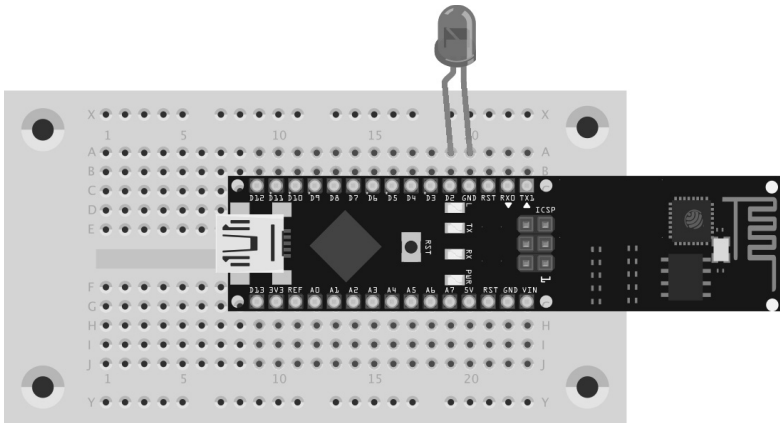
The data rate must be set correctly in the dropdown in the lower right corner of the window, in this case **9600 Baud**.

3. Day

3rd Day

In the Advent calendar today

- 1 x LED red with dropping resistor
- 1 x switching wire



fritzing

You do not need any separate resistor, since the LED already has an integrated one.

Flashing light

Today, you will make an LED flash at a frequency of 2 Hz.

Components: 1 x board, 1 x LED red with dropping resistor

The program

The program for this day is called `Tag03.ino` and located in directory `Tag03`.

```
const int ledPin = 2;
int ledState = LOW;

void setup() {
  pinMode(ledPin, OUTPUT);
}

void loop() {

  if (ledState == LOW) ledState = HIGH;
  else ledState = LOW;
  digitalWrite(ledPin, ledState);
  delay(500);
}
```

How the program works

```
if (ledState == LOW) ledState = HIGH;
else ledState = LOW;
```

The variable `ledState` records whether the LED is lit or not. Initially, the variable has the value `LOW`. This value is switched every 500 ms via `delay(500)`. This makes the LED flash.

4th Day

In the Advent calendar today

• 1 x LED yellow with dropping resistor

Alternating flash

Two LEDs flash alternately.

Components: 1 x board, 1 x LED yellow with dropping resistor, 1 x LED red with dropping resistor

The program

The program for this day is called `Tag04.ino` and located in directory `Tag04`.

```
const int ledPin1 = 10;
const int ledPin2 = 12;

int ledState = LOW;
int pin = ledPin1;

void setup() {
  pinMode(ledPin1, OUTPUT);
  pinMode(ledPin2, OUTPUT);
  digitalWrite(ledPin1, LOW);
  digitalWrite(ledPin2, LOW);
}

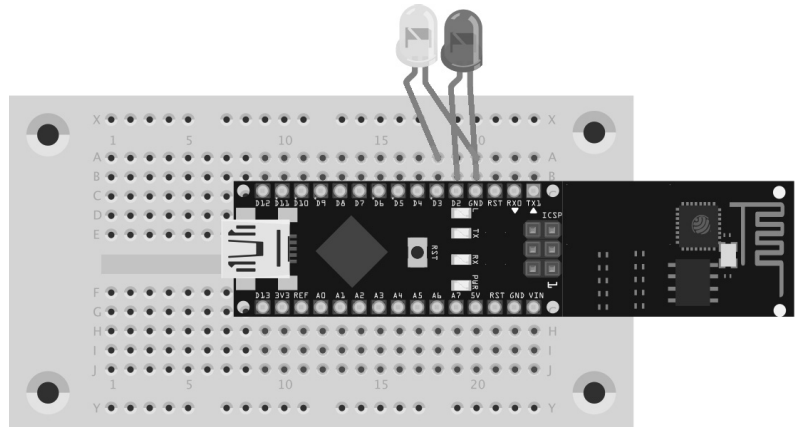
void loop() {
  if (pin == ledPin1) {
    pin = ledPin2;
    digitalWrite(ledPin1, LOW);
  } else {
    pin = ledPin1;
    digitalWrite(ledPin2, LOW);
  }
  digitalWrite(pin, HIGH);
  delay(500);
}
```

How the program works

```
if (pin == ledPin1) {
  pin = ledPin2;
  digitalWrite(ledPin1, LOW);
} else {
  pin = ledPin1;
  digitalWrite(ledPin2, LOW);
}
```

The temporary variable `pin` saves which LED is currently lit. Initially this value is `ledPin1`. An if-query switches the variable to the other pin, and the currently lit LED is switched off. Switching takes place every 500 ms.

4. Day



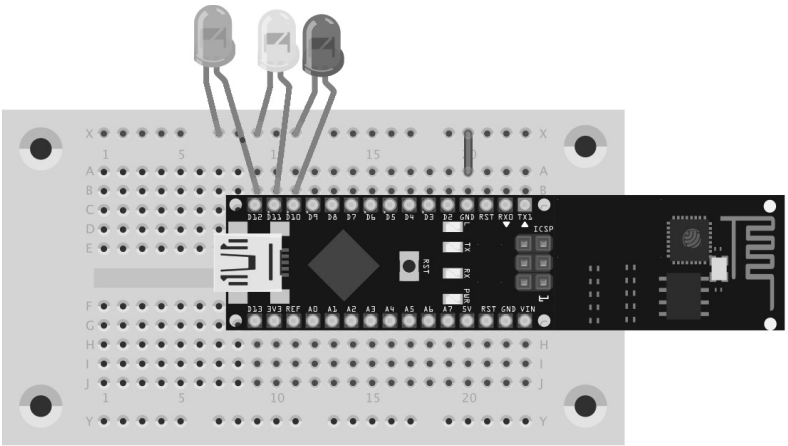
fritzing

Both cathodes (short legs of the LED) must be connected to GND. The left one is the yellow LED, the right one the red LED.

5. Day

5th Day

In the Advent calendar today
 • 1 x LED green with dropping resistor



fritzing

Starting at three LEDs, things get a little crowded, so that GND is separately connected to the upper strip. LEDs from the left to the right: green, yellow and red.



The category **Arduino** contains the elements for controlling the IoT-board.

In order to be able to use Snap4Arduino on your board, you need to install a special Sketch on your board: Firmata. For this, connect the IoT-board to your PC and open the Arduino IDE. Select **File/Examples/Firmata/Standard Firmata** and download the Sketch to the board by clicking the arrow to the right. Now you can program the board with Snap4Arduino.

Do not use Arduino IDE and Snap! at the same time

The Arduino IDE and Snap! cannot be used at the same time. Therefore, close the respective other program before you work in the desired environment.



The program created in Snap!.

Traffic light

Today's project is a traffic light that is created with the graphical development interface Snap!.

Components: 1 x board, 1 x LED green with dropping resistor, 1 x LED yellow with dropping resistor, 1 x LED red with dropping resistor, 1 x jumper

Installing Snap! and preparing the IoT-board

Snap! is a graphical development interface that exists specifically for Arduino as well in the form of Snap4Arduino. Download the software version used for this calendar from <http://www.buch.cd5>. After installation, switch the interface language to German. For this, click the settings icon (cogwheel) in Snap4Arduino and choose the language **German** in the menu **Language**.

Implementing a program in Snap!

A program is assembled in Snap! with graphical elements that are divided into categories. The category menu can be found in the upper left.

Now pull the element with the green colour into the middle working area from **Control**. Use the element **Set digital pin** in order to switch the LEDs on and off. Set the values via the switchable element **true** from **Operators**. To prevent the program from ending, you need the loop element from **Control**. To keep the LEDs from switching again at once, include a break of one second using **wait** from **Control**.

Before starting the program, you need to connect the IoT-board to Snap!. For this, click the element **Connect to Arduino** in the category **Arduino**. When clicking the symbol, you will see the available connections. Select the first connection. Now click the green arrow (upper right), and the traffic light will start.

Opening an external project in Snap!

The program for today is located in folder **Tag05**. Go to the first symbol in the menu bar and select **Import...** Navigate to the folder **Tag05** and select **tag05-snap.xml**. The project is now open and you can use it.

5 You can also download the latest version from <http://snap4arduino.org/>. The interface may deviate slightly from the screen shots, but the programs should work the same.

6th Day

In the Advent calendar today

• 1 x LED blue with dropping resistor

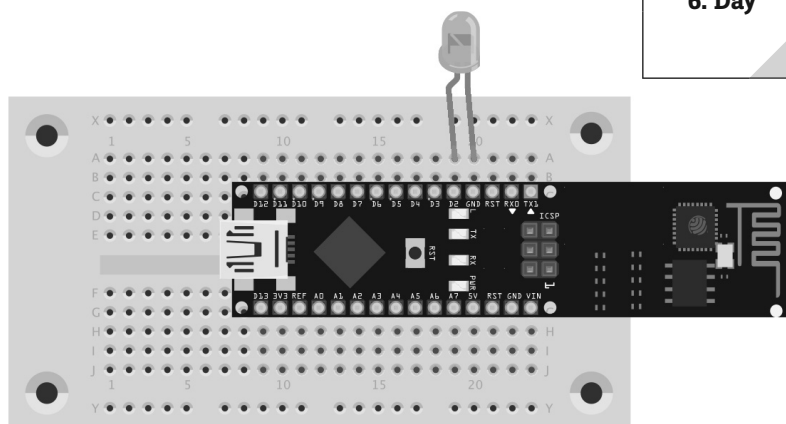
Connection to the IoT-board

Today, you will connect your Smartphone to the IoT-board and switch the blue LED on and off again with your Smartphone.

Components: 1 x board, 1 x LED blue with dropping resistor

Installing the App for control

Today, you will not use a dedicated App yet, but control the board via the free App **Serial Bluetooth Terminal** from the App-Store Google Play.



The circuit is similar to the circuit from day 3.

6. Day

fritzing

The program

The program for this day is based on the firmware for the IoT-board. The program parts needed from the firmware are enclosed in the file `Vorlage.ino` in directory `Template`.

Radio connection ready

The firmware (and thus also the template) is programmed in such a way that the orange LED at pin 13 remains lit when the wireless connection is ready. Therefore, wait until this LED lights up before you connect to the IoT-board.

Copy the file `Vorlage.ino` into a new directory `Tag06` and rename the file `Tag06.ino`. You can also use the finished file `Tag06.ino` from the directory `Tag06` right away. Now open the file with the Arduino IDE. The template contains some functions already. The constant for the internal LED is present already. The constant `LedPinBlue` for the additional LED can be defined right below the internal LED:

```
#define LedPin 13
#define LedPinBlue 2
```

At the end of the method `setup`, the LED is switched on:

```
void setup() {
  ...
  digitalWrite(LedPinBlue, HIGH);
}
```

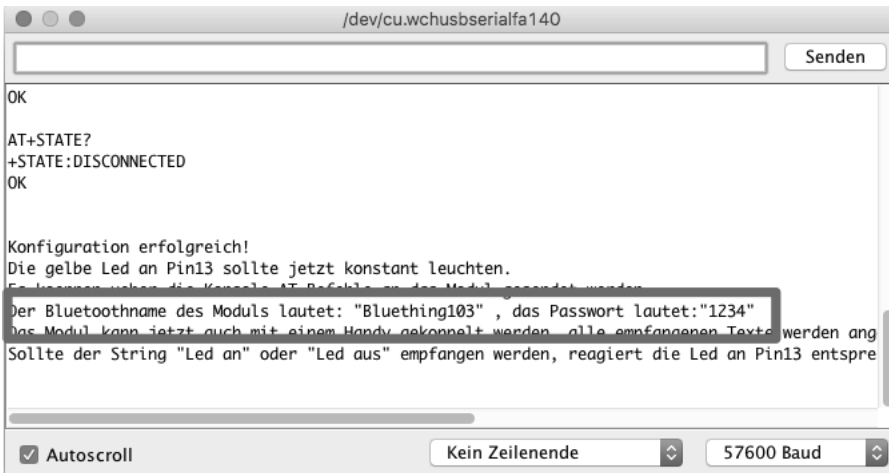
The code for switching the integrated LED at pin 13 on and off is already there. Add the following to it:

```
if (Text.startsWith("Led on") || Text.startsWith("LED on") || Text.startsWith("LED ON")){
  digitalWrite(LedPin, HIGH);
  digitalWrite(LedPinBlue, HIGH);
}
if (Text.startsWith("Led off") || Text.startsWith("LED off") || Text.startsWith("LED OFF")){
  digitalWrite(LedPin, LOW);
  digitalWrite(LedPinBlue, LOW);
}
```

When the board receives `LED on` OR `Led on` OR `LED ON`, `digitalWrite(LedPin, HIGH)` and `digitalWrite(LedPinBlue, HIGH)` will switch the internal LED and the blue LED on.



After the first start, you will see an empty black window.



The name of the network is **Bluething103** and the password is **1234**.

When the IoT-board receives data, the following loop will be performed:

```
while(HC05.available() > 0){
  ...
  Text="";
}
```

Now enter a call for the function to be programmed `hookRec(Text)` before the last line:

```
while(HC05.available() > 0){
  ...
  hookRec(Text);
  Text="";
}
```

In this function, you can now output the received text via the wireless interface:

```
void hookRec(String text) {
  if (text.startsWith("echo") || text.startsWith("Echo")) {
    HC05.print(">> " + text);
  }
}
```

Now you can install the program on the IoT-board. Switch on Bluetooth at your Smartphone and select the newly established Bluetooth network. You can see the corresponding name in the serial monitor of the Arduino IDE. Open the respective window via **Tools/Serial monitor**. When connecting, you need to enter a password; you can see the password in the serial monitor as well.

Now start the Bluetooth-App you installed before. After starting, go to the menu (three dashes on top of each other at the left edge) of the App and select the menu item **Bluetooth Devices**.

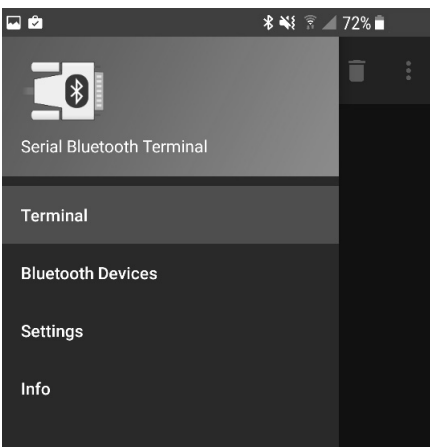
Now select the corresponding network and switch to the terminal by clicking the menu icon once and then selecting **Terminal**.

In the terminal, click the connection symbol (to the left of the garbage bin). After a short moment, you will be asked to enter the password. It is **1234**. The connection cannot be established the first time; therefore, click the icon again. Now the connection should be established. The terminal window will show the message **Connected**. Now you can communicate with the IoT-board.

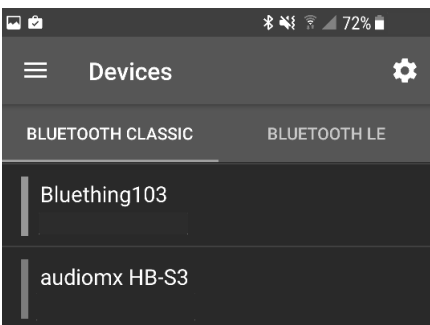
The connection must be established twice

As long as you have not entered a password in the Bluetooth-App, you need to establish the connection twice. The first time, you enter the password. The second time, the connection is established.

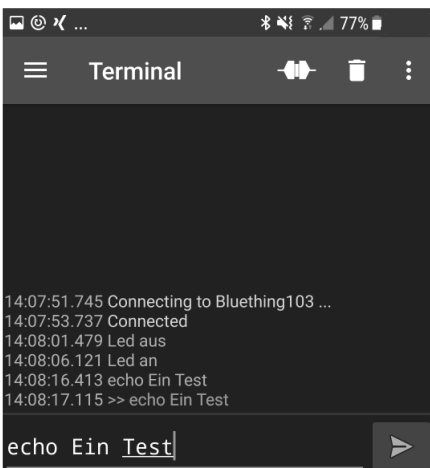
If you now enter **Led off** and push the arrow icon, the two LEDs will be switched off. Entering **Led on** will switch the LEDs on again. Every input starting **echo** will be sent to the board, which will return the input starting with two arrows. The text is then output on the terminal.



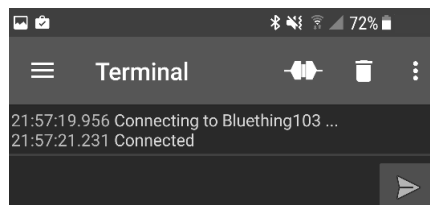
Use the central menu to get to the App settings.



You can also find the network of the IoT-board in the list of the Bluetooth networks.



echo shows that the App can also receive information from the IoT-board.



The Terminal-App is now ready for communication.

7th Day

In the Advent calendar today

• 1 x potentiometer, 15 kOhm

Controllable running light

Today's project is a running light the speed of which can be controlled with a potentiometer.

Components: 1 x board, 1 x LED green with dropping resistor, 1 x LED yellow with dropping resistor, 1 x LED red with dropping resistor, 1 x 15-kOhm-potentiometer, 4 x jumper (different lengths)

The program

The program for this day is called `Tag07.ino` and located in directory `Tag07`.

```
int analogPin = A5;
int analogValue;

int led1 = 8;
int led2 = 6;
int led3 = 4;

void setup() {
  pinMode(analogPin, INPUT);

  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  digitalWrite(led1, LOW);
  digitalWrite(led2, LOW);
  digitalWrite(led3, LOW);
  Serial.begin(9600);
}

void loop() {
  analogValue = analogRead(analogPin);
  Serial.println(analogValue);

  delay(analogValue);
  digitalWrite(led3, LOW);
  digitalWrite(led1, HIGH);

  delay(analogValue);
  digitalWrite(led1, LOW);
  digitalWrite(led2, HIGH);

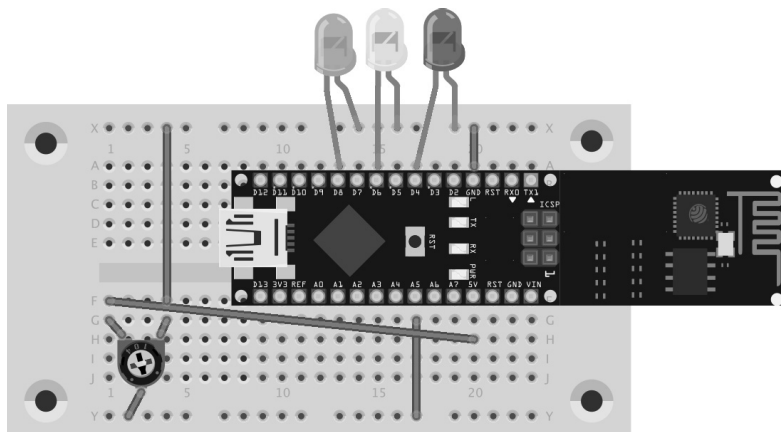
  delay(analogValue);
  digitalWrite(led2, LOW);
  digitalWrite(led3, HIGH);
}
```

How the program works

```
analogValue = analogRead(analogPin);
delay(analogValue);
digitalWrite(led3, LOW);
digitalWrite(led1, HIGH);
```

The temporary variable `analogPin` is used to read the value set for the potentiometer. `delay` is used to wait and then switch the LEDs.

7. Day



fritzing

The potentiometer takes up some space. Therefore, the IoT-board must be pushed up higher than on the day before. LEDs from the left to the right: green, yellow and red.



8th Day

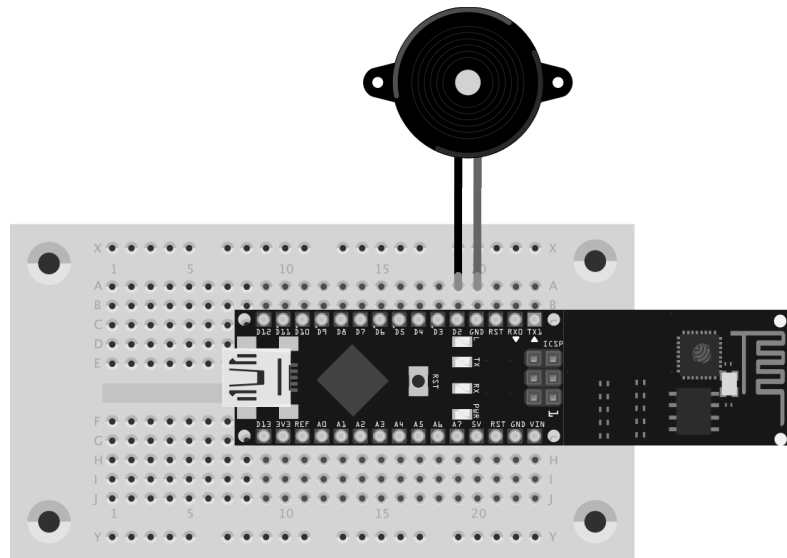
In the Advent calendar today

- 1 x piezo

Outputting sounds through the App

An App can be used to output sounds on a piezo.

Components: 1 x board, 1 x piezo

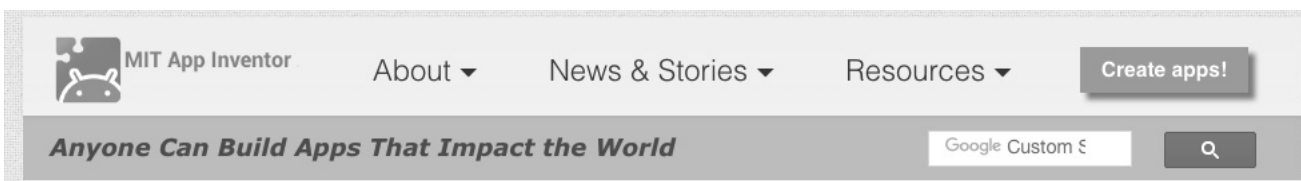


fritzing

The two wires of the piezo are connected to D2 and GND. The rest is solved via the software.

Development environment for the Apps

The IoT-board in this Advent calendar is controlled using the **Serial Bluetooth Terminal** App that has been used before, and the self-developed Smartphone-Apps for the Android operating system⁶. The finished project files can be downloaded at <http://www.buch.cd> each. The development environment used is MIT App Inventor 2 (<http://appinventor.mit.edu>).



The development environment runs in the browser. Therefore, you need internet access during development.

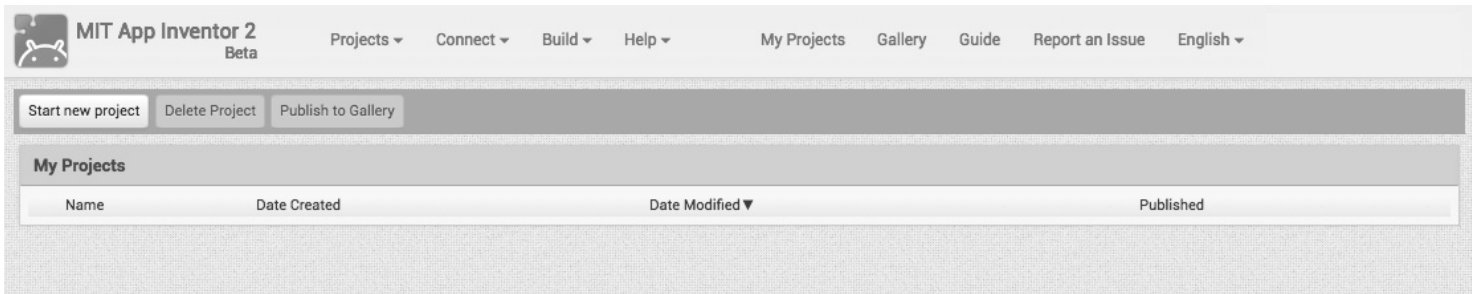
Click the button **Create apps!** to start the development environment. Using them requires a free account with Google.

Setting up an account with Google

If you have no Google account yet, set up an account for development with the **MIT App Inventor 2 (AI2)** via <https://accounts.google.com>.

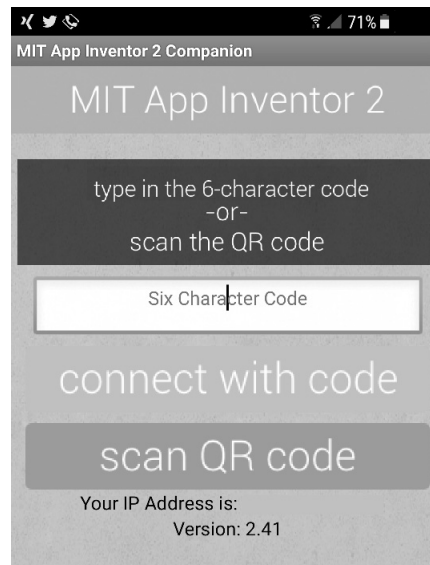
After successful login, you will get to the dashboard. Here, you will see all Apps developed so far. Since you have probably never developed an App with AI2 before, you will see an empty dashboard.

⁶ The chipset integrated on the IoT-board is supported only by Android-Smartphones.



The interface is not yet available in German.

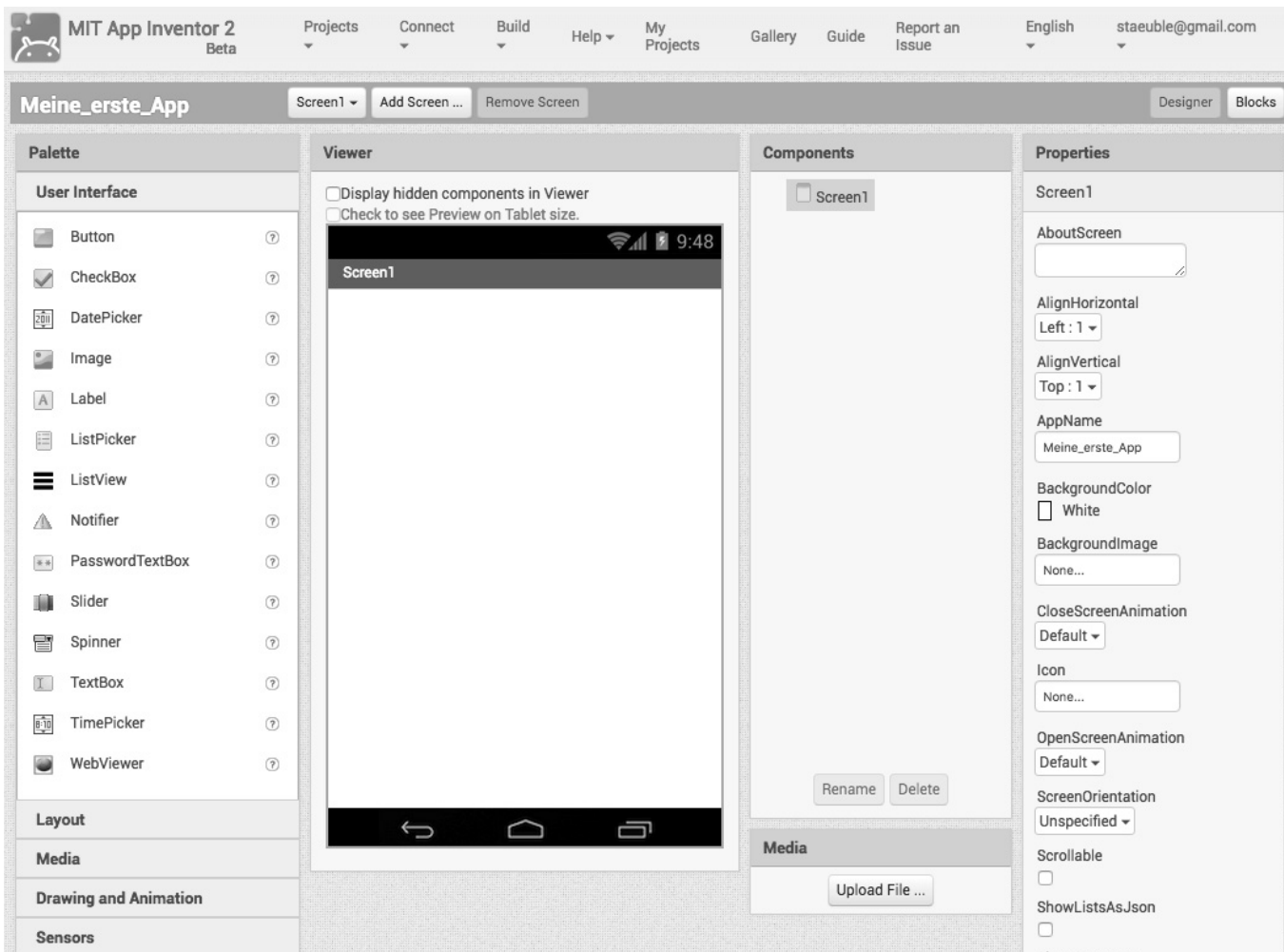
In order to test your self-developed App during development, you need to install the free App **MIT AI2 Companion** from the Google Play Store on your Smartphone. After starting, you need to enter the code for your App; for this, you need to create an App first.



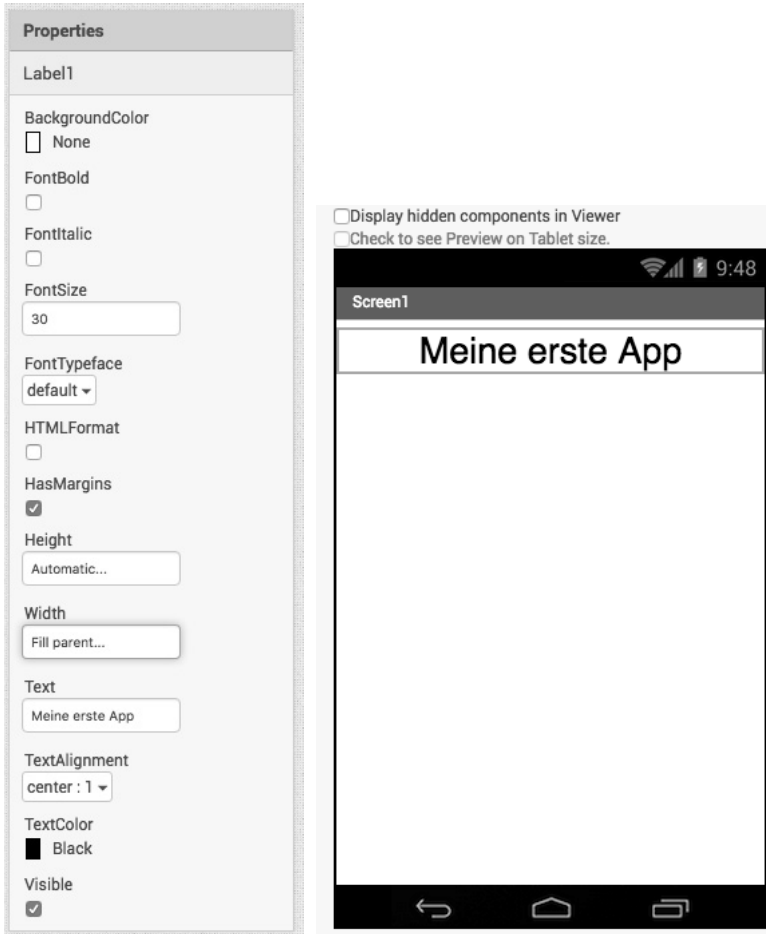
During development, do not start the App directly but take the detour through the MIT App Inventor 2 Companion. This way, you do not need to install the App on the device manually.

Your first App with AI2

Now click the start button **Start new project** and enter the name **Meine_erste_App**. Note that the name must not contain any spaces. The development environment will open in your browser now.



After starting, the App interface will still be empty.



You can change components in the window **Properties**. The display in the Viewer will change at once then.

The window is broken up into several areas. On the left, you can see the interface elements available for the App in the area **Pallet**. Next to it, you can see the interface of your App in the **Viewer**. **Components** shows the components used in your App and **Properties** the properties of the currently selected component.

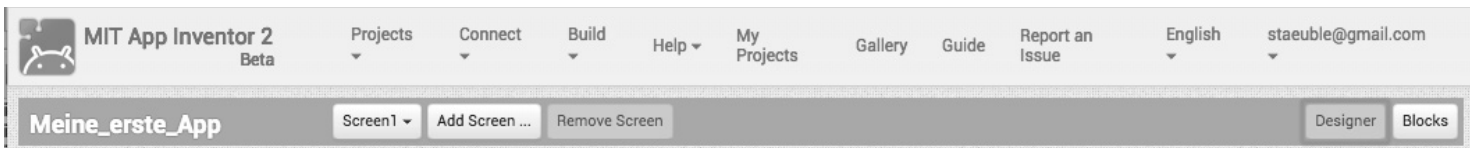
Now draw text in the form of a label into the window. Adjust the label by setting **Width** to **Fill parent** and **TextAlignment** to **center**. Now the label is centered horizontally. Next, change the font size to **FontSize 30** and change the text (field: **Text**) to **My first App**.

Now the App is to be expanded by an interaction. For this, enter buttons from the window **Components** and change the label to **Touch me** via the **Properties**. Below this, insert a label with the text **Status: not touched**.



App-interface with the three components

AI2 has two views for an App: the view for design of the interface, called the Designer (tab **Designer**), and the view for programming in block language (tab **Blocks**). These two tables are displayed in the upper area of the development environment.

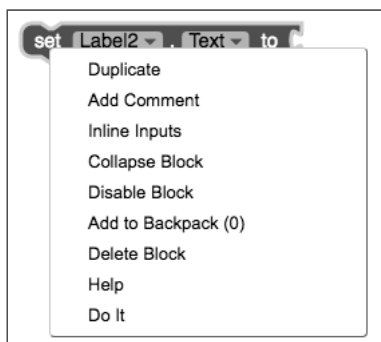


Use the tabs **Designer** and **Blocks** to get to the two views of the development environment.

Clicking **Blocks** will open the view of the block programming language. Now select the component **Button1** within the **Blocks** area; you will see the available results to which you can react in the App.

Now select the two blocks **when Button1.TouchDown** and **when Button1.TouchUp**. In both cases, the content of the label **Label2** is to be adjusted. For this, click **Label2** and select the element **set Label2**.

Text to. Either insert the element twice or insert the element once and copy it via the context menu (right mouse button) of the component.

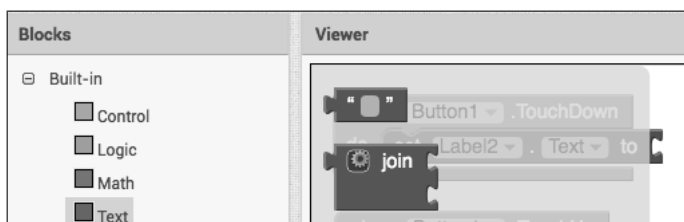


Components have a context menu. Use **Duplicate** to copy the selected component.

Now insert the two components within the previously inserted elements (**when Button1.TouchDown** and **when Button1.TouchUp**). Now you still need text content. For this, select the empty element from the category **Text** (top-most element).



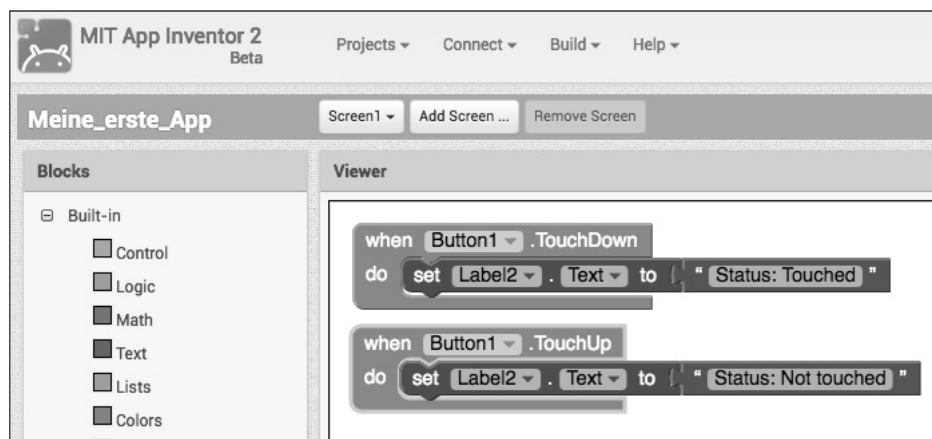
For programming the logic, you can go to the selection via categories (**Built-in**) or via the components (**Screen1**).



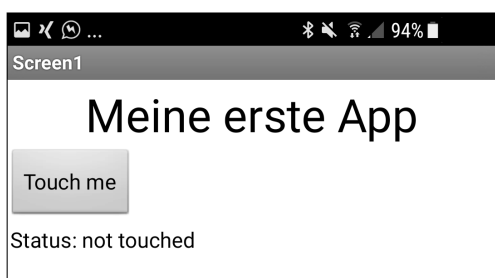
The element with the two quotation marks is used for changing the label.

In case of **when Button1.TouchDown**, the text is set to **Status: Touched**. For **when Button1.TouchUp**, the text is set to **Status: Not touched**.

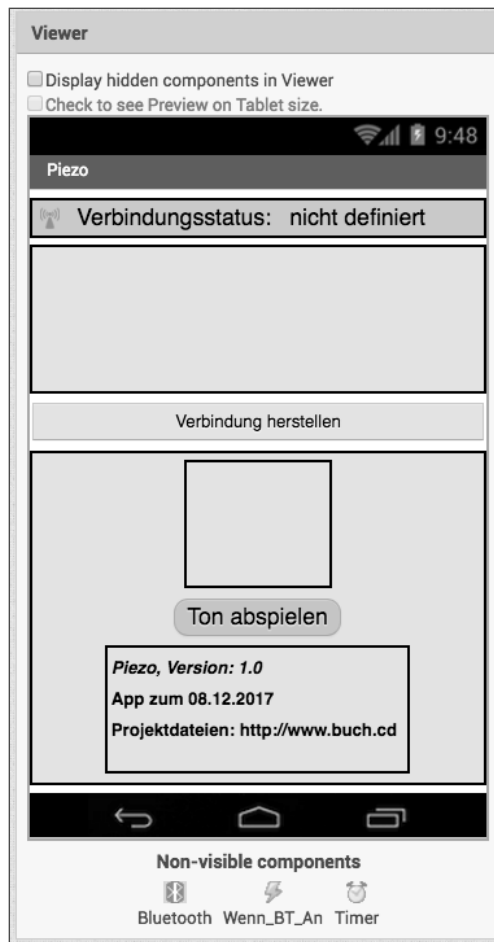
Now you can test the app on your Smartphone. For this, select **Connect/AI Companion** in the menu. Now a text code and a QR-code will appear. Enter either the code in the **MIT AI2 Companion** or scan the QR-Code with **scan QR code**. Start the App with **connect with Code**. Now the App opens on your Smartphone. When you touch the button, the text will change.



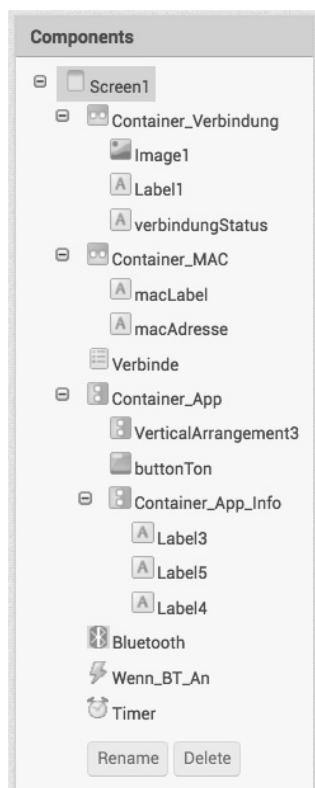
The text of the label is adjusted when the button is touched.



This is how simple App development with AI2 is.



The upper area of the App shows whether the Bluetooth connection has been established or not.



All components contained in the App in a hierarchy overview. The most important components have been given indicative names.

Controlling the piezo with the App

In order to control the IoT-board with an App, you need a Sketch on the board that reacts to the commands from the App and an App. Today, we want to control the enclosed piezo with an App. The Sketch you need for this is `Tag08.ino` in the directory `Tag08`. Load this Sketch onto the IoT-board.

The Sketch is based on the already-used file `Vorlage.ino`. The principle of all Sketches in this Advent calendar that react to the App is as follows: A text is sent through the wireless interface, which is evaluated in the Sketch, followed by execution of an action. It works the same way in the other direction: The App will receive a text from the IoT-board via the wireless interface, evaluate the text and then perform a corresponding action. The command to play a sound is `Sound`:

```
if (Text.startsWith("Sound") || Text.startsWith("SOUND") || Text.startsWith("sound")){
    playMelody();
}
```

The associated App is contained in file `piezo.aia`. Import the file into AI2.

Importing projects into AI2

All Apps are provided as aia-files ready for use in the download archive. In order to open a file in AI2, you must import the project file via the menu item **Projects/Import project (.aia) from my computer ...**

Function of the App

The following Apps that were created with AI2 are built based on a template; therefore, this App is now described precisely and referred to in the further course. The App is made up of four areas in total:

- Connection area
- Function area
- App info area
- Internal, invisible area

The left figure shows the individual areas of the App. The connection area includes **Container_Verbindung** and **Container_MAC**. The actual App, which changes from project to project, is located in the **Container_App**. This container contains the function area (**buttonTon**) and the App information area (**Container_App_Info**).

What is a container?

You can place your elements using various layout managers in the category **Layout** (e.g. **HorizontalArrangement**); these help you with horizontal and vertical placement. The elements are placed within this kind of layout manager. Since these layout managers usually contain several elements, they are called containers (layout containers).

Changing component names

If you add components to an interface in the AI2, the component will receive an automatically generated name. In contrast to the other properties of a component, you cannot change names in the window **Properties**, but need to use the area **Components** for this. Here, you can select the corresponding component and click the button **Rename**.

The interface contains three invisible components in total. For use of Bluetooth, you need the component **BluetoothClient** from the category **Connectivity**. The component is only pulled into the interface area and does not require any further configuration. To select the corresponding Bluetooth device, the component **ActivityStarter** (in the upper screen **Wenn_BT_An**) from **Connectivity** is needed. The ActivityStarter is assigned the value `android.Bluetooth.adapter.action.REQUEST_ENABLE` in the window **Properties**. For continuous verification of the wireless connection, the element **Click** (in the upper figure **Timer**) from **Sensors** is used. The **TimerInterval** is set to **200** in window **Properties**.

Programming takes place within the tab **Blocks**. The **Timer** sets the connection status. The property **isConnected** can be used to check if the App is connected to a Bluetooth device.

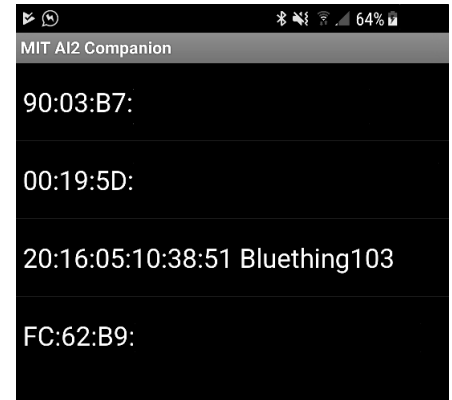
When clicking the button **Connect**, the list of available Bluetooth devices will be displayed. If the list is empty, you need to go back into the App, where you will automatically be asked for the access rights for the Bluetooth interface. Enter the password **1234**. When clicking the button again, the list should now be displayed.

Here, an entry named **Bluething** should appear as well. After clicking the entry, you need to enter the password **1234** to connect the App to the board. This list is displayed via the Block **when Verbinde.BeforePicking**. As you can see by the following block, the button has two functions: Connecting and disconnecting.

```

when Timer.Timer
do
  if not Bluetooth.Enabled or not Bluetooth.IsConnected
  then
    set verbindungStatus.BackgroundColor to [red]
    set verbindungStatus.Text to "Nicht aktiv"
  else
    set verbindungStatus.Text to "Aktiv"
    set verbindungStatus.BackgroundColor to [green]
  
```

There will be a regular check of whether the wireless connection is active or not.



The name and MAC-address of the available devices are displayed.

```

when Verbinde.BeforePicking
do
  if Bluetooth.IsConnected
  then call Bluetooth.Disconnect
  else if not Bluetooth.Enabled
  then call Wenn_BT_An.StartActivity
  else set Verbinde.Elements to Bluetooth.AddressesAndNames
  
```

The button initiates the Bluetooth connection between the board and the App.

The connection itself is established in the block **when Verbinde.AfterPicking**.

The three upper blocks will appear in all Apps in this Advent calendar. The actual function in today's App is implemented in the block **when buttonTon.Click**. In this function, only the value **Sound** is submitted, completed by a \n. A text is sent via the wireless interface using the block **call Bluetooth.SendText**. Submit the text as parameter. Do not forget to define an end criterion. In this example, the end criterion is \n. This is the sign that the Sketch reacts to.

```

when Verbinde.AfterPicking
do
  if call Bluetooth.Connect address Verbinde.Selection
  then
    set macAdresse.Text to Verbinde.Selection
    set Verbinde.BackgroundColor to [red]
    set Verbinde.Text to "Verbindung trennen"
    set verbindungStatus.BackgroundColor to [red]
    set verbindungStatus.Text to "Aktiv"
  
```

The connection to the selected device is established.

Copying blocks to other projects

The area **Blocks** contains a backpack icon. You can use it to copy blocks from the block editor from one project into another. Simply pull the corresponding blocks into the backpack; you can then reuse these blocks in other projects. In order to empty the backpack again, right-click the work area and select **Empty Backpack**.

```

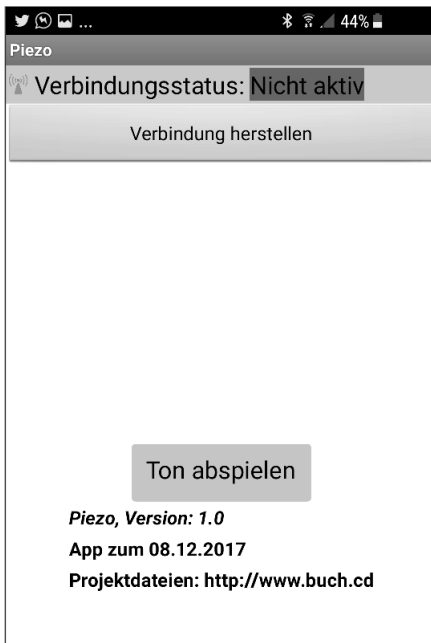
when buttonTon.Click
do
  call Bluetooth.SendText
  text [Ton] join "\n"
  
```

Strings can be connected to each other with the component **join**.

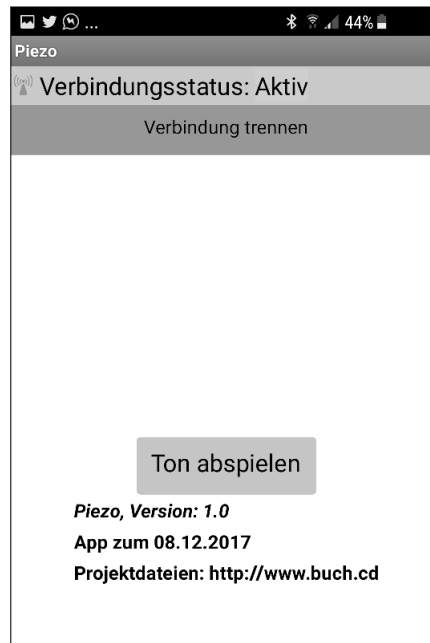
Testing the App

Now start the App via **Connect/AI Companion** and call the App **MIT AI2 Companion** on your Android-Smartphone. Scan the QR-Code or enter the code and start the App via **connect with code**. The App now starts.

Touch **Connect**. Now you can connect to the IoT-board.



The connection is not active yet at first.



The App is now connected to the IoT-board.

If you touch **Play sound** now, the IoT-board will play a melody via the piezo.

9th Day

In the Advent calendar today

• 1 x RGB-LED with dropping resistor

RGB-LEDs

A normal LED is always lit in only one colour. The RGB-LEDs used in the Advent calendar can be lit in different colours. Generally, three LEDs with different colours are installed in a transparent housing here. Each of these three LEDs has its own anode, through which it is connected to a GPIO pin. The cathode, which is connected to the ground line, is only present once. Therefore, an RGB-LED has four connection wires.

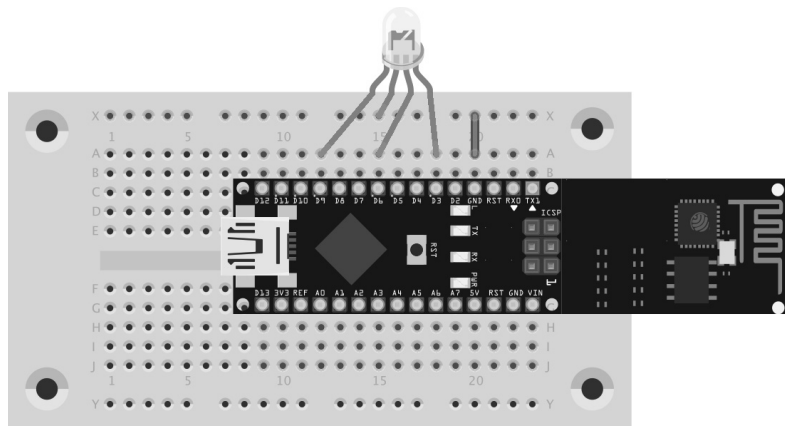
The connection wires of the RGB-LEDs have different lengths to identify them clearly. In contrast to regular LEDs, the cathode is the longest wire here.

RGB-LEDs work like three individual LEDs and therefore also need three 220-Ohm dropping resistors (red-red-brown). The RGB-LEDs in this Advent calendar have them already installed.

Changing the colour of an RGB-LED with the App

Today, you will program an App to change the colour of an RGB-LED.

Components: 1 x board, 1 x RGB-LED with dropping resistor, 1 x jumper



fritzing

The RGB-LED also already has its dropping resistors integrated. The second leg (short leg) is the cathode and must be connected to the ground.

The Sketch for the IoT-board

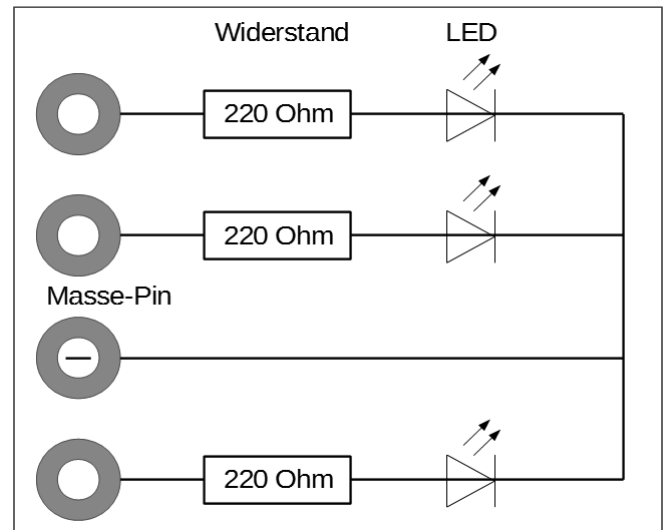
The program for this day is called Tag9.ino and located in directory Tag9. The Sketch reacts to four buttons:

```

if (Text.startsWith("Red") || Text.startsWith("RED") || Text.startsWith("red")){
  rot();
} else if (Text.startsWith("Blue") || Text.startsWith("BLUE") || Text.startsWith("blue")){
  blau();
} else if (Text.startsWith("Green") || Text.startsWith("GREEN") || Text.startsWith("green")){
  gruen();
} else if (Text.startsWith("Off") || Text.startsWith("OFF") || Text.startsWith("off")){
  aus();
}
    
```



Connection pins of an RGB-LED



Circuit diagram for an RGB-LED with 3 dropping resistors



Each colour has its own function within the Sketch. Below, the function `rot()` is presented:

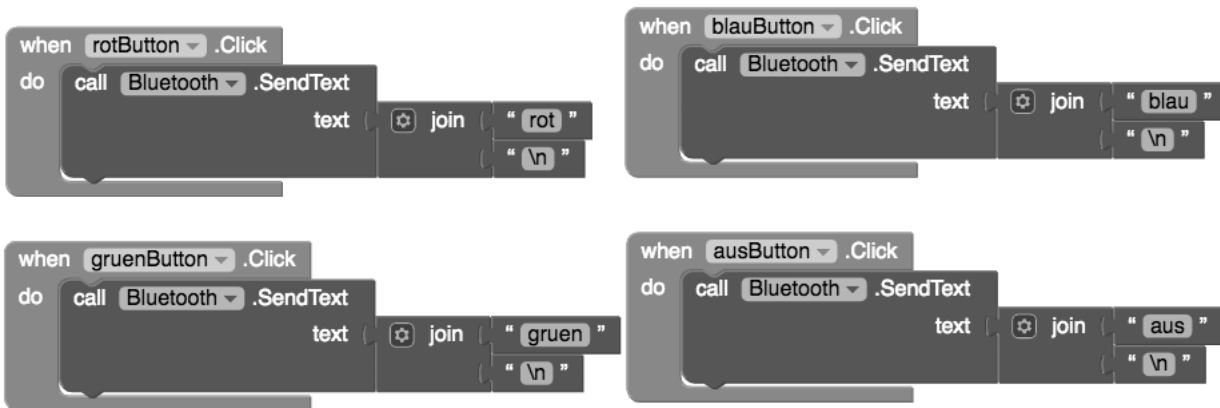
```
void rot() {
  Serial.println(„Red“);
  analogWrite(redPin, HIGH);
  analogWrite(greenPin, LOW);
  analogWrite(bluePin, LOW);
}
```

The App

The associated App is contained in file `RGB.aia`. Import the file into AI2.

Depending on the buttons pushed, a corresponding text is sent via the Bluetooth interface: For red, `red\n` is sent, for blue, `blue\n` is sent, for green, `green\n` is sent and for switching off, `off\n` is sent.

The LED is switched via four buttons.



The blocks in the AI2

10th Day

In the Advent calendar today

• 1 x button

Displaying the push of a button

Today, your IoT-board will react to a mechanical push of a button and send a message to the wireless interface.

Components: 1 x board, 1 x button, 1 x potentiometer, 5 x jumpers (different lengths)

Digital pins can not only output data, e.g. via LEDs, but also be used to enter data. We use a button for input in today's project that is directly connected to the board. The button has four connection pins, with two opposite ones (large distance) being connected to each other from case to case. While the button is pushed, all four connections are connected to each other. In contrast to a switch, a button will not latch. The connections are broken at once when the button is released.

When a +5-V-signal is pending on a digital input, this is evaluated as logically **true**.

If the button were open, the input would not have any clearly defined condition. When a program requests this pin, there may be random results. In order to prevent this, a comparatively very high resistor is connected to the ground. This pull-down-resistor pulls the status of the input pin back down to 0 V when the button is open. Since the resistance is very high, there is no danger of short circuit while the button is pushed either. When the button is pushed, +5 V and the ground line are connected directly via this resistor.

The Sketch

The program for this day is called `Tag10.ino` and located in directory `Tag10`. `digitalRead` is called to evaluate whether the button has been pushed:

```
void loop() {
  int reading = digitalRead(buttonPin);

  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;

      if (buttonState == HIGH) {
        ledState = !ledState;
      }
    }
  }

  digitalWrite(LedPin, ledState);
}
```

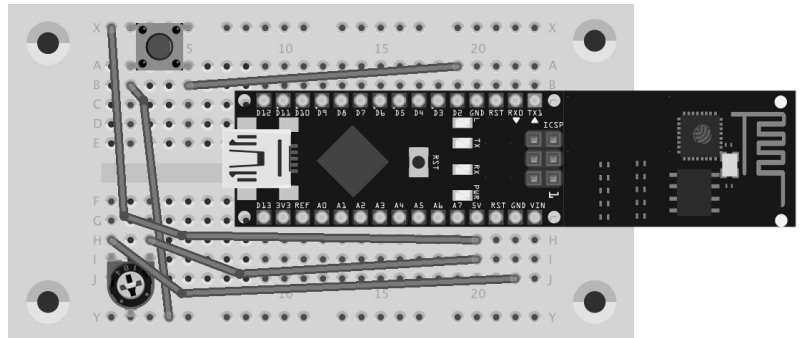
If the button was been pushed, a message will be sent through the wireless interface:

```
if (reading != lastButtonState) {
  HC05.print("Button pushed\n");
}
```

Displaying the reaction of the IoT-board

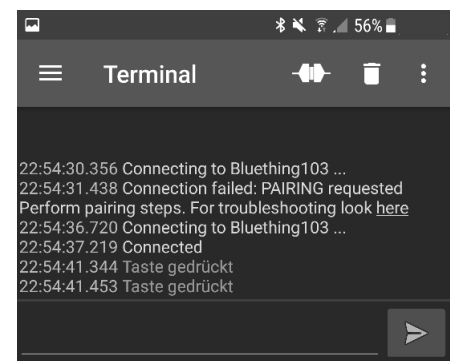
The App **Serial Bluetooth Terminal** that has been used before is used for displaying the push of the button on the Smartphone. After connecting to the IoT-board, the message **Button pushed** will be displayed when the button is pushed.

10. Day



fritzing

The IoT-board is very long; therefore, the button must be placed crosswise on the board, and the jumpers must be partially placed below the connected USB cable.

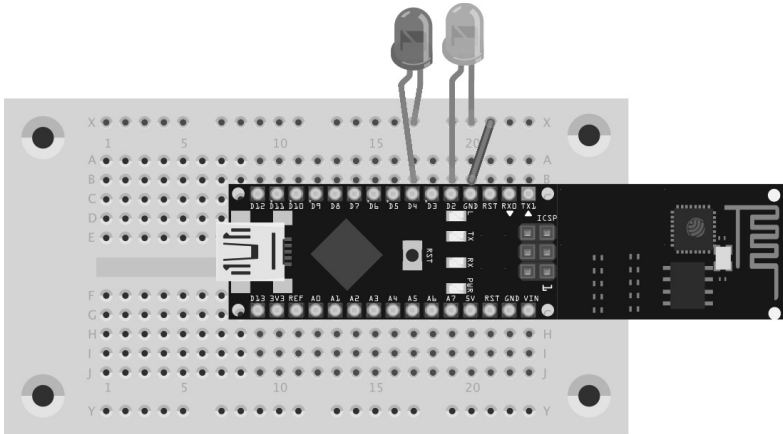


Messages received are written in the terminal window in green.

11. Day

11th Day

In the Advent calendar today
 • 1 x switching wire



fritzing

The integrated dropping resistors make the circuit very compact. LEDs from the left to the right: red and green.

LED echo by App

Today's project is an LED echo. Set a sequence with two interfaces in the App. The LEDs on the board will flash in this sequence.

Components: 1 x board, 1 x LED red with dropping resistor, 1 x LED green with dropping resistor, 1 x jumper

The Sketch

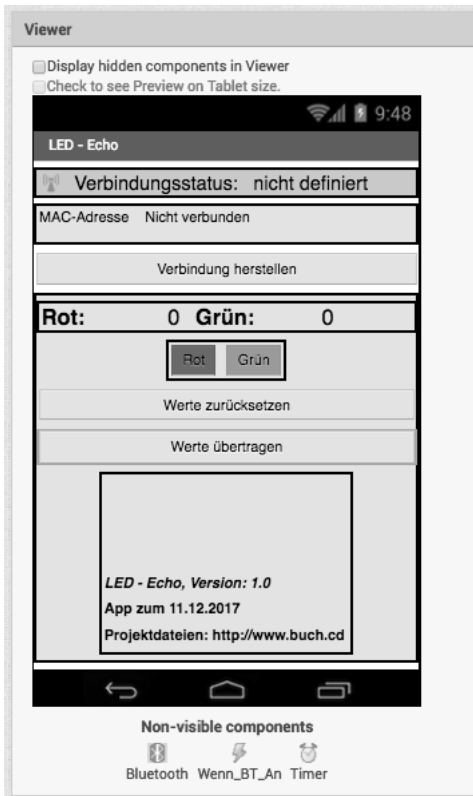
The program for this day is `Tag11.ino` and located in directory `Tag11`. The App determines the echo sequence. The text is output in the following form: RNNNGNN. R means red and G means green, e.g. R5G3 - this would mean having the red LED flash five times and then the green LED three times:

```
if (Text.indexOf("R") != -1 && Text.indexOf("G") != -1) {
    setzeFarbe(Text);
} else if (Text.startsWith("Off") || Text.startsWith("OFF") || Text.startsWith("off")){
    aus();
}
```

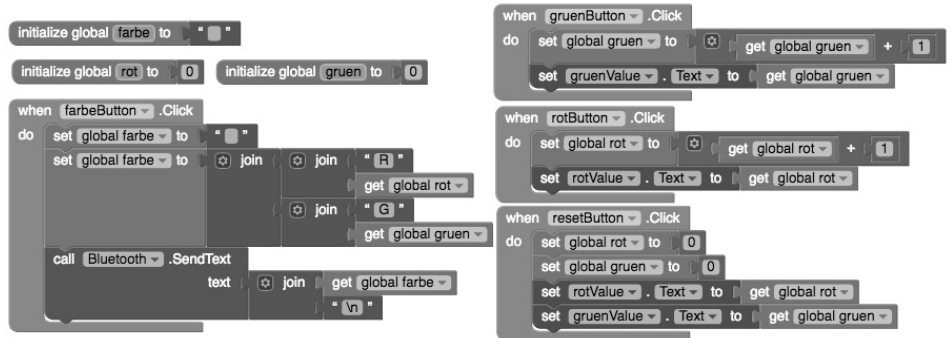
The App

The App has four buttons in addition to the **Connect** button that is already familiar: **Red**, **Green**, **Submit values** and **Reset values**.

When touching **Red** or **Green**, the respective label is increased by one. **Reset values** resets the two values to 0. **Submit values** submits the values to the IoT-board. The value to be submitted is put in interim storage in a global variable. The content of the variables is transmitted via Bluetooth when the button **Submit values** is touched.



The App uses the App already familiar from day 8.



Each button has a separate when.Click-query.

12th Day

In the Advent calendar today

• 1 x LED orange with dropping resistor

Setting the running light speed by App

In today's project, you will control the speed of a running light with the App.

Components: 1 x board, 1 x LED red with dropping resistor, 1 x LED orange with dropping resistor, 1 x LED green with dropping resistor, 1 x jumper

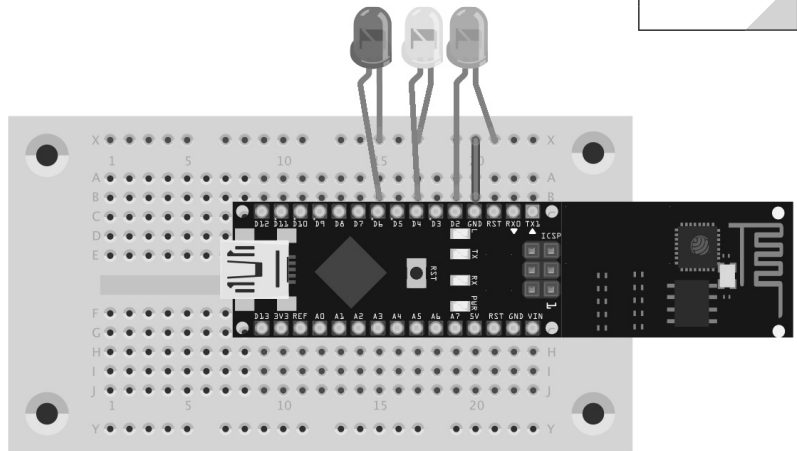
The Sketch

The program for this day is `Tag12.ino` and located in directory `Tag12`. To control the running light, you need two parameters: The waiting time while another LED is lit and the number of passes. These values are sent by the App in a string in the form `wTTTdNN`. For example, `w500d4` means that each LED will be lit for 500 ms and that there will be a 500 ms break before activating another LED. This running light would go through the sequence four times:

```
if (Text.indexOf("w") != -1 && Text.indexOf("d") != -1) {
  String temp = Text.substring(Text.indexOf("w")+1,Text.indexOf("d"));
  int wartezeit = temp.toInt();
  temp = Text.substring(Text.indexOf("d")+1);
  int durchlaeufe = temp.toInt();
  lauflicht(wartezeit, durchlaeufe);
}
```

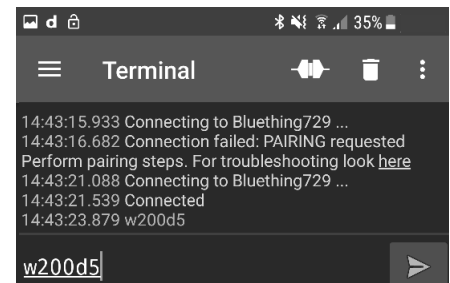
The App

The values are submitted using the App **Serial Bluetooth Terminal**. After connecting, you can send the command directly to the IoT-board in the form as described.



Running light with three LEDs: red, orange and green.

fritzing



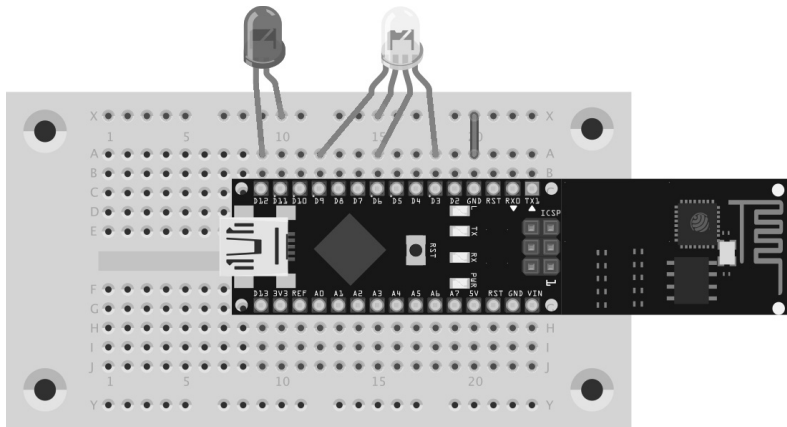
The output shows that two connection attempts are needed for successfully connecting to the IoT-board. If the connection does not work, check the menu item **Bluetooth Devices** to see that you have chosen the right device.

13. Day

13th Day

In the Advent calendar today

- 1 x LED pink with dropping resistor



The pink LED is used to signal that the connection is active.

Adjusting RGB via a slider in the App

In addition to displaying basic colours, an RGB-LED can also display graduations. You can use an App to set the colour of the RGB-LED precisely.

Components: 1 x board, 1 x RGB-LED with dropping resistor, 1 x LED pink with dropping resistor, 1 x jumper

The Sketch

The program for this day is `Tag13.ino` and located in directory `Tag13`. The RGB-LED is controlled via a string in the form `RNN-NGNNNBNNN`. Only when a string in this form is received by the IoT-board will the colour be set via the method `setzeFarbe`. To switch off the LED, the string `off` must be sent:

```
while(HC05.available() > 0){
  Zeichen = HC05.read();
  Text.concat(Zeichen);
  if (Zeichen == ',\n') {
    if (Text.indexOf("R") != -1 && Text.indexOf("G") && Text.indexOf("B") ) {
      setzeFarbe(Text);
    } else if (Text.startsWith("Off") || Text.startsWith("OFF") || Text.startsWith("off"))
    {
      aus();
    }
    Text="";
  }
}
```

In the function `setzeFarbe`, the submitted text is broken down into parts with the `substring` method and the values are then written via `analogWrite`:

```
void setzeFarbe(String text) {
  if (text.indexOf("R") != -1 && text.indexOf("G") && text.indexOf("B") ) {
    String temp = text.substring(text.indexOf("R")+1,text.indexOf("G"));
    if (temp.indexOf(".") != -1) {
      temp = temp.substring(0, temp.indexOf("."));
    }
    int rot = temp.toInt();

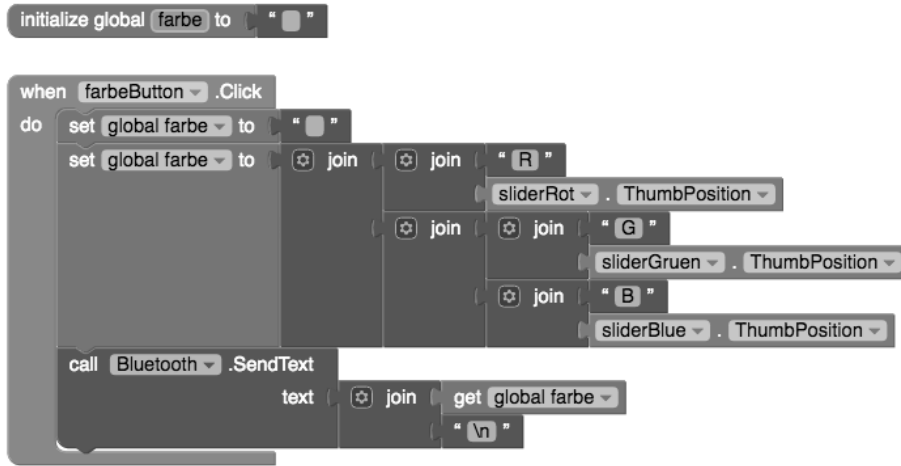
    temp = text.substring(text.indexOf("G")+1,text.indexOf("B"));
    if (temp.indexOf(".") != -1) {
      temp = temp.substring(0, temp.indexOf("."));
    }
    int gruen = temp.toInt();

    temp = text.substring(text.indexOf("B")+1);
    if (temp.indexOf(".") != -1) {
      temp = temp.substring(0, temp.indexOf("."));
    }
    int blau = temp.toInt();
    analogWrite(redPin,rot);
    analogWrite(greenPin,gruen);
    analogWrite(bluePin, blau);
  }
}
```

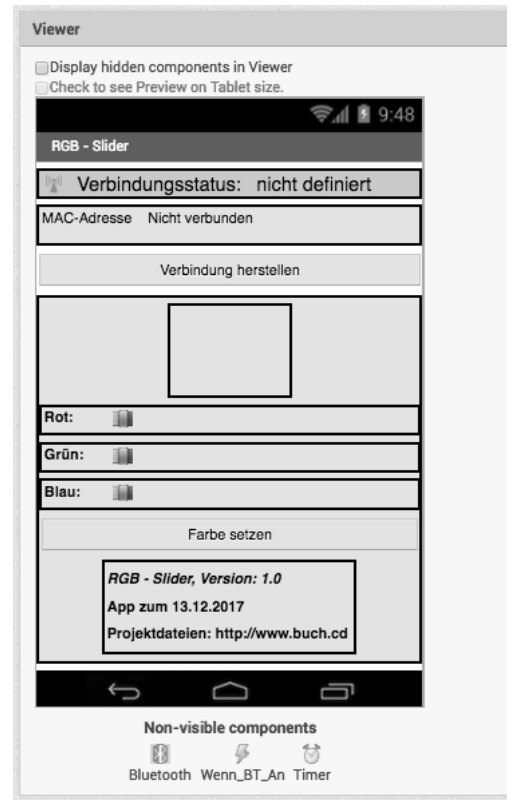
The App

The development environment for today's App is the AI2. For this, import the file RGB_Slider.aia. Use three sliders to set the value for the respective colour. Push **Set colour** to send the value of the three sliders to the IoT-board.

Touching **Set colour** will cause the text to be submitted to be composed in the variable farbe. `Bluetooth.SendText` sends the text via the wireless interface.



The string is composed via **join**, which can also be nested.



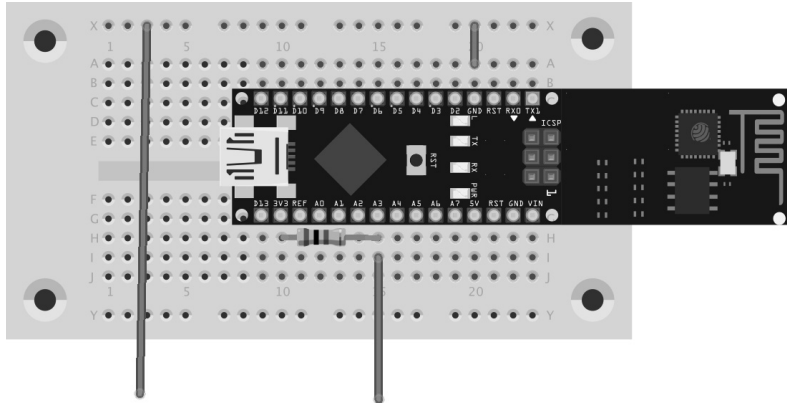
Each slider is combined with a label in a **HorizontalArrangement**. To place all sliders on top of each other, the width of the three labels has been set to 15%.

14. Day

14th Day

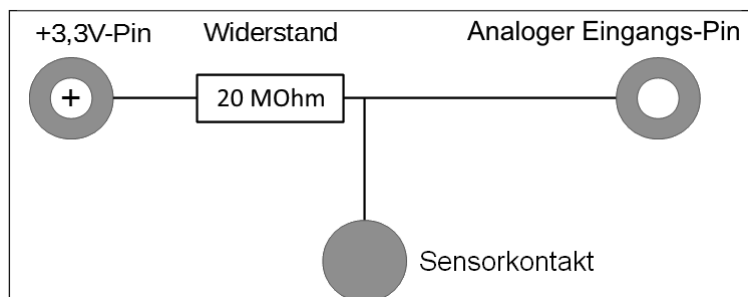
In the Advent calendar today

- 1 x modelling clay red
- 1 x resistor 20 MOhm



fritzing

The end of the two protruding jumpers goes into a piece of modelling clay each. Best shape it into a small, round ball. The modelling clay contact at A3 is used as the input.



Circuit diagram for sensor contacts on the IoT-board

Modelling clay contact

Today, you will use the enclosed modelling clay as an input and change the background colour of an area in the App when the modelling clay is touched.

Components: 1 x board, 2 x pieces of modelling clay, 1 x resistor 20 MOhm, 3 x jumper (different lengths)

This is how sensor contacts work:

The pin switched as input is connected to +3.3 V via an extremely high-Ohmic resistance (20 MOhm), so that a weak signal that is still clearly defined as high is pending. A person who is not floating freely in the air is always grounded and supplies a low level through electrically conductive skin. When this person touches a sensor contact, the weak high signal is overlaid by the much higher low level of the fingertip, pulling the pin to low level.

The actual height of the resistance between hand and ground depends on many things, including shoes and floor. Barefoot in wet grass offers the best ground connection, but stone floors usually work well, too. Wood floors insulate more strongly, and plastic floorings often even are positively charged. For the circuit to work at all times, an additional ground contact is installed in each circuit, similarly to sensor buttons at elevators and doors. When it and the actual sensor are touched at the same time, the ground connection is made in any case.

Putty conducts current about as well as the human skin does. It can easily be formed in any desired shape, and a putty contact

is much easier to grasp than a simple piece of wire. The area on which the hand touches the contact is much larger. Thus, a "loose contact" is less likely to occur. Cut a piece of about 10 cm from the switching wire, remove the insulation for about 1 cm on either end and push one end into a piece of modelling clay. Push the other end into the board as illustrated.

The IoT-board has analogue inputs that are very suitable for sensor contacts. Analogue inputs supply values between 0 (low level) and 1023 (high level): Values between 100 and 200 are good limits to tell apart contacted and non-contacted sensor contacts.

The Sketch

The program for this day is `Tag14.ino` and located in directory `Tag14`. The actual function in this sketch can be programmed with a few lines. `analogRead` reads in the value; if it is below the threshold, a text will be sent via the wireless interface with `HC05.print`:

```
void loop() {
  sensorValue = analogRead(sensorPin);

  if (sensorValue < 256) {
    HC05.print("LOW\n");
    delay(1000);
  }
}
```

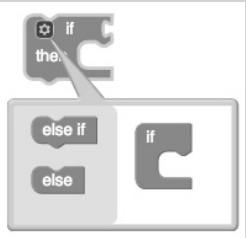
The App

Today's App `knete_Schalter.aia` contains a label that has an orange background colour at first.

To check if a text has been sent via the Bluetooth interface, it must be periodically reviewed whether a text is present. This is done with the existing timer.

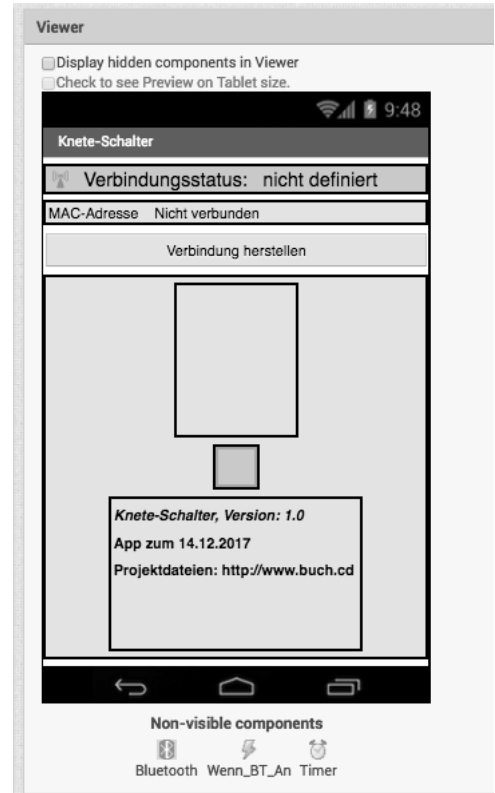
Configuring blocks

The elements in the area **Blocks** in the block editor can partially be configured. The configuration takes place using the blue cogwheel at the component.



The if-block also supports else and else-if.

Then click the element you want to use. The component will change.



To make the coloured label well visible, its width and height have been set to 30 pixels.

The timer has only been used to display the status of the wireless connection so far.

`Bluetooth.ReceiveText` is used to read the text from the wireless interface.

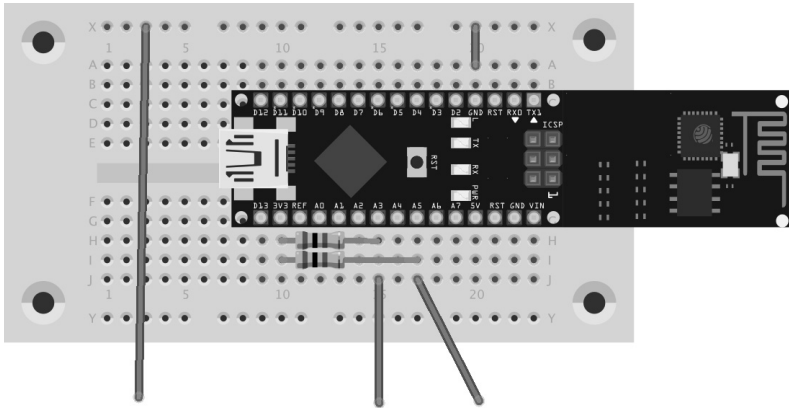
You can select the colour to be set in a selection field.

15. Day

15th Day

In the Advent calendar today

• 1 x resistor 20 MOhm



fritzing

The ends of the three protruding jumpers go into a piece of modelling clay each. You need two high-Ohmic resistors due to the two modelling clay contacts.

Differentiable modelling clay contacts

Now use two modelling clay contacts and display in the App which contact has been pressed.

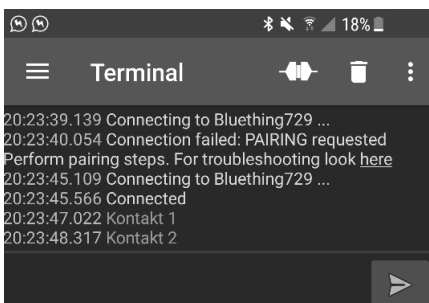
Components: 1 x board, 3 x pieces of modelling clay, 2 x resistor 20 MOhm, 4 x jumper (different lengths)

The Sketch

The program for this day is `Tag15.ino` and located in directory `Tag15`. As on day 14, activation of the modelling clay contact is read via `analogRead`. There is a separate variable for each modelling clay contact. `HC05.print` then transfers the command to the wireless interface.

```
void loop() {
  sensorValue1 = analogRead(sensorPin1);
  sensorValue2 = analogRead(sensorPin2);

  if (sensorValue1 < 256) {
    HC05.print(„Kontakt 1\n“);
  }
  if (sensorValue2 < 256) {
    HC05.print(„Kontakt 2\n“);
  }
  if (sensorValue1 < 256 || sensorValue2 < 256) {
    delay(1000);
  }
}
```

**The App**

The texts are displayed using the App **Serial Bluetooth Terminal**. First, connect to the IoT-board; the texts should appear once you touch a modelling clay contact.

Activating a modelling clay contact causes the programmed output in the terminal.

16th Day

In the Advent calendar today

• 1 x flashing LED red with dropping resistor



Controlling the flashing LED with the App

Switch the flashing LED on with a button and off again with another button.

Components: 1 x board, 1 x flashing LED red with dropping resistor

The Sketch

The program for this day is Tag16.ino and located in directory Tag16. The Sketch reacts to on and off from the wireless interface:

```
if (Text.startsWith("On") || Text.startsWith("ON") || Text.startsWith("on")){
  an();
} else if (Text.startsWith("Off") || Text.startsWith("OFF") || Text.startsWith("off")){
  aus();
}
}
```

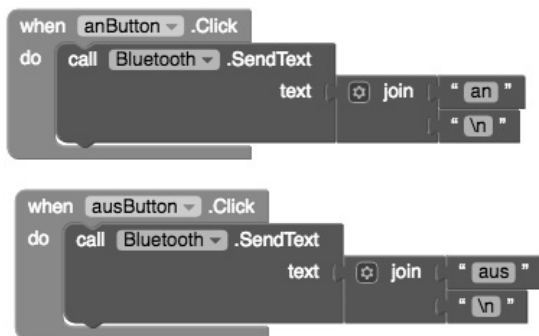
The flashing LED is switched in two separate functions. Only the value LOW or HIGH is written via digitalWrite.

```
void an() {
  Serial.println("An");
  digitalWrite(blinkPin, HIGH);
}
void aus() {
  Serial.println("Aus");
  digitalWrite(blinkPin, LOW);
}
}
```

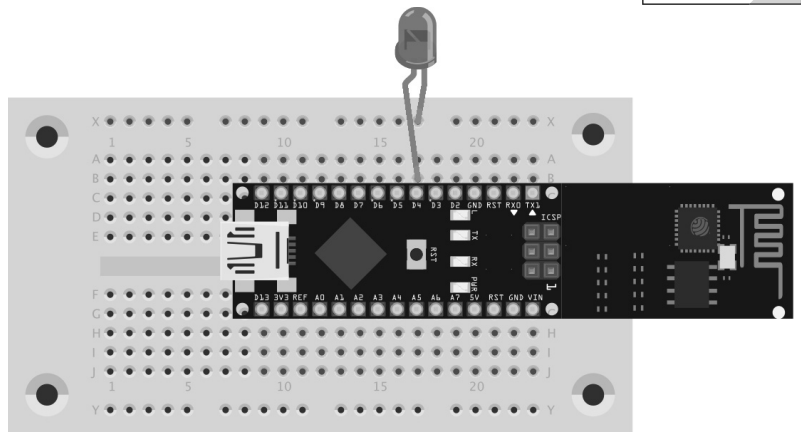
The App

The App Blinker.aia has two buttons for switching the flashing LED on and off.

The already-familiar method call Bluetooth.SendText is used for sending.

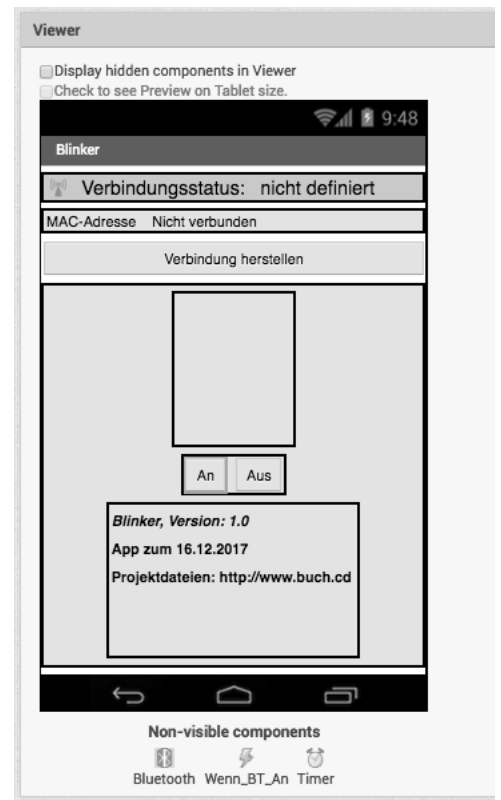


Remember that you need to attach the end criterion to the string when sending.



fritzing

The flashing LED also has a dropping resistor and therefore does not need any further elements for the circuit.

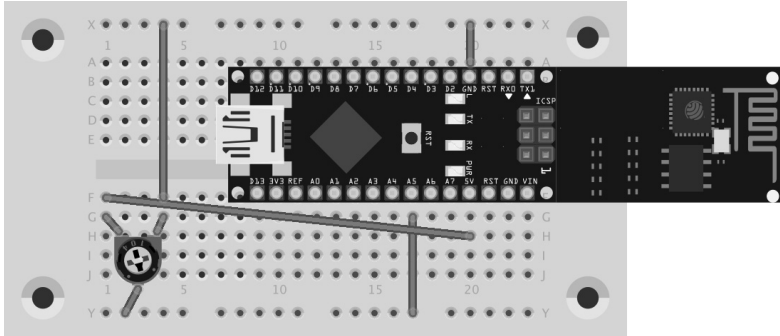


There is a VerticalArrangement above the two buttons (on and off) as a positioning element.

17th Day

In the Advent calendar today

• 1 x potentiometer



fritzing

A potentiometer is just an analogue sensor from the program's point of view.

```
int sensorPin = A5;
int sensorValue = 0;
```

In the method `loop`, the value is read in via `analogRead` and passed on to the wireless interface. Then the program will wait for 2000 ms before reading the value again:

```
void loop() {
  sensorValue = analogRead(sensorPin);
  HC05.print("value potentiometers = ");
  HC05.println(sensorValue);
  delay(2000);
}
```

The App

The texts are displayed using the App **Serial Bluetooth Terminal**. First, connect to the IoT-board. You will see the potentiometer value then.

Display of the resistor value

Today, you will measure the value of a potentiometer via an analogue input and can output the value in an App.

Components: 1 x board, 1 x potentiometer 15 kOhm, 4 x jumper (different lengths)

The Sketch

The program for this day is `Tag17.ino` and located in directory `Tag17`. First, the two variables `sensorPin` and `sensorValue` are defined. The potentiometer is connected to the analogue input 5, and therefore, `sensorPin` is set to `A5`:

18th Day

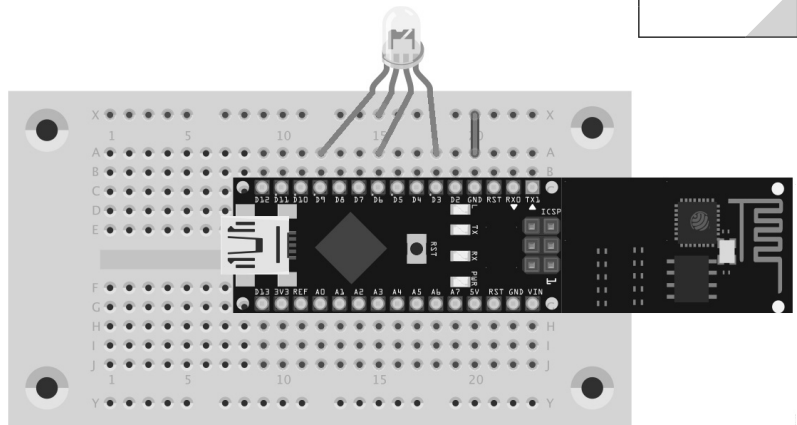
In the Advent calendar today

• 1 x jumper cable

RGB-running light

This App controls the flashing duration of an RGB-LED. The LED will flash in alternating colours. You can also stop its flashing again.

Components: 1 x board, 1 x RGB-LED with dropping resistor, 1 x jumper cable



The Sketch

The program for this day is Tag18.ino and located in directory Tag18. The Sketch reacts to the text stop and to a set number. stop will switch off the RGB-LED. If a number is received, it will be used as the waiting time between switching one of the three LEDs of the RGB-LED on or off:

```
if (Zeichen == ',\n') {
  Serial.println(Text);
  if (Text.startsWith("stop")){
    aus();
    blinkDauer = 0;
  } else {
    blinkDauer = leseBlinkdauer(Text);
  }
}
```

The integrated dropping resistors make the circuit very compact.

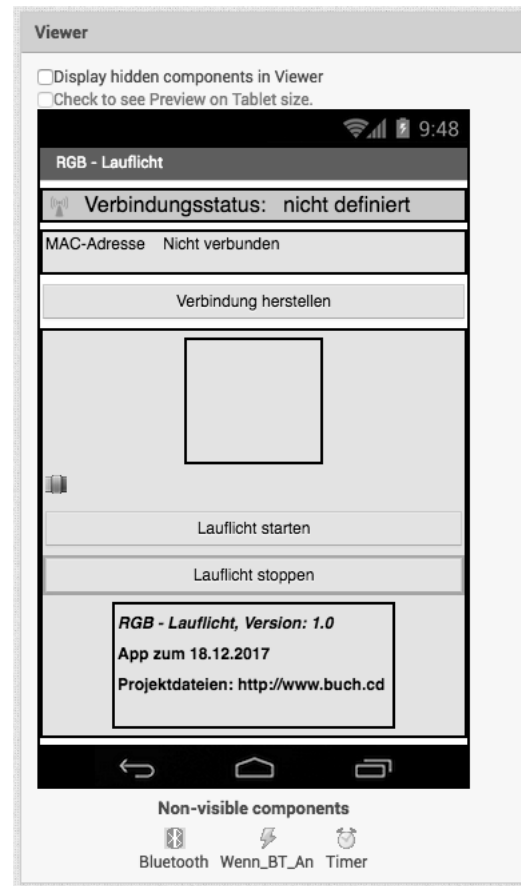
The App

The App RGB_Lauflicht.aia has a slider and two buttons for switching the flash on and off.

The command Bluetooth.SendText that is already familiar sends the waiting time or the text stop via the wireless interface.



The waiting time is not separately coded but only the value of the slider is sent, followed by a final \n.



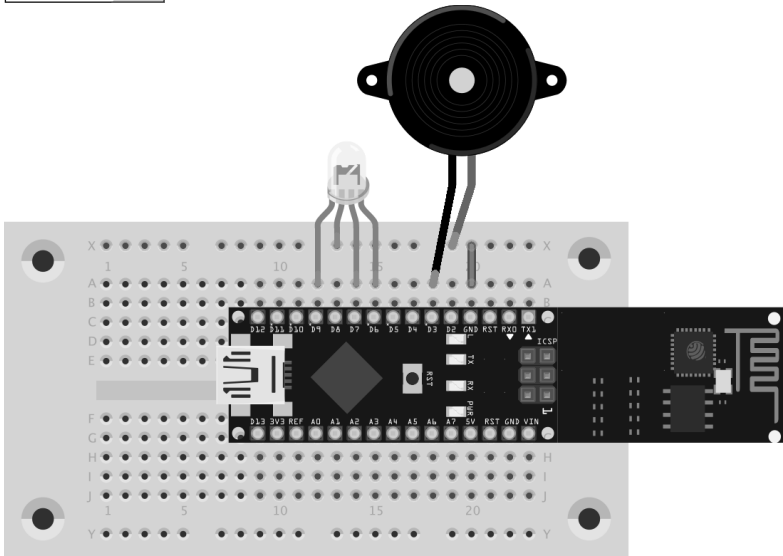
The minimum value of the slider is 0 and its maximum value is 2000. The values represent the waiting time in milliseconds.

19. Day

19th Day

In the Advent calendar today

- 1 x jumper cable



App to select hardware Apps

Today, we will combine two functions in one circuit: Playing a sound with the piezo and flashing of an RGB-LED. Use the App to select the corresponding function.

Components: 1 x board, 1 x RGB-LED with dropping resistor, 1 x piezo, 1 x jumper cable

The Sketch

The program for this day is `Tag19.ino` and located in directory `Tag19`. First, you need to define the proper pins as variables:

```
//---- Pins of the RGB-LED
int redPin = 9;
int greenPin = 7;
int bluePin = 8;

//---- Pin Piezo
int piezo = 3;
```

fritzing

The RGB-LED doesn't need any dropping resistors here either. Use the jumper cable to connect to the ground.

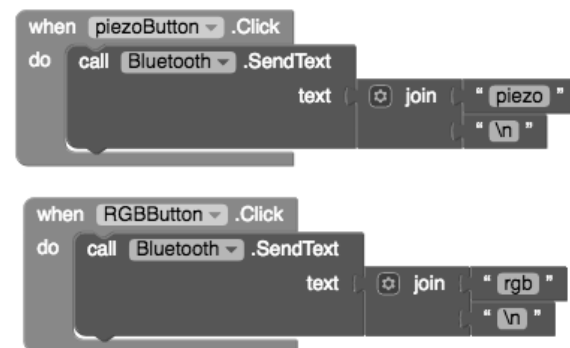
The Sketch reacts to the commands `piezo` and `rgb`. The methods have been taken from the previous Sketches:

```
if (Text.startsWith("piezo")){
  aus();
  playMelody();
} else if (Text.startsWith("rgb")) {
  blinken(500);
};
```

The App

The App `App_Wechsler.aia` has two buttons to switch between the piezo and RGB.

The function is mapped in two `when.Click` blocks. Only the text to be sent is adjusted.



This App uses the already-familiar elements from the other Apps for connecting. Use **Connect** to establish the connection.

The blocks are very similar. In such cases, watch that you select the right button in the `when.Click` block.

20th Day

In the Advent calendar today

• 1 x NTC

Heat sensor in the App

Today's project uses an NTC for temperature measuring. You can test this project well, e.g. by approaching a candle on the Advent wreath with the NTC.

Components: 1 x board, 1 x NTC, 1 x potentiometer
15 kOhm, 6 x jumper (different lengths)

The Sketch

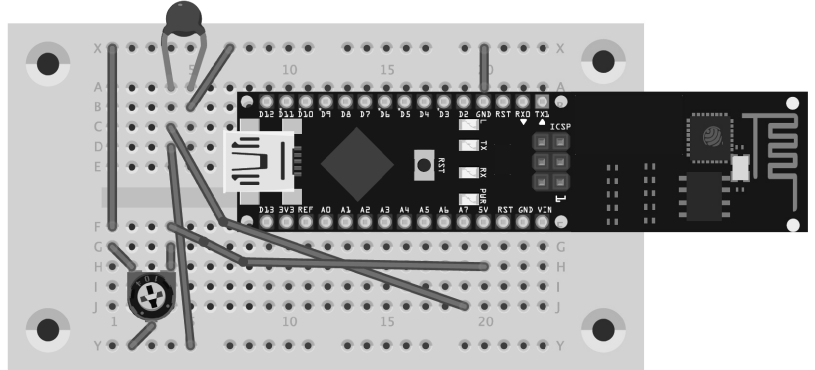
The program for this day is `Tag20.ino` and located in directory `Tag20`. The Sketch is set up very simply. After calling `analogRead`, the value will be sent to the App by Bluetooth; then there will be a pause of 2 seconds and the value will be read again and sent:

```
void loop() {
  sensorValue = analogRead(sensorPin);
  HC05.print("Value of the NTC = ");
  HC05.println(sensorValue);
  delay(2000);
}
```

The App

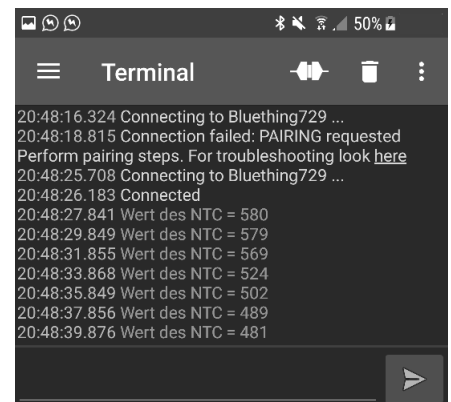
The texts are displayed using the App **Serial Bluetooth Terminal**. First, connect to the IoT-board. You will see the NTC value then. When the values grow smaller, this indicates increasing temperature.

20. Day



fritzing

Do not forget to use the 10-kOhm-resistor; you will not be satisfied with the measuring results otherwise.



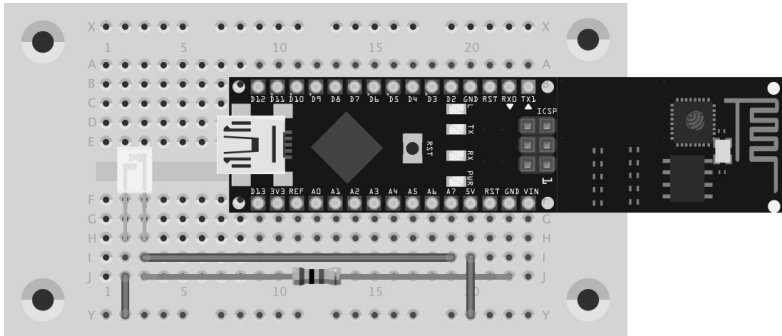
As in the other projects, the data received are displayed in green text. Do not forget to call `delay`, in the Sketch. You will otherwise receive too many values.

21. Day

21st Day

In the Advent calendar today

- 1 x photo transistor
- 1 x resistor 1 kOhm



fritzing

The setup is similar to that of day 20. Caution: You are using a different resistor and the connection with VCC (+5 V) is important.

Measuring brightness and darkness in the App

Now you have another sensor for your projects. You can use the photo transistor to display brightness and darkness. You will learn how to today. You can use this knowledge to build a light barrier, which will give you an alarm system with a display in an App.

Components: 1 x board, 1 x photo transistor, 1 x resistor 1 kOhm, 3 x jumper (different lengths)

The Sketch

The program for this day is `Tag21.ino` and located in directory `Tag21`. The sketch is similar to the sketch from the previous day. If you want to see more values on your Smartphone, you must reduce the waiting duration when calling `delay`.

```
void loop() {
  sensorValue = analogRead(sensorPin);
  HC05.print("Phototransistor value = ");
  HC05.println(sensorValue);
  delay(2000);
}
```

The App

The texts are displayed using the App **Serial Bluetooth Terminal**. First, connect to the IoT-board. You will see the phototransistor value then. Look at how the values change in light and darkness.

22nd Day

In the Advent calendar today

- 1 x moisture sensor
- 1 x resistor 1 kOhm

Moisture measurement

Today, you get another sensor: a moisture sensor. You can use it to measure moisture.

Components: 1 x board, 1 x moisture sensor, 1 x resistor 1 kOhm, 3 x jumper (different lengths)

The Sketch

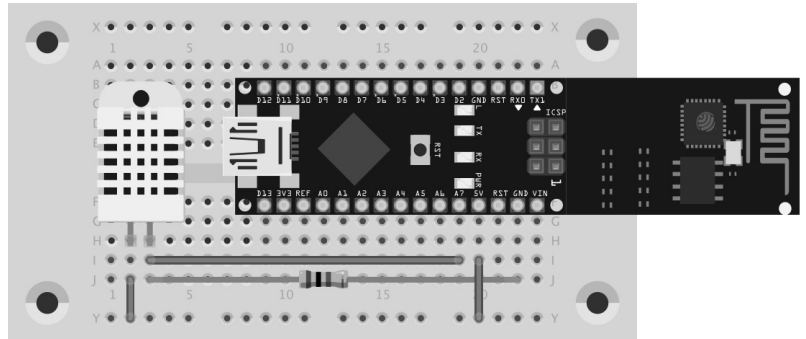
The program for this day is `Tag22.ino` and located in directory `Tag22`. As on the previous days, the sensor is read via `analogRead` and the value is passed on to the wireless interface:

```
void loop() {
  sensorValue = analogRead(sensorPin);
  HC05.print("Moisture sensor value= ");
  HC05.println(sensorValue);
  delay(2000);
}
```

The App

The texts are displayed using the App **Serial Bluetooth Terminal**. First, connect to the IoT-board. You will see the moisture sensor value then. Take the moisture sensor between two fingers and observe how the values develop.

22. Day



fritzing

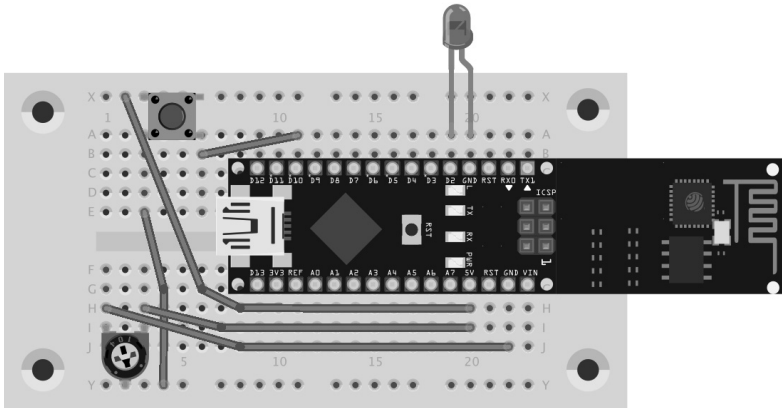
The circuit is similar to the circuit from day 21. Only the sensor needs to be swapped.

23. Day

23rd Day

In the Advent calendar today

• 1 x jumper cable



fritzing

For the button to work well, a voltage divider is set up via the potentiometer. You can also replace the jumpers with the jumper cables.

Code breaker

Today's project is a little game: You will set a code in an App. Then you will submit this code to the IoT-board. Now the code must be entered with the button. If the code has been entered correctly, the red LED will light up and the App will display that the code has been entered correctly. Otherwise, the red LED will not light up and the App will display that the code has not been entered correctly.

Components: 1 x board, 1 x button, 1 x LED red with dropping resistor, 1 x potentiometer 15 kOhm, 5 x jumper (different lengths)

The Sketch

The program for this day is `Tag23.ino` and located in directory `Tag23`. The code is submitted in the form of C++:

```
if (Text.startsWith("C")){
    anzahlTasten = Text.substring(Text.
    indexOf("C")+1).toInt();
```

All in all, the player has five seconds time to enter the code:

```
interval = millis() - start;
Serial.println(interval);
while (interval < 5000) {
    pressed += tasterAuslesen();
    delay(100);
    interval = millis() - start;
}
```

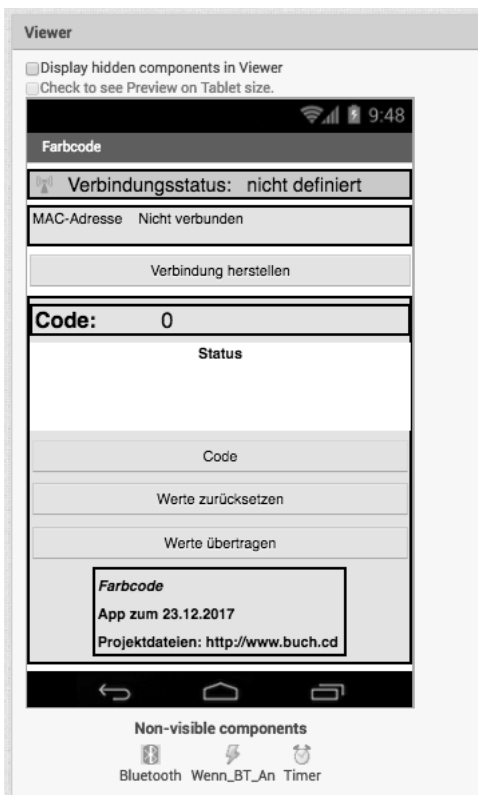
The method `tasterAuslesen` will check whether a button has been pushed. The program will wait for 100 ms between checks. The result is returned to the App:

```
if (pressed == anzahlTasten) {
    HC05.print("JA");
    digitalWrite(redPin, HIGH);
    delay(5000);
    digitalWrite(redPin, LOW);
} else {
    HC05.print("NEIN");
}
```

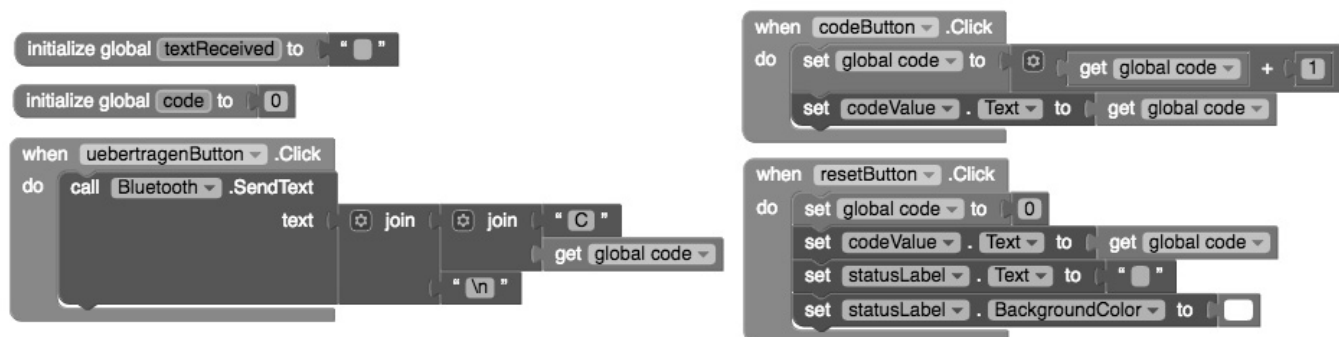
The App

The App `Farbcode.aia` has three buttons and two labels. One label will display the code that has been entered and the second one the game status. If the code has been entered correctly via the hardware button, the background colour of the game status label is set to green; otherwise, it will be red.

The code is put in interim storage in a global variable and then submitted by `Bluetooth.SendText`.

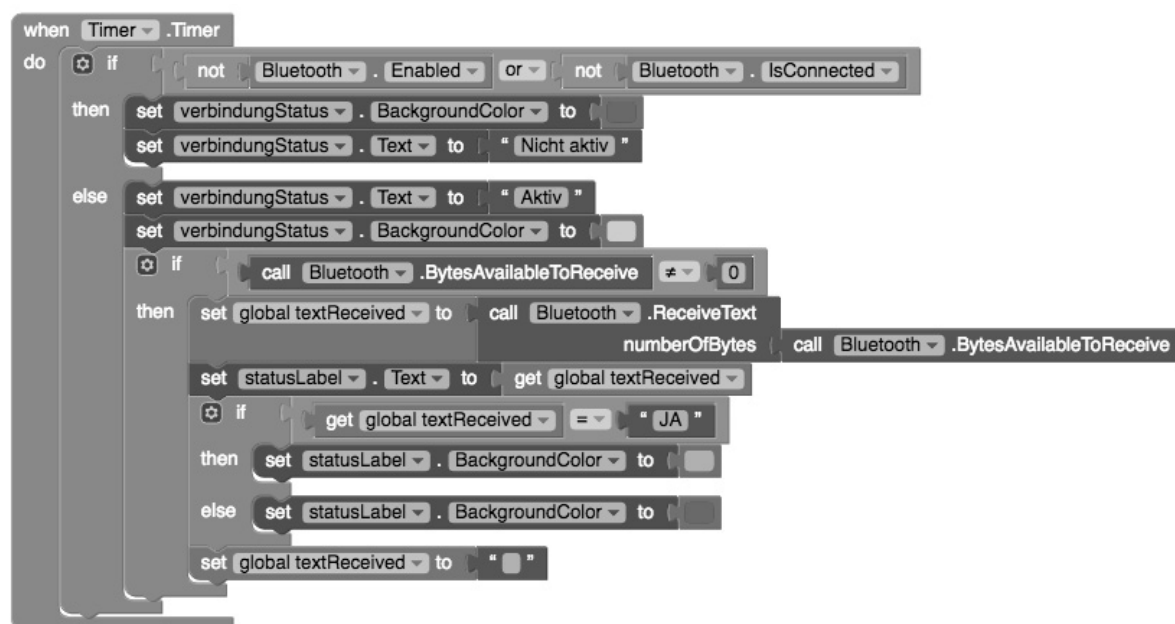


A coloured label shows whether the code has been entered correctly via the hardware button or not.



Since three buttons are needed for this game, there are also three when.Click blocks.

The texts from the IoT-block are received in the periodic timer. The timer that was used for displaying the connection status is used for this.



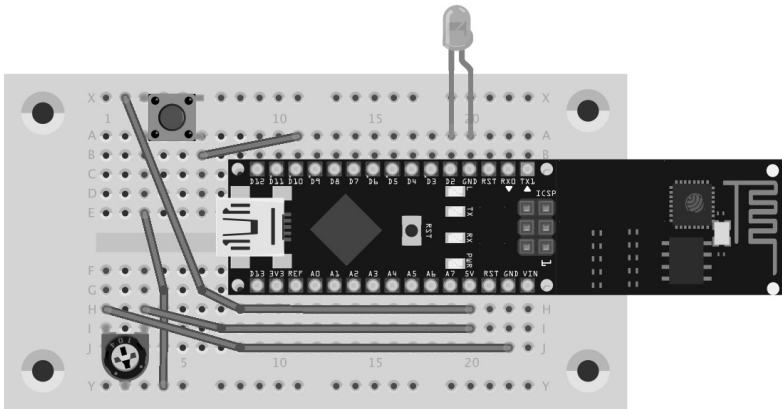
If YES is received, the background colour is switched to green; otherwise, it will be red.

24. Day

24th Day

In the Advent calendar today

• 1 x button



fritzing

Today's circuit is similar to that of the previous day. Only the red LED has been replaced by a green LED.

Reaction game

We will end this Advent calendar with a little reaction game. You can start the game with a button in the App. The LED on the board comes on; once the LED goes out, you need to push the button. The duration is displayed on the App.

Components: 1 x board, 1 x button, 1 x LED green with dropping resistor, 1 x potentiometer 15 kOhm, 5 x jumper (different lengths)

The Sketch

The program for this day is `Tag24.ino` and located in directory `Tag24`. The function `millis` measures the duration until the button is pushed:

```
if (Text.startsWith("START")){

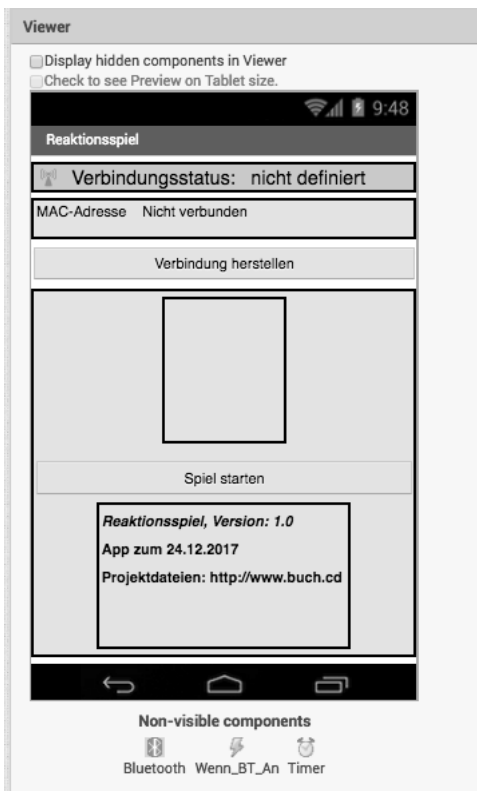
    digitalWrite(greenPin, HIGH);
    delay(1000);
    digitalWrite(greenPin, LOW);

    //Starting time
    int dauer = millis();

    Serial.println(interval);
    while (tasterAuslesen() != 1) {
        delay(100);
    }
    dauer = millis() - dauer;
    HC05.print(dauer);
    digitalWrite(greenPin, LOW);
}
```

The App

The App `Reaktionsspiel.aia` has a button called **Start game**. Pushing this button sends the text via the wireless interface. The text arriving from the IoT-board is displayed in the label above the button.



Above the button **Start game**, there is a label without text. Only when a text from the IoT-board comes in will the text in the label be set.

To start the game, the block when `startButton.click` is evaluated when the button **Start game** is clicked, and the text `START` is submitted via the wireless interface.

```

initialize global textReceived to ""

when startenButton.Click
do call Bluetooth.SendText
   text "START\n"

when Timer.Timer
do if not Bluetooth.Enabled or not Bluetooth.IsConnected
then
  set verbindungStatus.BackgroundColor to 
  set verbindungStatus.Text to "Nicht aktiv"
else
  set verbindungStatus.Text to "Aktiv"
  set verbindungStatus.BackgroundColor to 
  if call Bluetooth.BytesAvailableToReceive != 0
  then
    set global textReceived to call Bluetooth.ReceiveText
      numberOfBytes call Bluetooth.BytesAvailableToReceive
    set dauerLabel.Text to get global textReceived
    set global textReceived to ""
  
```

The wireless interface should always be read out with a timer. You cannot tell precisely when a data package will arrive.

Merry Christmas!