Workshop
Arduino-Programmierung #5

WS2812 smart LEDs, FastLED, Reaktionsspiel

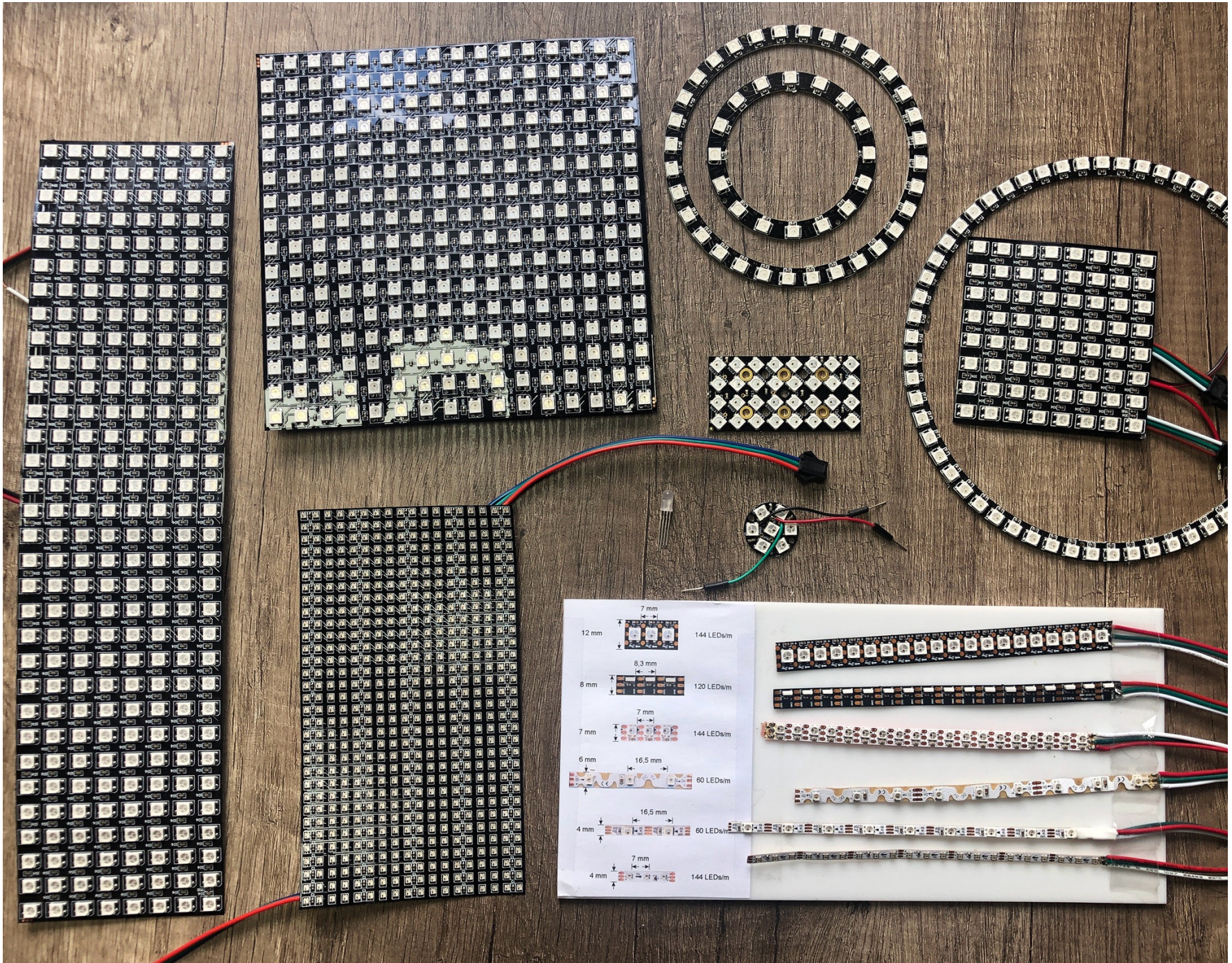
Joachim Baur

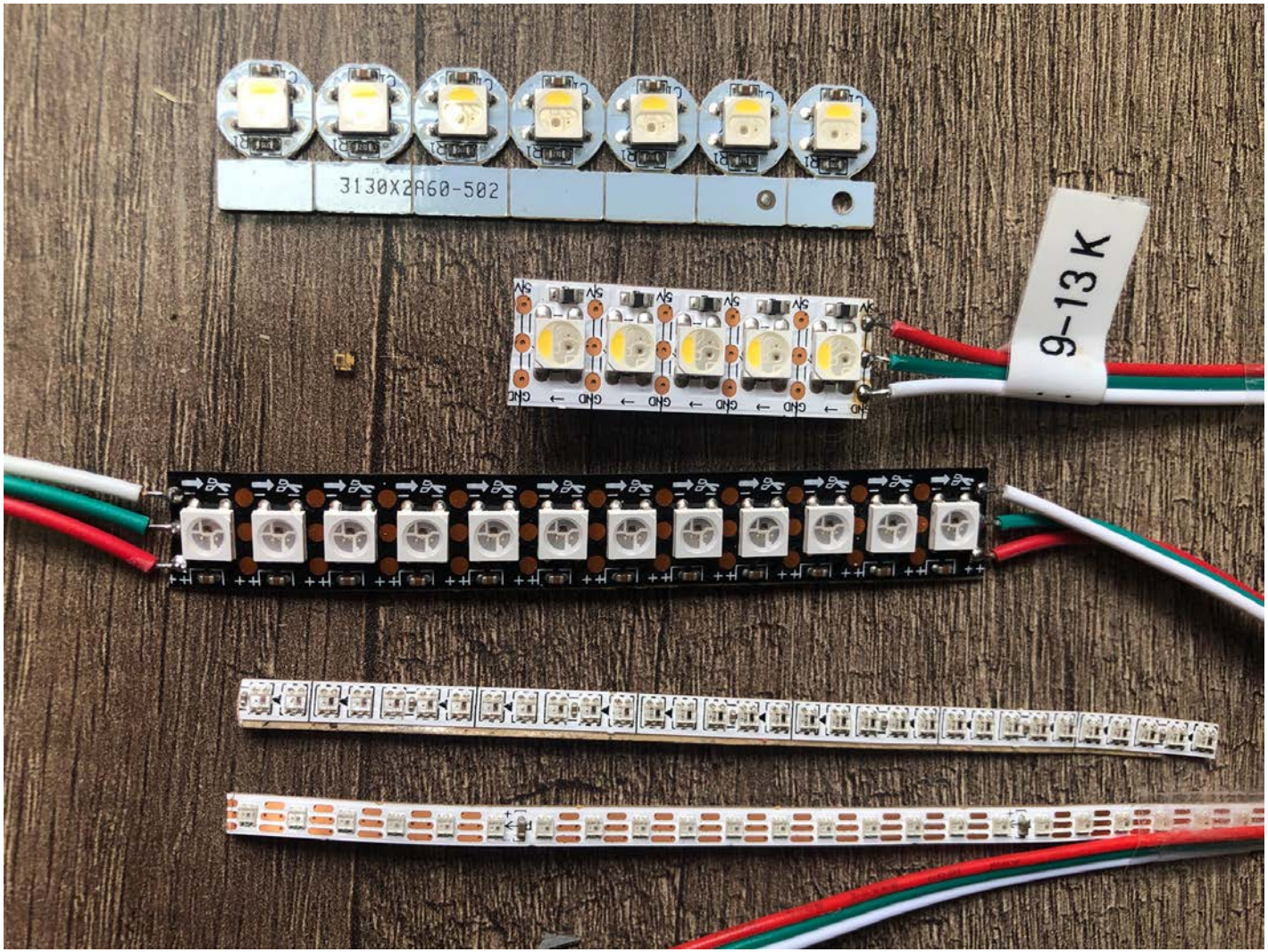
E-Mail: post@joachimbaur.de

ZTL-Alias: @joachimbaur

Download für diesen Workshop: www.joachimbaur.de/WS5.zip

Smart RGB LEDs (WS2812 „Neopixel“)





LED Treiber Chip:
WS2812B
SK6812 (neuer)

5V (i.d.R, auch 12V)

Nur 3 Leitungen:
- +5V (hier rot)
- DATA (hier grün)
- GND (hier weiß)

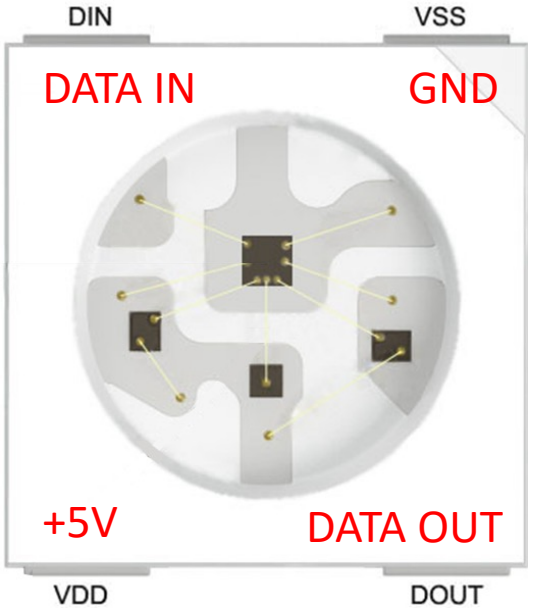
alle LEDs in Reihe, trotzdem einzeln ansprechbar

RGB und RGBW (RGBW mit diversen Weiß-Varianten kalt/neutral/warmweiß)
Strips mit 5050 LEDs, 3535 LEDs oder 2020 LEDs (Maße in 1/10 mm)
Strips können getrennt und mit Kabeln verlängert/verbunden werden

https://www.pololu.com/category/180/sk6812-ws2812b-based-led-strips

Funktionsweise der Ansteuerung

- Die RGB-Werte für **alle** LEDs werden nacheinander mit einer festen Taktrate über die DATA-Leitung an die angeschlossenen LEDs gesendet (RGB RGB RGB…)
- Jeder LED-Treiber (SK6812 Chip)
  - nimmt am DIN-Pin alle RGB-Werte nacheinander entgegen
  - trennt die ersten 3 Werte (R, G, B) ab, um die eigenen On-Chip-LEDs entsprechend anzusteuern
  - sendet alle restlichen RGB-Werte über die DOUT-Leitung an die nächste LED in der Kette weiter
  - Hält die zuletzt empfangenen Werte, bis neue kommen



- Es müssen daher IMMER ALLE LEDs in der Kette aktualisiert werden, auch wenn sich nur die Farbe einer einzelnen LED ändert

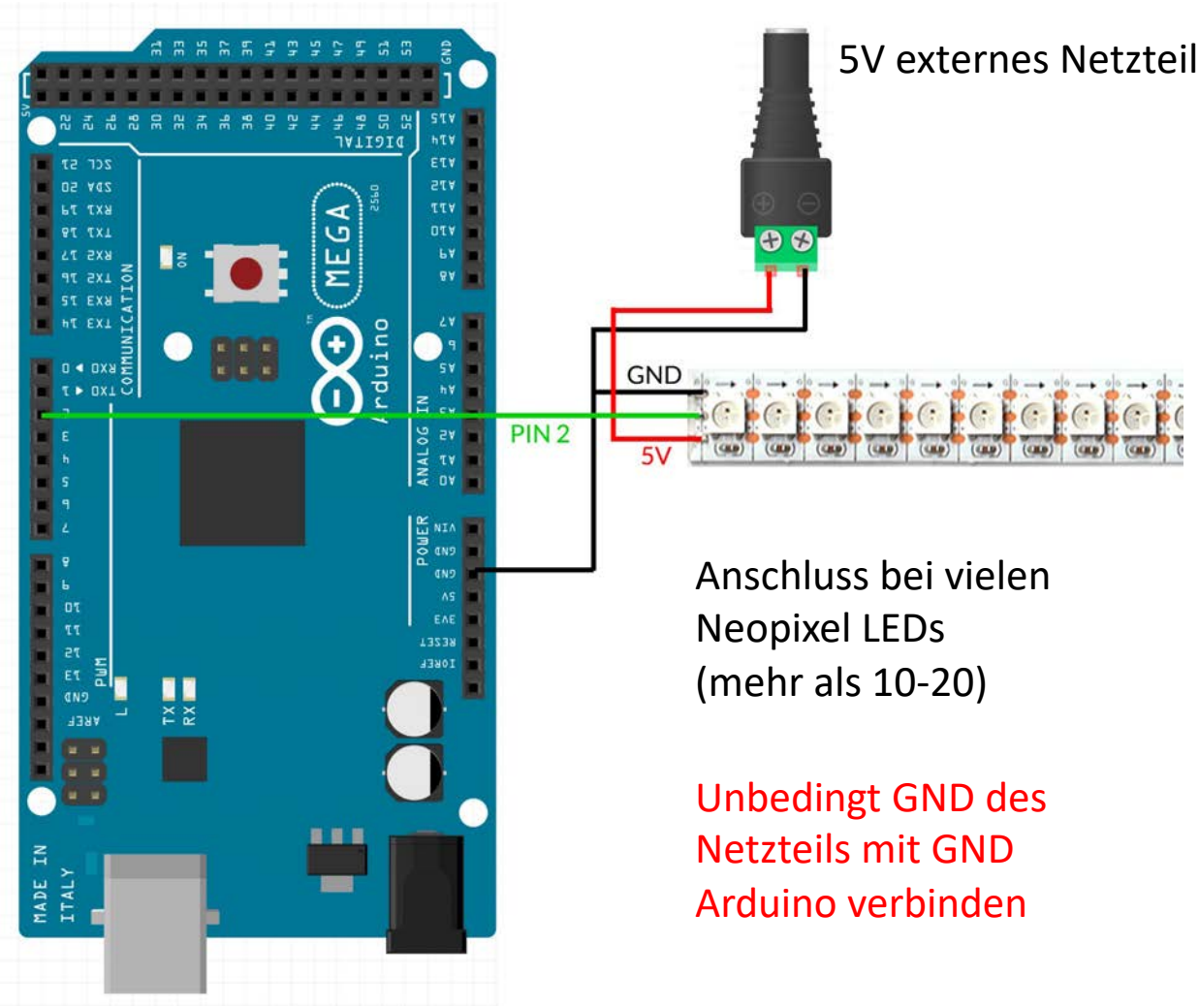
Stromverbrauch und Anschluss

WS2812 LEDs leuchten **sehr hell** und benötigen dann relativ viel Strom per LED (50-80 mA, also bei 10 LEDs 400-800mA)

Der Arduino kann **500mA** am 5V Ausgang liefern, für wenige LEDs (und nicht bei vollem Weiß) reicht das

Ansonsten muss ein **externes 5V-Netzteil** verwendet werden

Der **DIN** (data in) Anschluss der Neopixel wird mit einem beliebigen GPIO Pin verbunden



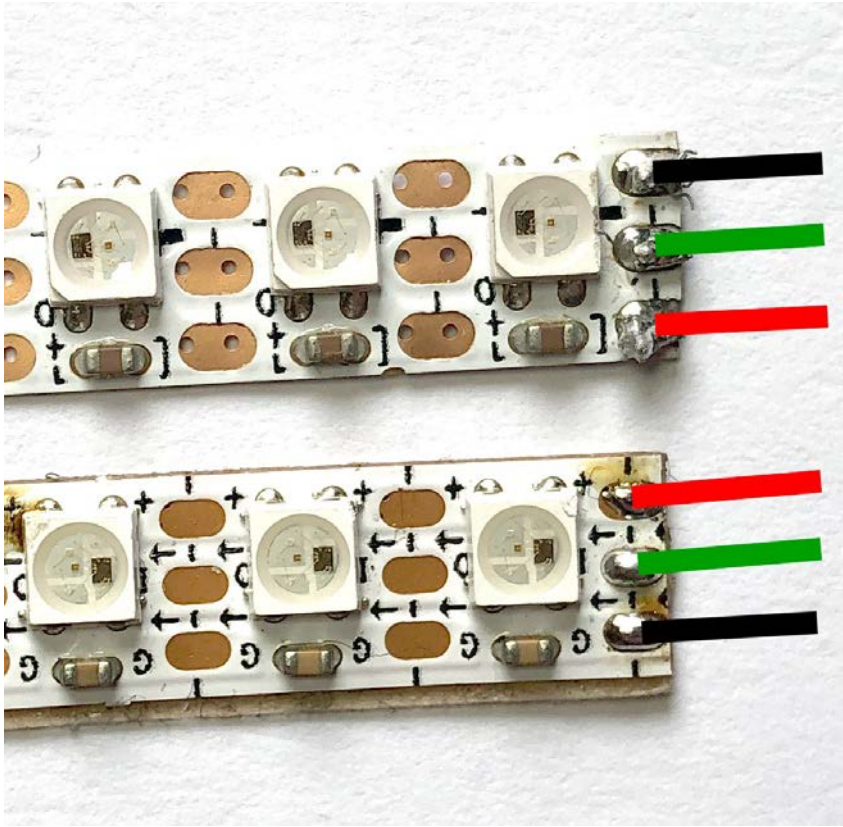
Anschluss bei vielen Neopixel LEDs (mehr als 10-20)

Unbedingt GND des Netzteils mit GND Arduino verbinden

Neopixel haben eine RICHTUNG, deshalb immer **DIN (data in)** der LEDs mit Arduino Pin verbinden!

Smart LED Besonderheiten

- Bei mehreren LEDs (Streifen, Ring, Matrix) sind immer alle LED in Reihe geschaltet
- Fällt eine LED in der Kette aus, bleiben alle nachfolgenden dunkel
- Erfahrungsgemäß ist dann die defekte LED **die letzte LED, die noch leuchtet** (nicht die erste dunkle), weil die helle LED die Daten nicht korrekt weiterleitet
- Die Pin-Belegung der Streifen ist nicht genormt, ist immer auf dem Trägerstreifen aufgedruckt
- **Fester Datentakt:** ca. 30 Updates pro Sekunde für 1000 angeschlossene LEDs (weniger LEDs -> schneller)
- **APA102** („Dotstar“) mit SPI und 2 Leitungen (Clock + Data) für noch schnellere Updates



Ansteuern per Arduino Code Bibliothek

Libraries

- **FastLED** (sehr mächtig und umfangreich, RGB)
  https://github.com/FastLED/FastLED

  RGBW-Hack für FastLED
  https://www.partsnotincluded.com/fastled-rgbw-neopixels-sk6812/

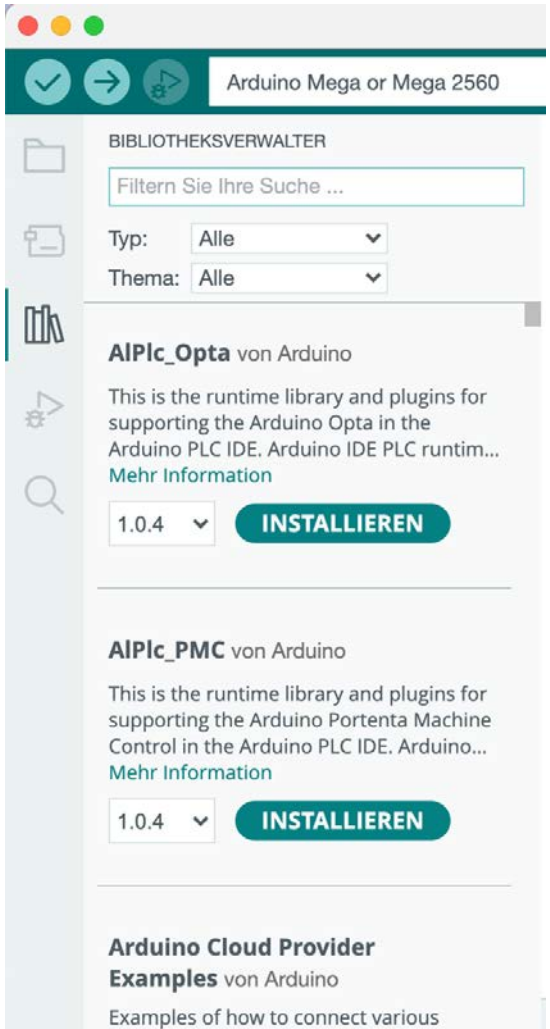
- **Neopixel** (von Adafruit, RGB und RGBW)
  https://github.com/adafruit/Adafruit_NeoPixel

App

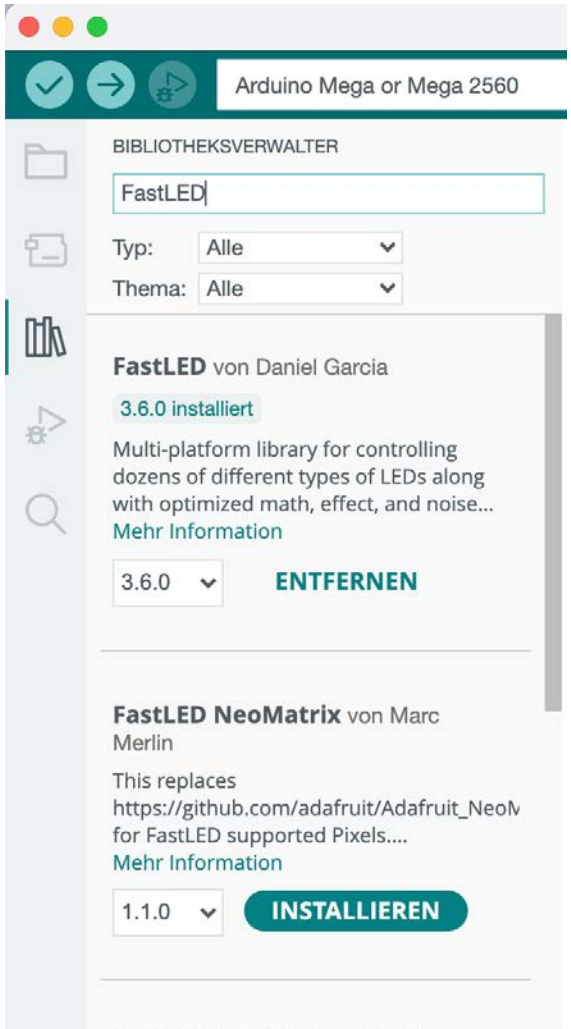
- **WLED** (für WLAN-Projekte, ESP32, mit mobile app etc)
  https://github.com/Aircoookie/WLED

Bibliothek installieren

In der Arduino IDE den „Bibliothek“-Button links anklicken (3. Icon von oben), Liste der verfügbaren Bibliotheken erscheint



Ins „Filtern Sie Ihre Suche…“-Feld „FastLED“ eintragen, **FastLED von Daniel Garcia** installieren (v 3.6.0 oder neuer)



Variablen-Arrays(Werte-Liste)

- Variablen, die nicht nur 1 Wert enthalten, sondern mehrere Werte desselben Typs, werden „Arrays“ genannt
- Um ein Array zu erstellen, wird dem Variablen**namen** ein eckiges Klammernpaar angehängt

```
int eineZahl = 1;
// eine einzelne Variable

int mehrereZahlen[] = { 1, 2, 3, 4 };
// ein Array mit 4 Zahlen

int weitereZahlen[10];
// ein Array mit 10 Zahlen, alle Zahlen sind zu Beginn 0
```

- Auf die einzelnen Array-Plätze wird über eckige Klammern zugegriffen
- Die Zählung der Elemente eines Arrays beginnt bei index 0!

```
int ersteZahl = mehrereZahlen[0];
// ersteZahl = 1

mehrereZahlen[2] = 1000; // Ein Wert innerhalb des Arrays wird (neu)
gesetzt, Array mehrereZahlen ist jetzt: { 1, 2, 1000, 4 }
```

Variablen vom Typ CRGB

```
CRGB meineFarbe = CRGB( 255, 0, 0 ); // ROT
```

- CRGB ist ein Variablen-Typ, der innerhalb der FastLED-Bibliothek (Datei libraries/FastLED/src/pixeltypes.h) definiert ist.
- Es ist keine „einfacher“ Typ wie int oder bool, sondern ein zusammengesetzter Typ, ein sogenanntes **„struct“**
- Ein struct kann eigene Variablen und eigene Funktionen enthalten, zB:

```
CRGB.r                  // liefert den Rot-Wert der aktuellen Farbe
CRGB.fadeLightBy(n);    // Funktion, um die im struct
                        // gespeicherte Farbe abzudunkeln
```

Vordefinierte Werte im CRGB struct werden mit zwei Doppelpunkten abgerufen:

```
meineFarbe = CRGB::Red;  // gibt vordefinierte Farbe "Red" (0xFF0000) zurück
```

Dokumentation: http://fastled.io/docs/struct_c_r_g_b.html

FastLED importieren und vorbereiten

- Um die Funktionen und Objekte der FastLED-Bibliothek im eigenen Code zu verwenden, muss die Bibliothek zu Anfang jedes Sketches importiert werden:

```
#include <FastLED.h>
```

- Anschließend geben wir die Anzahl der LEDs in unserem Aufbau und den Arduino Pin, an dem die Datenleitung angeschlossen ist, an:

```
#define NUM_LEDS 10 // wie const int NUM_LEDS = 10;
#define DATA_PIN 2  // wie const int DATA_PIN = 2;
```

- Schließlich erstellen wir noch ein Array vom Typ „CRGB“, in dem die Farbwerte jeder LED gespeichert sind.

```
CRGB led_array[ NUM_LEDS ];
```

FastLED inititalisieren und aktualisieren

- In der setup()-Funktion teilen wir jetzt dem FastLED-Objekt mit, welchen LED-Streifentyp wir verwenden wollen:

```
void setup() {
    FastLED.addLeds<NEOPIXEL, DATA_PIN>(led_array, NUM_LEDS);
}
```

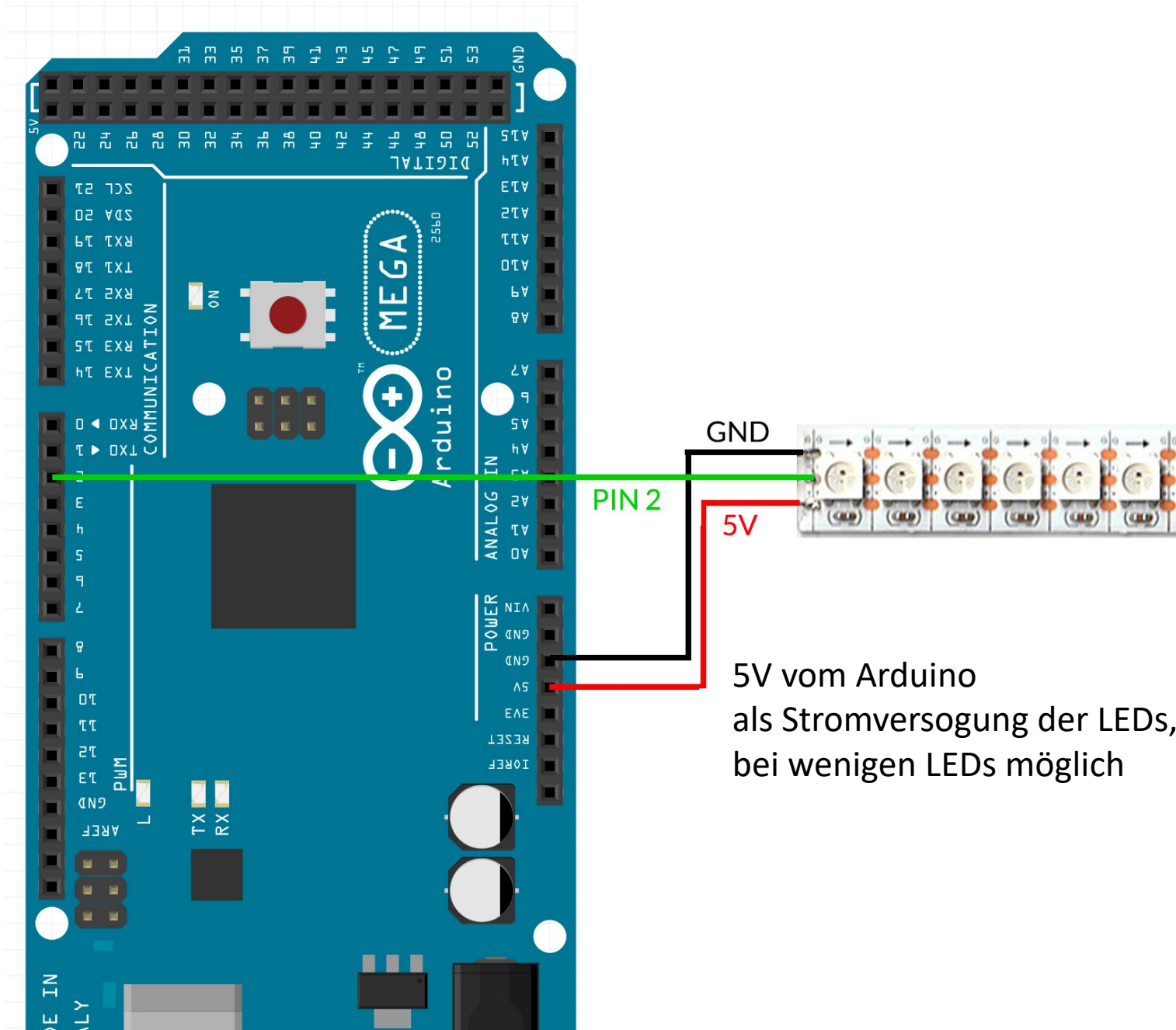
- „NEOPIXEL“ ist wiederum ein vordefinierter FastLED-Typ, ebenso funktioniert „SK6812“ stattdessen.

- In der loop()-Funktion rufen wir dann noch jedesmal `FastLED.show();` auf, wenn wir die LEDs aktualisieren wollen

```
void loop() {
    FastLED.show();
    delay(100);
}
```

- Zu Beginn werden alle CRGB-Objekte im `led_array` auf R=0, G=0, B=0 gesetzt, also sind alle LEDs erstmal aus!

LED Strip anschließen (Pin 2 = DATA IN)



Alle LEDs leuchten blau

Sketch: 20_FastLED_blau.ino

```
#include <FastLED.h>

#define NUM_LEDS 10
#define DATA_PIN 2

CRGB led_array[ NUM_LEDS ];

void setup() {
    FastLED.addLeds<NEOPIXEL, DATA_PIN>(led_array, NUM_LEDS);
    FastLED.setMaxPowerInVoltsAndMilliamps(5, 500);
}

void loop() {
    fill_solid( &led_array[0], NUM_LEDS, CRGB::Blue );
    FastLED.show();
    delay(100);
}
```

Alle/einzelne LEDs ansprechen

```
fill_solid( &led_array[…], NUM_LEDS, CRGB::Blue );
```

- **fill_solid()** ist eine FastLED-Funktion und füllt einen LED-Bereich mit einer Farbe. Dazu werden 3 Parameter übergeben:
  - **&led_array[…]** index der ersten zu füllenden LED
  - **NUM_LEDS** Anzahl der zu füllenden LEDs
  - **CRGB::Blue** vordefinierter RGB-Wert in FastLED, entspricht CRGB(0, 0, 255)

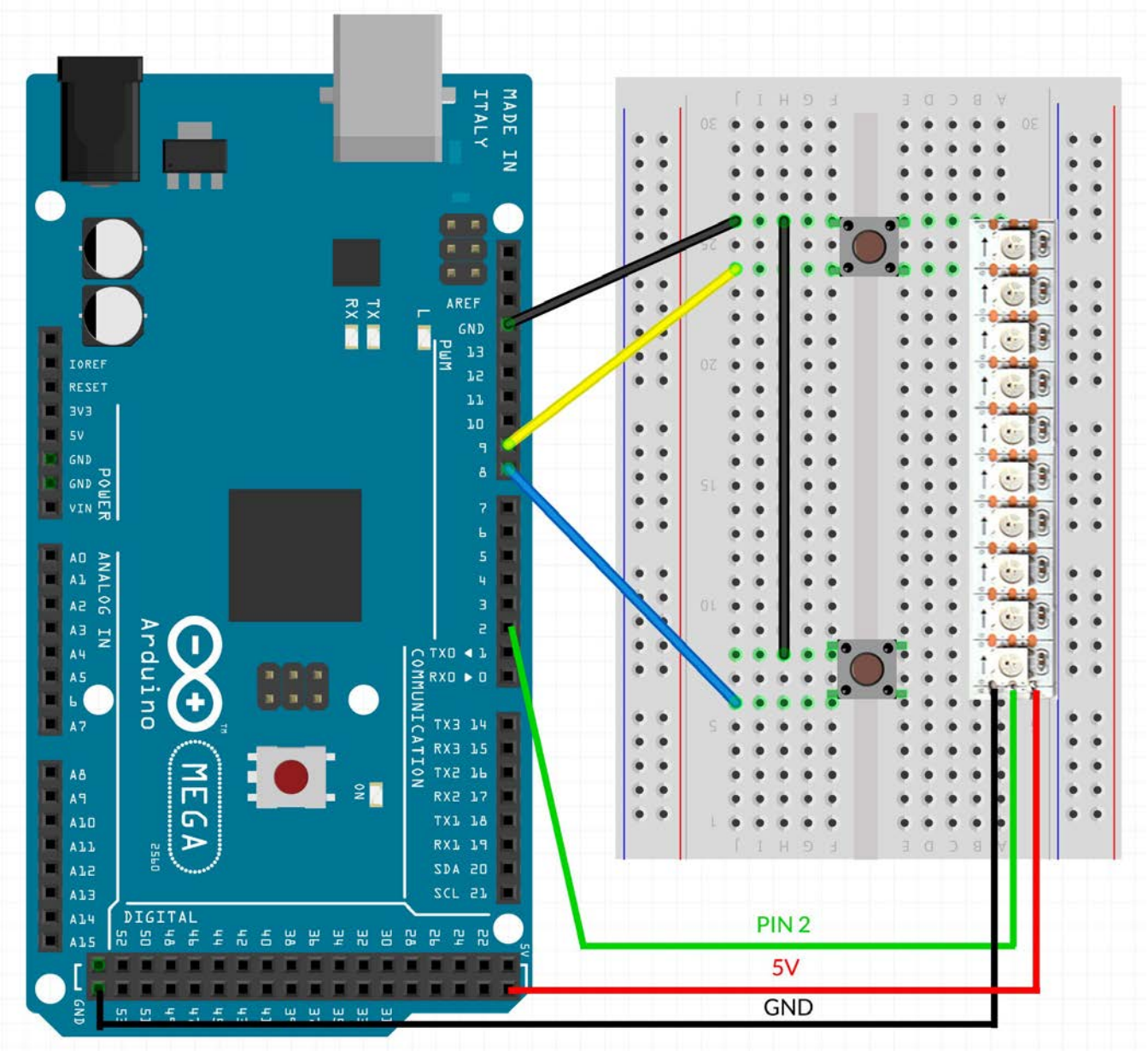
- Wenn nur die Farbe einer LED geändert werden soll, werden direkt die Farbwerte im `led_array` geändert:

```
led_array[ 0 ] = CRGB( 255, 0, 0); // LED Nummer 0 ist jetzt rot
```

- Mit einer “for“-Schleife kann zB ein Farbverlauf erzeugt werden:

```
for (int i=0; i<10; i++) {
    led_array[ i ] = CRGB( 0, 0, i*25 ); // Blauverlauf
}
```

Reaktionsspiel Aufbau



Taster Rechts:
Pin 9 + GND

Taster Links:
Pin 8 + GND

Neopixel-Streifen
DIN = Pin 2
GND
5V

Konzept für Reaktionsspiel

- Hardware: 2 Taster und LED-Streifen an Arduino
- Licht (einzelne blaue LED) läuft von der Mitte aus zufällig nach links oder rechts
- Wenn die letzte LED links oder rechts erreicht ist, muss der jeweilige Taster gedrückt sein
- Ist der Taster gedrückt (aber der andere nicht!), leuchtet der ganze Streifen kurz grün, ansonsten rot
- Geschwindigkeit der LED-Bewegung nimmt von Runde zu Runde zu

Vorüberlegungen

- Welche Variablen/Funktionen brauchen wir?
- Pseudo-Code (Umgangssprache) schreiben und Schritt für Schritt durch richtigen C-Code ersetzen

Variablen:

Licht-Animation
- an welcher **Position** ist das Licht aktuell
  (0 bis 5 LEDs von der Mitte)?
- Update-**Intervall** der Licht-Animation
  (millis zwischen updates)?
- in welche **Richtung** läuft das Licht
  (links/rechts)?

Spieler-Aktionen
- Zustand des **linken Tasters** (gedrückt?)
- Zustand des **rechten Tasters** (gedrückt)?

Spielablauf
- in welcher **Spielphase** sind wir
  (Start, Ablauf, Ende)
- in welcher **Spielrunde** sind wir

Funktionen:

Spiel-Start:
- Licht-Position auf Mitte setzen
- Richtung links/rechts zufällig auswählen
- gleich zu Spiel-Ablauf übergehen

Spiel-Ablauf:
- Licht-Position aktualisieren
- sind wir am Ende (links oder rechts)?
- wenn ja, zu Spiel-Ende übergehen

Spiel-Ende:
- ist (nur) der richtige Taster gedrückt?
- Gewonnen/Verloren darstellen
- Spielrunde erhöhen (= schnellere Geschwindigkeit)
- nach Wartezeit wieder zu Spiel-Start übergehen

```
// FastLED library importieren
// und LED Hardware Parameter (Anzahl, Data-PIN) festlegen

// Taster-Pins für linken und rechten Taster festlegen

// Spielablauf-Variablen

// Position der aktuell leuchtenden LED (= index im Streifen)
// Richtung der Bewegung

// Update-Zeit der LED-Bewegung (Pause in Millisekunden zwischen LED-Updates)
// Bewegung der LED alle 150 Millisekunden -> 750 Millisekunden für 5 LEDs gesamt
// Wert wird nach jeder Spielrunde runtergezählt, damit das Spiel schneller wird

void setup() {
     // LEDs initialisieren
     // Taster-Pins initialisieren
     // Spiel starten (Funktion spielStart() aufrufen)
}

void loop() {
     // Check ob nächster Spiel Schritt ausgeführt werden soll
     // wenn ja, Funktion spielAblauf() aufrufen
}

void spielStart() {
     // zuerst alle LEDs aus und 2 Sekunden warten
     // Richtung links oder rechts zufällig entscheiden
     // random(2) liefert Zufallszahlen zwischen 0 und 1
     // entsprechend der Richtung dann die Start-Position (index) der LED
     // festlegen: bei Richtung LINKS Start-Position 4 (Bewegung 4 > 3 > 2 > 1 > 0)
     // bei Richtung RECHTS Start-Position 5 (Bewegung 5 > 6 > 7 > 8 > 8)
}
```

```
void spielAblauf() {
     // zuerst alle LEDs aus und nur die aktuelle LED an in blau

     // Position der LED weiter bewegen nach links oder rechts für nächsten loop()
     // sind wir über dem Spielbereich von LED-Position = 0 bis 9?
     // dann Funktion spielEnde() aufrufen
}

void spielEnde () {

     // Prüfen, ob wir gewonnen haben
     // bei Richtung LINKS: Ist nur der linke Taster gedrückt (LOW), nicht der rechte?
     // bei Richtung RECHTS: Ist nur der rechte Taster gedrückt (LOW), nicht der linke?
     // Ergebnis optisch anzeigen für 1 Sekunde (gewonnen = alle LEDs grpn, verloren = rot)

     // Geschwindigkeit für die nächste Runde erhöhen und nächste Runde starten
     // durch Aufruf der Funktion spielStart()
}
```

21_FastLED_Reaktionsspiel.ino

```
#include <FastLED.h>

#define NUM_LEDS 10
#define DATA_PIN 2

CRGB led_array[ NUM_LEDS ];

const int TASTER_LINKS_PIN = 8;
const int TASTER_RECHTS_PIN = 9;

const int RICHTUNG_LINKS = -1;
const int RICHTUNG_RECHTS = 1;
int ledRichtung = RICHTUNG_LINKS
int ledPosition = 0;

unsigned long ledUpdateZeit;
int ledDelay = 150;

void setup() {
     FastLED.addLeds<NEOPIXEL, DATA_PIN>(led_array, NUM_LEDS);
     FastLED.setMaxPowerInVoltsAndMilliamps(5, 500);

     pinMode( TASTER_LINKS_PIN, INPUT_PULLUP );
     pinMode( TASTER_RECHTS_PIN, INPUT_PULLUP );

     spielStart();
}
```

21_FastLED_Reaktionsspiel.ino

```
void spielStart() {
     fill_solid( &led_array[0], NUM_LEDS, CRGB::Black );
     FastLED.show();
     delay(2000);

     if (random(2) == 0) {
          ledRichtung = RICHTUNG_LINKS;
          ledPosition = 4;
     } else {
          ledRichtung = RICHTUNG_RECHTS;
          ledPosition = 5;
     }
}

void loop() {
     if (millis() > ledUpdateZeit) {
          ledUpdateZeit = millis() + ledDelay;
          spielAblauf();
     }
}

void spielAblauf() {
     fill_solid( &led_array[0], NUM_LEDS, CRGB::Black );
     led_array[ledPosition] = CRGB::Blue;
     FastLED.show();

     ledPosition += ledRichtung;

     if (ledPosition < 0) spielEnde();
     if (ledPosition == NUM_LEDS) spielEnde();
}
```

```
void spielEnde () {
     bool gewonnen = false;

     if (ledPosition < 0) {
          if ((digitalRead( TASTER_LINKS_PIN )  == LOW) &&
              (digitalRead( TASTER_RECHTS_PIN ) == HIGH)) {
               gewonnen = true;
          }
     }

     if (ledPosition == NUM_LEDS) {
          if ((digitalRead( TASTER_LINKS_PIN )  == HIGH) &&
               (digitalRead( TASTER_RECHTS_PIN ) == LOW)) {
               gewonnen = true;
          }
     }
     if (gewonnen) {
          fill_solid( &led_array[0], NUM_LEDS, CRGB::Green );
     } else {
          fill_solid( &led_array[0], NUM_LEDS, CRGB::Red );
     }
     FastLED.show();
     delay(1000);

     if (ledDelay > 60) ledDelay -= 10;

     spielStart();
}
```

Farbexplosion!

- Testet auch gerne die **FastLED-Beispiele**:

  Arduino Menü -> Beispiele -> FastLED

Dazu müsst ihr ein Beispiel öffnen, es in eurem lokalen Arduino Project-Verzeichnis speichern und dann die **Werte für LED_PIN und NUM_LEDS anpassen**, bevor ihr es auf den Mega flashed…

- FastLED library Dokumentation (Modules, Classes, Examples):

http://fastled.io/docs/index.html

Sketch: 22_Fire2012.ino