Dokumentation

Projekt	Reaktometer
Semester	Wintersemester 2018/2019
Name	Heiko Jedamski
Datum	22.01.18

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Zusammenfassung

Im Rahmen der Veranstaltung „eingebettete Software“ soll ein Reaktionstester mittels Arduino Uno realisiert werden. Der entwickelte Tester verfügt über ein Menü, in dem der Benutzer entscheiden kann welches Spielniveau gespielt werden soll. Wenn der Taster nach der Einschaltanimation einmal gedrückt wird, wird der „Anfänger“-Test durchgeführt. Bei doppelten Tastendruck wählt der Benutzer den „fortgeschrittenen“ Test und bei dreifachen Tastendruck wird die Schwierigkeit „legendär“ aktiviert.

Über eine Zufallszahl wird eine Animation mittels LEDs gesteuert, die nach der generierten Zufallszeit erlischt. Nun beginnt der Reaktionstest. Umso schneller der Benutzer den Taster drückt, desto weniger LEDs werden auf HIGH gesetzt.

Das Programm ist iterativ, also kann der Benutzer nach jedem Test entscheiden welches Spielniveau gespielt werden soll.

Um die Ergebnisse zusammenzufassen wurde auf GitHub ein Projekt mit den jeweiligen Dateien erstellt.

2 Benutzerdokumentation

2.1 Benutzerhandbuch

1. Besorgen Sie sich die Hardware-Elemente und bauen Sie diese wie Abbildung 10: „Aufbau der Hardware“ zeigt auf.
2. Installieren Sie sich die Arduino Software für Ihren Desktop.
3. Laden oder kopieren Sie die im Repository hinterlegte Software „Reaktometer.zip“ auf ihren Arduino. (https://github.com/igit-igit/Reaktionstester)Nachdem Sie die Hardware aufgebaut haben, kann der Tester durch das Anschließen an einer Stromquelle in Betrieb genommen.

   Zum Beispiel können Sie das USB-C Kabel, welches für den Arduino benötigt wird, mit Ihrem PC verbinden.
4. Haben Sie die Punkte 1-3 befolgt können Sie nun Ihre Reaktion überprüfen.
5. Wenn ihnen die Voreinstellungen zu schwer beziehungsweise zu leicht sind, gehen Sie auf das File „Reaktometer.cpp“ und ändern Sie die Wertebereiche der Funktionen „*test_loop1*“ , „*test_loop2*“ und „*test_loop3*“.
6. Den Reaktionstaster können Sie außer Betrieb setzen, in dem Sie Stromversorgung kappen.
7. Die Software bleibt solange erhalten bis Sie diese durch Änderungen überschreiben.

Sobald der Arduino mit Strom versorgt wird läuft das Programm wie folgt ab:

1. Es wird eine Einschaltanimation gezeigt in dem sogenannten Setup.
2. Ist die Animation erloschen befindet sich der Tester im Hauptprogramm.
3. Hier kann der Benutzer drei Spielniveaus wählen

   Der Modus „*Anfänger*“ wird mit einem **einfachen** Tastendruck gewählt.

   Der Modus „*Fortgeschritten*“ mit **doppelten** Tastendruck.

   Der Modus „*Weltklasse*“ mit **dreifachen** Tastendruck.

   Achtung: Bevor Sie einen Spielmodus wählen, warten Sie bis die ersten beiden grünen LEDs kurz aufleuchten.
4. Nachdem gewählt wurde, erscheint die sogenannte „Testanimation“. Diese wird über eine zufällige Zeit angezeigt und erlischt bei Überschreitung dieser Zufallszeit.
5. Nun sind Sie gefragt.Sie müssen so schnell wie möglich den Taster drücken, damit möglichst wenig LEDs angezeigt werden.
6. Das Programm ist iterativ. Der Benutzer wird nach jedem Test gefragt, welches Niveau gewählt werden soll.

3 Entwicklerdokumentation

3.1 Wartung

Das Projekt kann in dem .CPP-File geringfügig verändert werden, durch (wie in 2.1 beschrieben) beispielsweise das Anpassen der Wertebereiche, um eine neue Spieldynamik zu bekommen. Des Weiteren können die Animationen verändert werden.

Je nachdem wie oft der Tester benutzt wird kann die Lebensdauer der LEDs verringert werden. Diese können bei Ausfall jedoch sehr einfach ersetzt werden. Zudem ist es möglich, dass Kabelspitzen abbrechen, die ebenfalls einfach ersetzt werden können. Die Software selbst hat jedoch keine Alterungsprozesse. Eventuell muss die Arduino.h Bibliothek sowie Bibliotheken, die hinzugefügt werden aktualisiert werden.

3.2 Ressourcen

Um das Projekt weiterzuentwickeln, kann die auf dem GitHub-Repository befindliche Software verwendet werden. Zudem werden dort weitere Elemente festgehalten:

- Aufbau des Steckbretts mittels Fritzing-Software „Schematik_Steckplatine.png“
- Verwendete Hardware (Dokumentation)
- Struktogramme des Endproduktes
- erste Code-Struktur „Reaktionstester (mit Hardware)“
- Zwischenversionen „Reaktionstest.zip“, bei der die Reaktionstester-Logik von der Hardware getrennt ist.
- Endprodukt „Reaktometer.zip“

Link zu meinem Repository: https://github.com/igit-igit/Reaktionstester

Während des Projektes wurde folgende Software verwendet:

- Fritzing → Schaltplan illustrieren
- Arduino Web Editor → Entwicklungsumgebung für Code
- Git Hub → Repository
- Structorize → Struktogramme erstellen
- Arduino Software → kompilieren/ hochladen des Codes
- Gyazo → verwendet um Codestrang mit Hilfe eines Bildschirmfotos in die PowerPoint Präsentation einzufügen

3.3 Funktionsweise des Projektes

```cpp
// Konstruktor
RT::RT() {
  //Pins definieren
  led1 = 3;
  led2 = 4;
  led3 = 5;
  led4 = 6;
  led5 = 7;
  led6 = 8;
  led7 = 9;
  buttonPin = 2;
  //LEDs und Button initialisieren
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(led4, OUTPUT);
  pinMode(led5, OUTPUT);
  pinMode(led6, OUTPUT);
  pinMode(led7, OUTPUT);
  pinMode(buttonPin, INPUT);
  //pressed und pressed_menu Initialisieren bevor die Haupt-Loop startet
  pressed = 0;
  pressed_menu = 0;
}
```

Abbildung 1: Konstruktor Teilausschnitt des CPP-Files

Der Konstruktor in dem CPP-File wird genutzt, um ein Objekt in der ino-Datei zu erzeugen. Des Weiteren werden dort die Pins und der Button initialisiert und definiert. Die unten dargestellte Header-Datei dient als Schnittstelle zwischen Ino- und-CPP-Datei und macht die in der Klasse RT befindlichen public-Funktionen in der ino.Datei zugreifbar. In der CPP-Datei werden alle Funktionen mit den dort gültigen Variablen festgehalten. Dort sind beispielsweise die Abläufe für Animation sowie die Zustände für die Switch-Anweisungen definiert.

```cpp
#ifndef Reaktometer_h
#define Reaktometer_h
#include "Arduino.h"
class RT {
  public:
    RT();
    void animation();
    void menuauswahl();
  private:
    void testAnimation();
    void wait_until_release();
    void test_loop1();
    void test_loop2();
    void test_loop3();
    void reset();
    void oneLed();
    void twoLed();
    void threeLed();
    void fourLed();
    void fiveLed();
    void sixLed();
    void sevenLed();
    int led1;
    int led2;
    int led3;
    int led4;
    int led5;
    int led6;
    int led7;
    int buttonPin;
    long start_time;
};
#endif
```

Abbildung 2: Header Datei

```cpp
#include "Reaktometer.h"

// Objekt erzeugen
RT reaktometer;

void setup() {
  reaktometer.animation();
}

void loop() {
  reaktometer.menuauswahl();
}
```

Abbildung 3: ino-Datei

Reaktometer
Eine grobe Übersicht

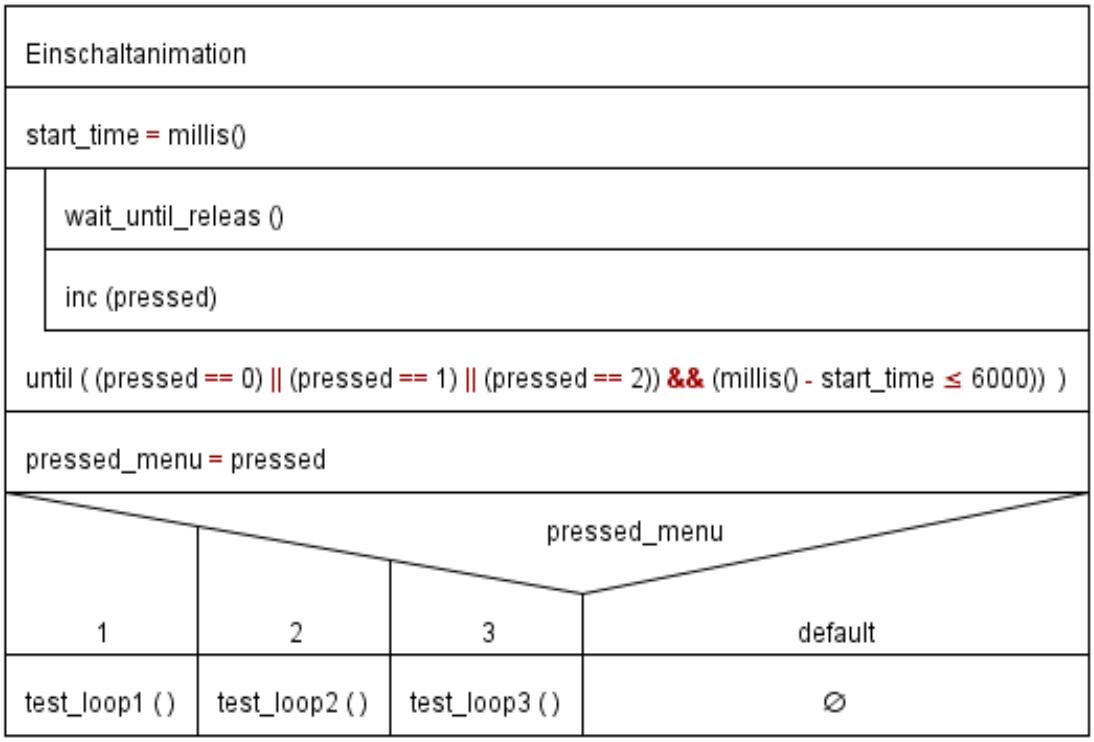

Abbildung 4: Programmcode Aufbau

Die oben angeführte Grafik zeigt die fundamentale Struktur des Programms mit Hilfe eines Struktogramms. Nachdem das Setup und somit auch die Aktion Einschaltanimation durchlaufen wurde, geht das Programm in die Loop (Hauptprogramm). Hier wird einer Zeitvariable (start_time) der Funktionswert von millis() zugewiesen. Die Variable start_time wird benötigt, um einen Zeitraum für die Wahl eines Spielniveaus zu geben. Das Programm springt dann in die „do-While-Schleife“ (Iteration mit Abbruchbedingung) solange bis die Abbruchbedingung zutrifft. Das heißt, wenn zum Beispiel pressed = 0 ist, und die Zeitdifferenz kleiner gleich 6 Sekunden ist, würde das Programm aus der Schleife raus gehen, jedoch wäre der übergebene Wert an pressed_menu ebenfalls 0 und keine Zeitmessung würde starten. Wenn der Benutzer jedoch einmal drückt und einen Moment wartet bis die Zeit eintritt, die dazu nötig ist um aus der do-While-Schleife zu gehen, wird pressed_menu der Wert 1 übergeben, wodurch das Programm den Reaktionstest für test_loop1() ausführt. Das gleiche gilt für den Fall, dass der Benutzer zwei mal den Taster drückt und test_loop2() ausgeführt wird. Geht man in dieser Schleife davon aus, dass der Taster bereits zwei Mal gedrückt wurde, und noch ein weiteres Mal gedrückt wird, geht das Programm schon vor Ablauf des Auswahlzeitraumes aus der Schleife, da pressed = 3 nicht in der Abbruchbedingung steht und False ergibt, woraus folgender Ausdruck resultiert : (False && True) → False

Nun wird der dritte Fall beziehungsweise test_loop3 ausgeführt.

Nachdem einer der test_loops durchlaufen wurde, wird die Funktion reset()

aufgerufen. Reset () dient zum Zurücksetzen der LEDs sowie dem iterativen Initialisieren der Variablen pressed und pressed_menu.

Die Funktion wait_until_release () im Schleifenrumpf der „do-While-Schleife“ sorgt dafür, dass der Taster länger gedrückt werden kann. Solange der Taster gesetzt ist soll das Programm in der Schleife verweilen. Mit Hilfe von Serial.println() kann der Zustand des Tasters über den seriellen Monitor ausgegeben werden.

```
// Funktion, die langes halten ermöglicht
void RT::wait_until_release() {
  while (digitalRead(buttonPin) != LOW) {
    // Serial.println(digitalRead(buttonPin));
    }
  }
```

Abbildung 5: Codestrang der Funktion wait_until_release

Ein weiteres erwähnenswertes Programmkonstrukt ist das der Funktionen test_loop1, test_loop2 und test_loop3, die alle dieselbe Semantik besitzen, jedoch unterschiedliche Wertebereiche in den Case- Anweisungen aufweisen. Diese unterschiedlichen Wertebereiche werden genutzt um die 3 vordefinierten Spielniveaus zu realisieren. Je nach Wahl des Spielniveaus werden diese Funktionen in den jeweiligen Case-Fällen ausgeführt.

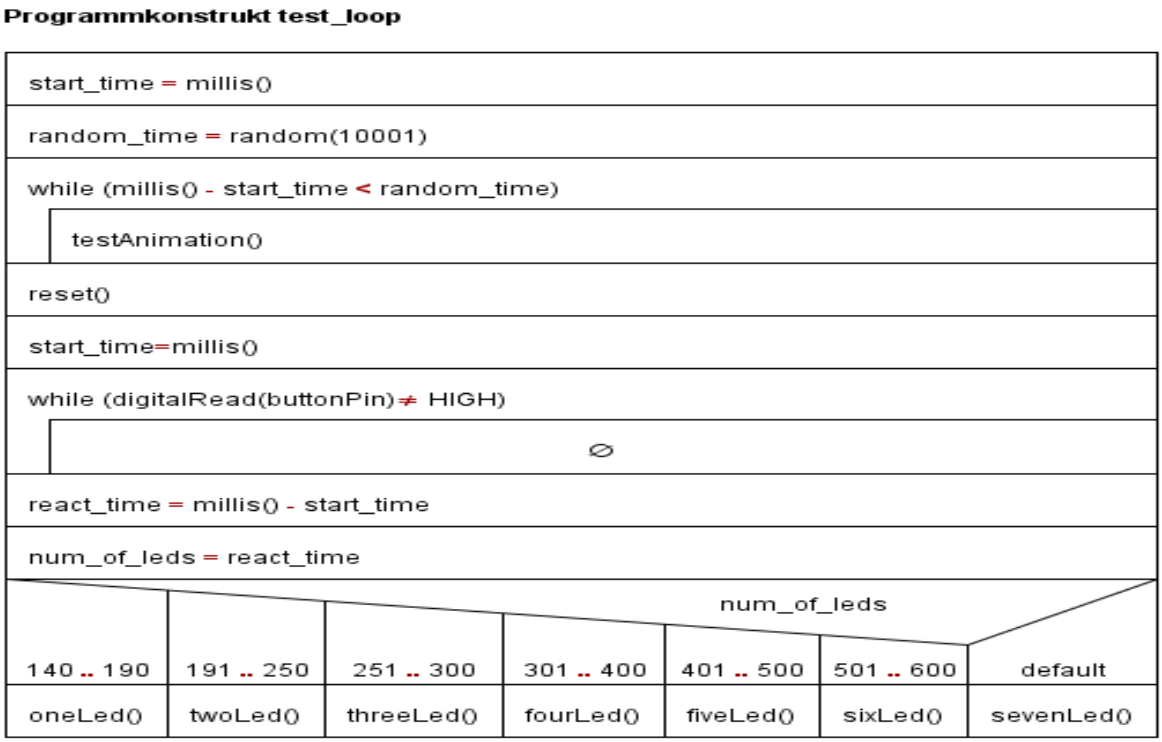

Abbildung 6: Aufbau der Funktion test_loop()

Dieses Struktogramm dient als Übersicht der Funktionen test_loop1 (), test_loop2 () und test_loop3 ().

Zu Beginn der jeweiligen Funktion wird einer Zeit-Variable der Funktionswert von millis() zugewiesen. Diese Zeit-Variable wird genutzt um den aktuellen Zeitwert zu bestimmen.

In der nächsten Aktion wird in einem Zeitraum von bis zu 10 Sekunden (10001 Millisekunden) eine Zufallszahl erzeugt. Die Zufallszeit wird in der Funktion menusauswahl () mit Hilfe der Funktion randomSeed (analogRead(0)) initialisiert. Zuvor hat der Zufallsgenerator immer die gleichen Zahlwert-Sequenzen generiert dieses Problem konnte mit Hilfe von Serial.println(random_time) auf dem seriellen Monitor festgestellt werden.

```
//Generieren der Zufallszeit
random_time = random(10001);
//Ausgabe der Zufallszeit
Serial.println(random_time);
```

Abbildung 7: Generieren der Zufallszahl und Ausgabe im seriellen Monitor

```
while ((millis() - start_time < random_time)) {
  testAnimation();
}
reset();
// Zeitmessung nach dem alle LEDs ausgschaltet sind
start_time = millis();
// Solange nicht gedrückt wurde bleibt er in while, wenn HIGH geht
// errechnet die Reaktionszeit
while (digitalRead(buttonPin) != HIGH) {
}
react_time = millis() - start_time;
Serial.println(react_time);
num_of_leds = react_time;
```

Abbildung 8: Animation anzeigen und Reaktionsmessung durchführen

Die Testanimation wird solange ausgeführt bis der aktuelle Zeitwert größer ist als die generierte Zufallszeit. Im Anschluss wird die Funktion reset () aufgerufen und alle LEDs gehen aus. Nun beginnt die Reaktionsmessung. Der Zeitpunkt an dem reset () aufgerufen wird, wird start_time zugewiesen und das Programm bleibt solange in der While-Schleife bis der Benutzer den Taster drückt. Die Zeitdifferenz zwischen dem aktuellen Zeitwertes millis() und dem der Variable start_time wird an react_time übergeben. Anschließend wird dieser Wert num_of_leds zugewiesen.

```
switch (num_of_leds) {
  // Ab 140 beginnen damit
  case 140 ... 190:
    oneLed();
    break;

  case 191 ... 250:
    twoLed();
    break;

  case 251 ... 300:
    threeLed();
    break;

  case 301 ... 400:
    fourLed();
    break;

  case 401 ... 500:
    fiveLed();
    break;

  case 501 ... 600:
    sixLed();
    break;

  // Auf 7 LEDs begrenzt -
  default:
    sevenLed();
    break;
}
```

Abbildung 9: Switch-Anweisung

Jetzt kann die Reaktionszeit mittels LEDs angezeigt werden. Der Wertebereich für den ersten Fall ist so definiert, dass ein permanentes Drücken des Tasters während der Testanimation zu sieben angezeigten LEDs führt. Wenn der Benutzer nach dem erlischen der Testanimation länger benötigt als der Wertebereich für Case 501 .. 600 es zulässt, werden sieben LEDs angezeigt.

4 Bestehende Probleme

Der Zeitabstand des Aufleuchtens der LEDs nach dem der Benutzer den Spielmodus gewählt hat nicht immer gleich. Die do-While-Schleife, die für dieses Ergebnis verantwortlich ist, wurde aber gewollt so verwendet, sodass es einen Zeitraum für die Auswahl eines Spielniveaus gibt.

Manchmal stimmt die Anzahl der Tastendrücke bei der Menüauswahl nicht mit der des ausgewählten Spielniveaus überein. Dies hängt wahrscheinlich mit dem Timer in der Abbruchbedingung der do-While-Schleife zusammen.

Des Weiteren ist die Reaktion von Mensch zu Mensch unterschiedlich. Deshalb muss man die Wertebereiche mit Hilfe vieler Selbstversuche (auch anderer Personen) kalibrieren.

5 Weiterentwicklung

Das Programm könnte beispielsweise durch eine Audioausgabe weiter ausgebaut werden. Hier wäre es möglich je nach Reaktionszeit eine Audioausgabe oder eine Melodie gespielt werden. Des Weiteren könnte man eine 7 Segment-Anzeige einbauen, um die Reaktionszeit, die bisher lediglich im seriellen Monitor und mit LEDs ausgegeben wird, anzuzeigen. Eventuell könnte auf Basis vieler Versuche von älteren Menschen ein Reaktionstester für diese Personengruppe kalibriert werden.

6 Hardware

Bauteil:	**Menge:**
230 Ohm Widerstand	7
10kOhm Widerstand (Tasterwiderstand)	1
Arduino Uno	1
LED	7
Taster	1
USB-C Kabel	1
Kabel	10

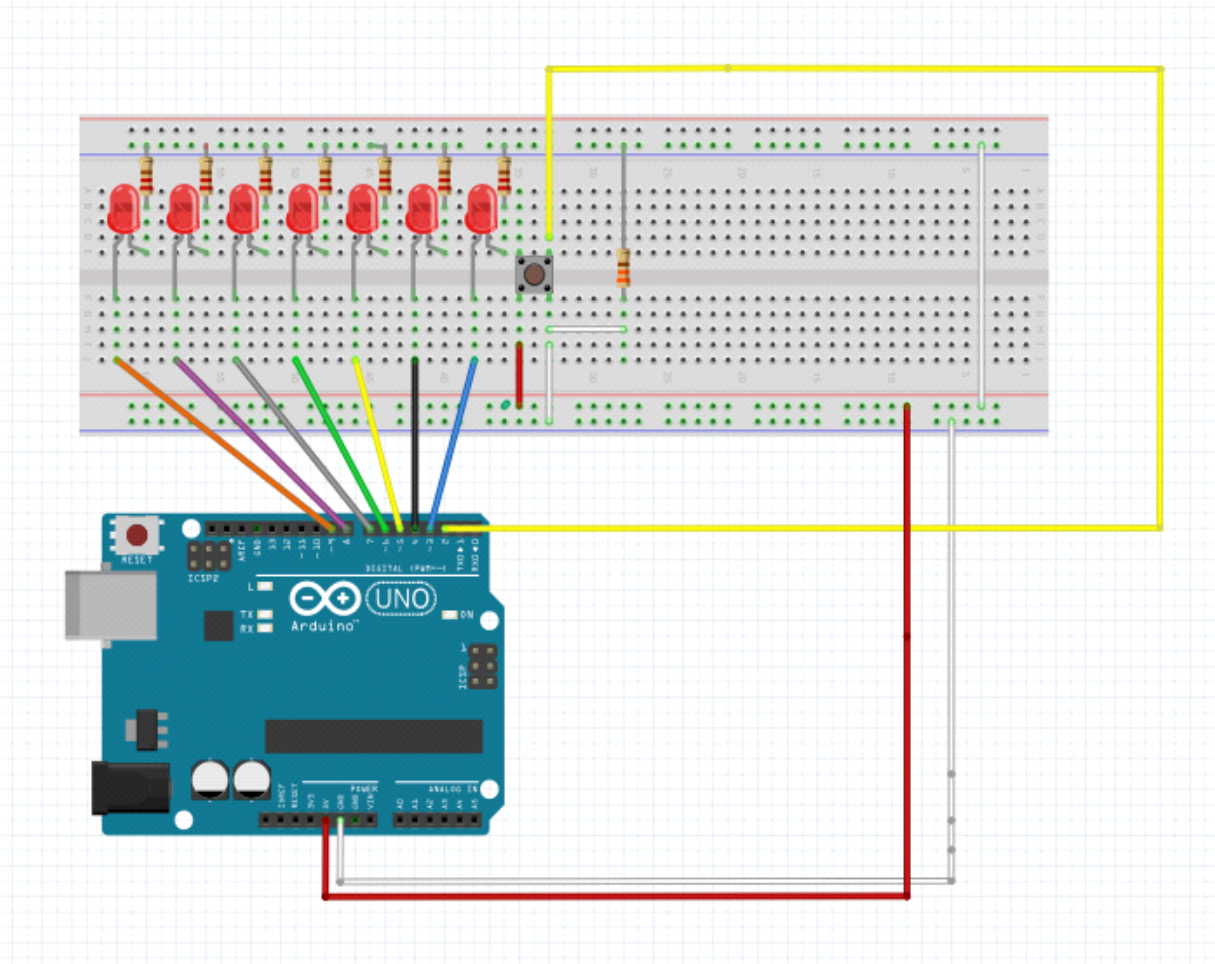

Abbildung 10: „Aufbau der Hardware“

Der Aufbau sieht folgendermaßen aus:

Der Arduino und das Steckbrett müssen mit dem 5V und dem GND Pin verbunden werden damit der Arduino die LEDs, welche durch die Pins 3-9 angesteuert werden, mit Strom versorgt. Der Pin 2 wird mit dem Taster verbunden. Der Taster dient als INPUT und wird über einen hohen Widerstand auf Masse gezogen. Die LEDs hingegen sind als OUTPUT konfiguriert. Sie werden mit dem „kurzen Bein“ über einen Widerstand auf Masse gezogen. Das „längere Bein“ wird mit 5V versorgt.