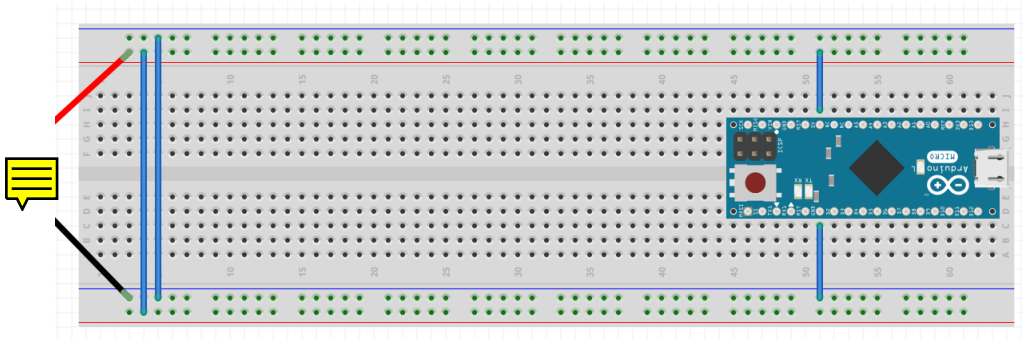
Übungsblatt 4 – Arduino

Aufgabe 1

Für den Code und Bilder/ Videos der Resultate siehe Ordner „Code“ und „Anhang“

1.1

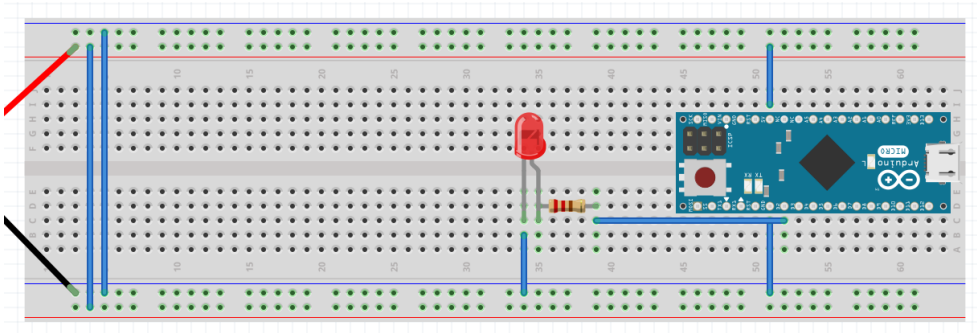
Ich habe zudem auch gleich noch die Stromversorgung des Arduino Micros über das Steckbrett getestet. Dazu habe ich einfach den Pluspol mit Pin „5V“ des Arduinos und den Minuspol mit dem „Ground“ Pin des Arduinos verbunden. Hat alles reibungslos funktioniert, und auch die LED blinkt (nach kurzer Verzögerung) entsprechend dem Programm, das auf den Arduino gespielt wurde.



1.2

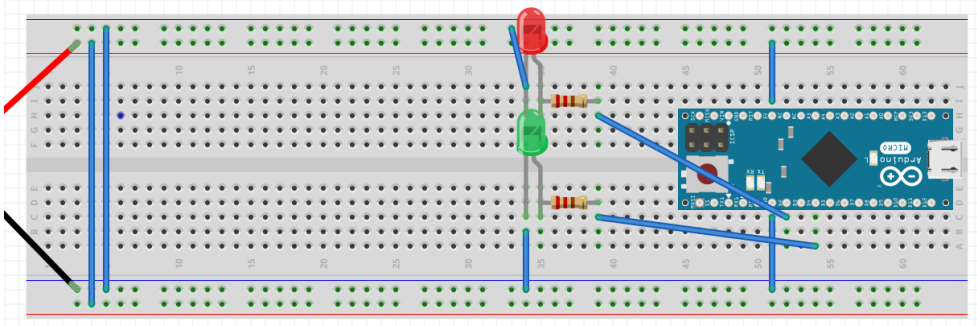
LED wurde an einen DigitalPin (nur 0V oder 5V) angeschlossen (Pin 2).

➔ Analoge Pins haben „Wellenlinien“ als Zeichen, digitale eine gerade Linie



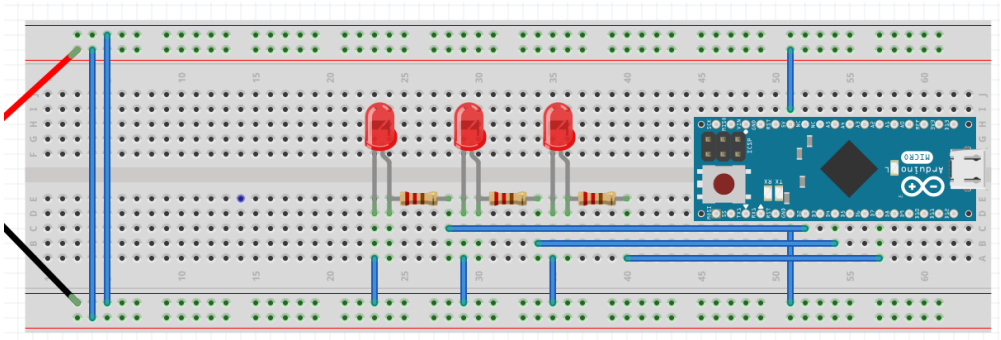
1.3

Eine grüne LED wurde an Pin 4 gelegt.



1.4

Drei LEDs wurden an die Pins 2, 4 und 7 gelegt. Die linke LED (Pin 2) repräsentiert den Wert 2^2, die mittlere (Pin 4) den Wert 2^1 und die rechte (Pin 7) den Wert 2^0.



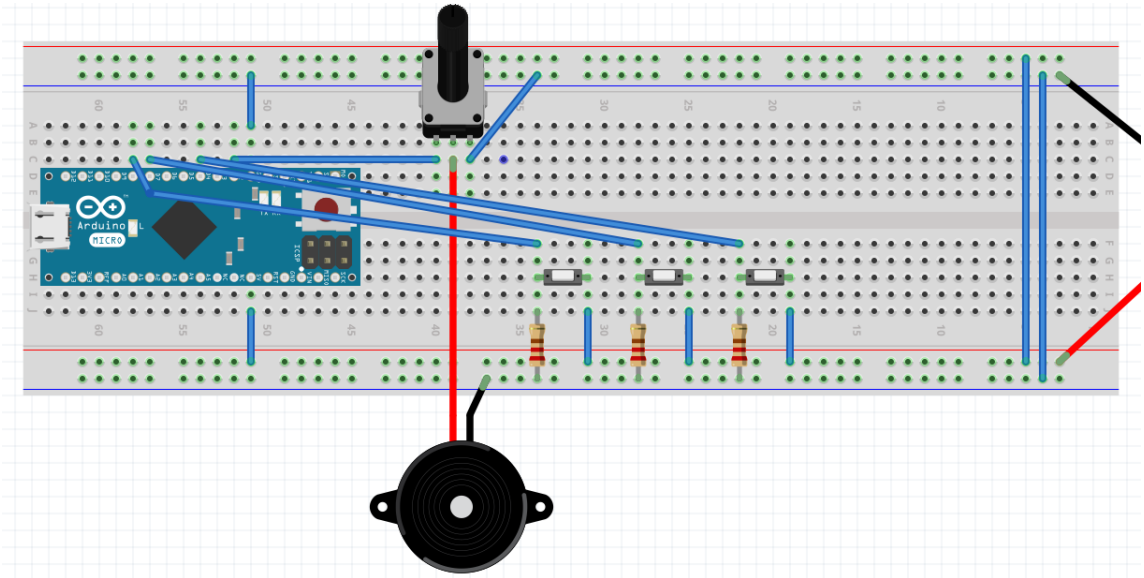
Übungsblatt 4 – Arduino

Aufgabe 2

Für den Code und Bilder/ Videos des Resultats, siehe Ordner „Code“ und „Anhang“

Planung und Umsetzung:

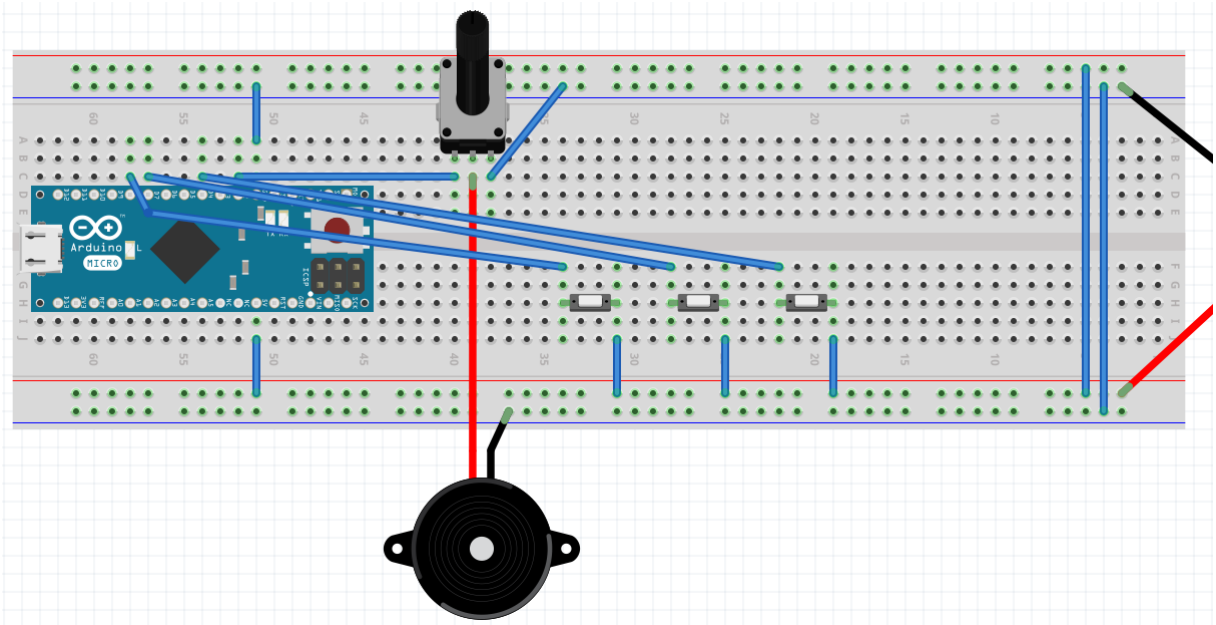
- Als Ausgabepin wähle ich einen digitalPin, da ja nur 2 Werte benötigt werden (HIGH oder LOW) → Pin 2 für Ausgabe/ Anschluss für Lausprecher
- Um in einer bestimmten Frequenz Spannung an einem Pin zu erzeugen, benötigt man die Methode: **tone(pin, frequency, duration)** bzw. **tone(pin, frequency)**
  - ➔ https://www.arduino.cc/reference/en/language/functions/advanced-io/tone/
  - ➔ Funktion **noTone()**, um zu stoppen, bzw. wird tone überschrieben, wenn man es erneut aufruft, also es können nicht zwei Töne/ Frequenzen gleichzeitig ausgegeben werden
- Für meine drei Töne wähle ich folgende Frequenzen:
  - ➔ Kammerton a: 440 Hz
  - ➔ Schloss-c: 260 Hz
  - ➔ h‘‘: 987 Hz
  - ➔ Liste mit Tönen und zugehörigen Frequenzen: http://www.sengpielaudio.com/Rechner-notennamen.htm
- Um unterschiedliche Töne vom Lautsprecher abspielen zu lassen wähle ich drei einfache Buttons. Wenn die Buttons gedrückt werden, soll eine Spannung am jeweilig verbundenen InputPin anliegen, sonst nicht. Im Code wird dann abgefragt, an welchem Pin gerade eine Spannung anliegt, und dann wird dementsprechend der dazugehörige Frequenzton ausgegeben bzw. eine Spannung in der bestimmten Frequenz am Output Pin erzeugt.
  - ➔ Als **InputPins** wurden die drei digitalPins **4, 7 und 8** verwendet.
  - ➔ Im Code müssen die Pins jeweils mit **pinMode()** richtig als **INPUT** oder **OUTPUT** definiert werden
- Ich will zudem die Lautstärke der Töne regulieren können. Dies kann man mit unterschiedlichen Vorwiderständen vor dem Lautsprecher erreichen (da sich die Spannung ja laut Spannungsteiler-Prinzip aufteilt). Ich wähle ein Potentiometer, da ich hier den Widerstand variabel ändern kann.



Notizen/ Probleme:

Fehlender Pull-Down Widerstand an den Buttons

Mein Breadboard sah zuerst folgendermaßen aus:



➔ **Pull-down Widerstände fehlen an den Buttons!**

Problemerläuterung:

Ich hatte also zuerst bei den einzelnen Pins immer 0-en und 1-en empfangen, auch wenn ich die Buttons dazu gar nicht gedrückt hatte und somit gar kein Strom hätte fließen dürfen (Debugging mit dem Serial Monitor). Trotzdem wurde anscheinend immer „abwechselnd“ Spannung bzw. keine Spannung an den Input Pins empfangen. Beim Anschließen des Lautsprechers hat sich das immer als unangenehmes Rauschen bemerkbar gemacht.

Erklärung für das Verhalten:

Ein Pullup- oder Pulldown-Widerstand wird dazu verwendet, einen Eingang auf einen definierten Wert zu "ziehen". Normalerweise befindet sich der Eingang im Zustand "schwebend/hochohmig", welcher sich irgendwo zwischen High und Low befindet. Nun sind Schaltungen leider nicht komplett ohne Störsignale, und durch Einstrahlungen von Signalen kann es nun passieren, dass kurzzeitig mal ein Wert über- oder unterschritten wird und der Eingang plötzlich ein High- oder Lowsignal bekommt.

(Quelle: https://rn-wissen.de/wiki/index.php/Pullup_Pulldown_Widerstand#:~:text=Ein%20Pullup%2D%20oder%20Pulldown%2DWiderstand,zwischen%20High%20und%20Low%20befindet)

Lösung: Pull-up- oder Pull-Down Widerstand

Um das oben genannte Problem zu vermeiden, wenn am Eingang (in unserem Fall InputPin) nichts passiert, wird die Spannung entweder auf HIGH (Pull-up) oder auf LOW (Pull-down) „gezogen“.

➔ Pull-up: zieht Spannung auf HIGH, wenn am Eingang nichts passiert
➔ Pull-down: zieht Spannung auf LOW, wenn am Eingang nichts passiert

➔ Um Pull-up Widerstand umzusetzen, muss man den Eingang über einen Widerstand mit Vcc verbinden

➔ Um Pull-down Widerstand umzusetzen, muss man den Eingang über einen Widerstand mit GROUND verbinden.

Sonstige Quellen:

- Video, das das Empfangen eines Imput-Signals an einem Pin und mit einem Button einfach erklärt: https://www.youtube.com/watch?v=eaFvQG8wrGw&t=113s

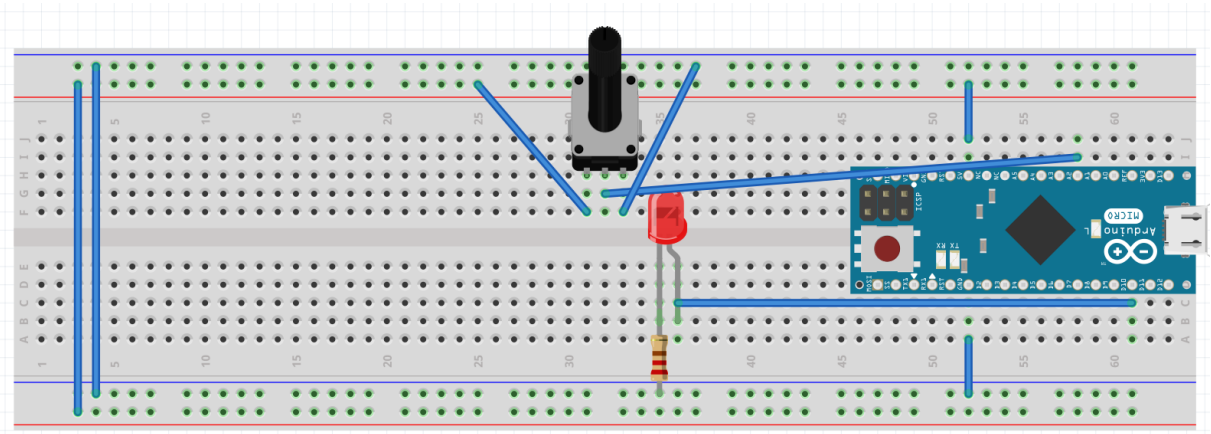
Übungsblatt 4 – Arduino

3.1

Für Bilder/ Videos/ Code der Resultate/ Probleme siehe Ordner „Anhang“ und „Code“

Lösung/ Umsetzung

- Potentiometer Pin 1 an Vcc, Pin 2 an Pin A2 des Arduinos, Pin 3 an Ground
- Im Code musste noch der eingelesene Wert von analogRead() angepasst werden
  ➔ Siehe Problem und Lösungsbeschreibung bei den Notizen unten drunter!

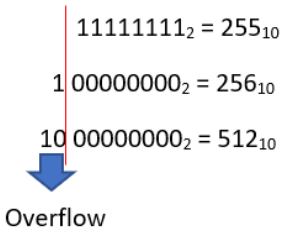


Notizen/ Probleme

LED hat mehrere Helligkeitszyklen

Problem: Anfangs hatte die LED mehrere Zyklen, d.h. wenn man das Potentiometer einmal von links nach rechts aufgedreht hat, wurde die LED immer heller bis zu einem Maximum. Dann ging sie plötzlich aus und wurde von da an wieder heller. Und dieses Prozedere ist beim einmaligen Aufdrehen des Potentiometers insgesamt 4-mal zu beobachten (siehe Video im Anhang). Beim Serial Monitor schienen aber die Werte des Potentiometers, da diese konstant steigen, wenn man es aufdreht (Werte steigen von 0 - 1023).

Erklärung: Das Problem liegt im Code. Die Methode **analogRead**() liest Werte von 0-1023 ein. Der **analogWrite**() Methode kann man allerdings nur int-Werte im Bereich 0-255 übergeben. Sobald also der Wert 256 übergeben wird, kommt es zu einem Overflow und das Programm schneidet bis auf die niedrigsten 8 Bit alle anderen Werte einfach ab (die führende 1 bei 256 in Binärdarstellung wird also „weggeworfen“). Gleiches Prinzip gilt dann für alle anderen Zahlen, die höher als 255 sind.



Lösung: Den analog eingelesenen Wert einfach mit 4 dividieren. Dadurch erhalten wir einen entsprechend umgewandelten Wert zwischen 0 und 255.

➔ 1023 / 4 = 255,75 → Da aber die Stellen nach dem Komma einfach abgeschnitten werden, erhalten wir hier trotzdem einen Maximalwert von 255!

3.2

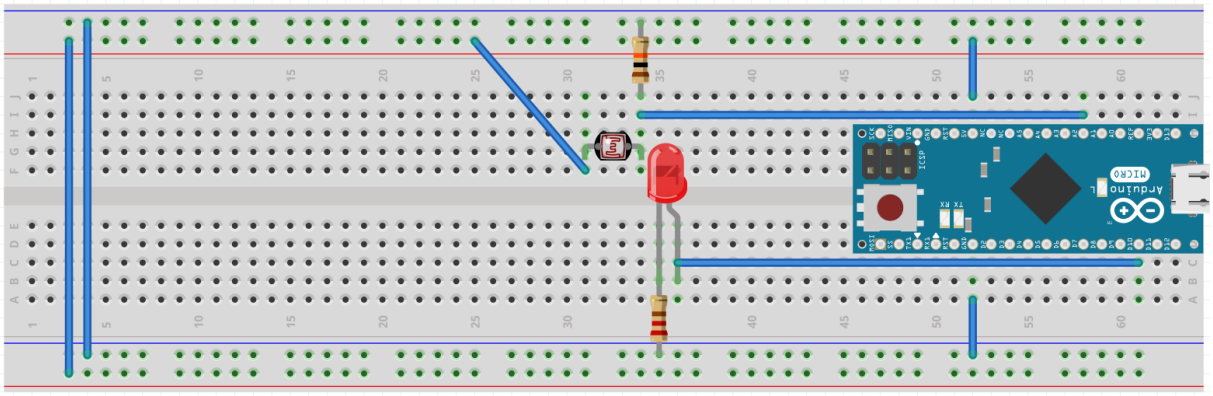
Allgemein:

- Photowiderstand Widerstandswerte (Photowiderstand ändert je nach Lichtstärke seinen Widerstand): Je dunkler, desto höher der Widerstand.

WIDERSTAND (KΩ) (UNGEFÄHRE WERTE)	HELLIGKEITSGRAD
0,1	Direkt an der Lichtquelle (sehr hell)
10	Bei normaler Raumbeleuchtung (hell)
30	Bei ausgeschaltetem Licht im Raum und auch sonst keiner großartigen Lichtquelle (dunkel)
150	Bei bestmöglicher Dunkelheit/ ausgeschaltetem Licht und Finger auf den Sensor (sehr dunkel)

Lösung/ Umsetzung:

- Wir benötigen einer Reihenschaltung/ Spannungsteiler. Diese erhalten wir, wenn wir einfach einen zweiten Widerstand hinter den Photoresistor bauen, sie „teilen“ sich dann die Spannung.
  → zusätzliche Beobachtung: Je höher der zweite Widerstand in der Reihenschaltung, desto deutlicher ist die Helligkeitsänderung der LED wahrzunehmen (getestet mit 220Ω, 10kΩ und 30kΩ)
- Die LED soll umso heller sein, je dunkler/ weniger Lichteinfall und umso dunkler, je mehr Lichteinfall. D.h. wir müssen eine Änderung im Code vornehmen, damit die LED bei Dunkelheit eine hohe Spannung bekommt (aktuell erhalten wir hohe analog-Werte, wenn es hell ist, der Photowiderstand also einen geringen Widerstand). Dazu „invertieren“ wir einfach den Wert für das analoge Input Signal einfach, indem wir vom Maximalwert 1023 den erhaltenen Wert abziehen.



Anmerkung:

Im Anhang ist das Video „automatische Raumbeleuchtung.mp4“ zu finden. Das soll einfach nur ein kurzes Programm zeigen, bei dem die LED angeht, wenn der Raum dunkel ist, und aus wenn das Licht an ist. Dazu habe ich einfach überprüft, welcher Inputwert vorliegt, wenn das Licht an bzw. aus ist und habe dann mit einer einfach if-Schleife abgefragt, ob ein bestimmter Schwellenwert überschritten wurde. Falls ja, geht die LED an, falls nein bleibt sie aus.

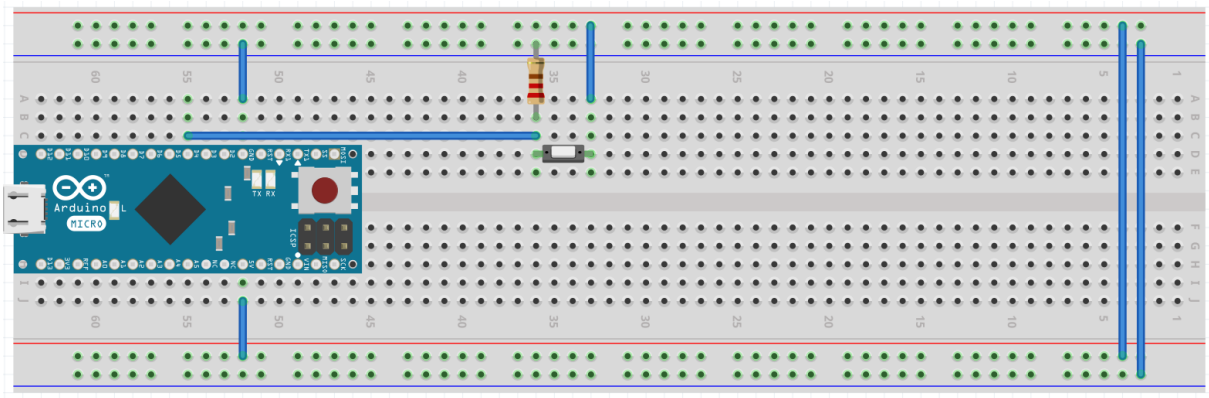
➔ Die Beleuchtung der Zimmerlampe war nicht hell genug, die Inputwerte haben sich nicht groß unterschieden, egal ob Licht an oder aus. Bei Licht an lag der Inputwert bei 248-249, bei ausgeschaltetem Licht bei 251-252. Durch Schwankungen hat die LED dann immer wieder geflackert, auch wenn die Zimmerlampe eigentlich an war. Aus diesem Grund habe ich für diesen Versuch die Schreibtischlampe zur Hand genommen, deren Helligkeit dann ausreichend war.

Übungsblatt 4 – Arduino

Für Videos/ Code siehe Ordner „Anhang“ und „Code“

4.1

Button wurde an den im Code vorgegebenen Pin (Pin 4) als Input angeschlossen (Wichtig: Pull-down Widerstand nicht vergessen!). Wenn Button gedrückt wird, wird quasi die Taste „a“ der Tastatur gedrückt.



Anmerkung:

Die Eingabe durch den vorgegebenen Code fühlt sich „unnatürlich“ durch den 1000ms Delay an. Durch folgende Änderung ist die Eingabe „wie gewohnt“.

```
void loop()
{
  Serial.print(digitalRead(BUTTON_PIN));
  // press 'a' as long as the button is down
  if(digitalRead(BUTTON_PIN) == HIGH)
  {
    // we can also send ASCII values
    Keyboard.press('a');
  }
  else{
    Keyboard.release('a');
  }
}
```

4.2

Idee

Für das Spiel Minecraft soll ein eigenes Eingabegerät entwickelt werden, welches die wichtigsten Ingame-Funktionen umfasst. Dazu zählen:

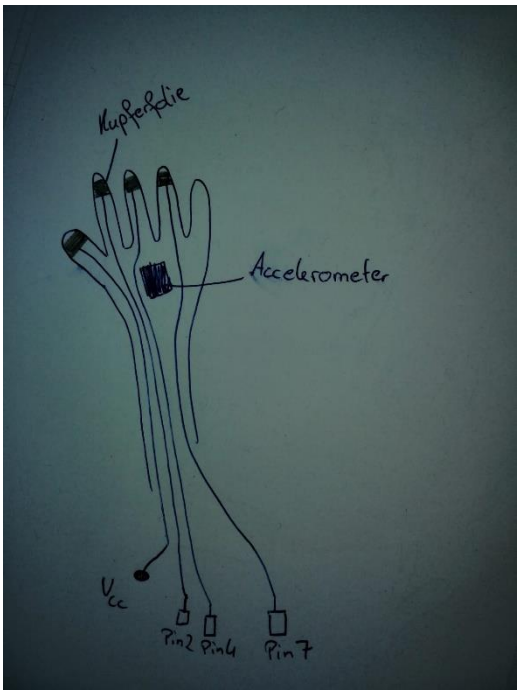
- Laufen (Taste ‚w')
- Umsehen (Maus)
- Abbauen/ Angreifen (linke Maustaste)
- Block setzen (Rechte Maustaste)

Ich habe mich für eine Art „Gaming-Handschuh" entschieden, der die genannten Funktionen abdeckt.

Funktion	**Eingabe normal**	**Neue Eingabe per**	**Am Handschuh**
Laufen	w-Taste	„Button"	Berührung von Zeigefinger und Daumen
Umsehen	Maus	3-Axis Accelerometer	Handschuhh bewegen
Abbauen/ Angreifen	Linke Maustaste	„Button"	Berührung von Mittelfinger und Daumen
Block setzen	Rechte Maustaste	„Button"	Berührung von Ringfinger und Daumen

Umsetzung

1. *Skizzieren/ Visualisieren meines Plans*

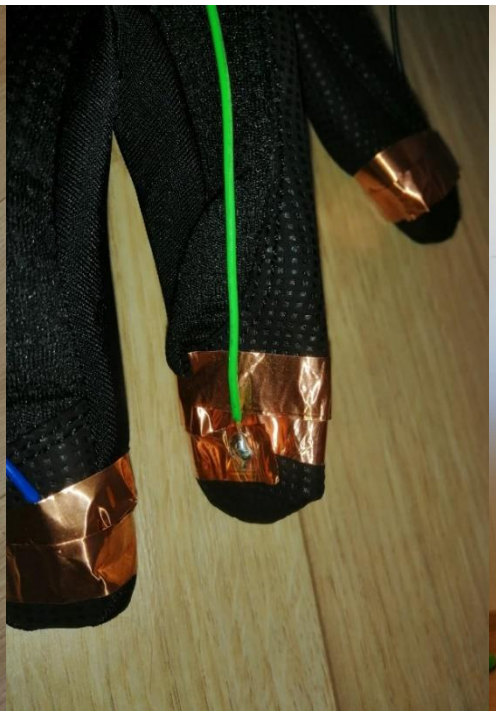


- Der Daumen ist mit der Spannungsquelle verbunden. Sobald einer der Finger mit dem Daumen in Berührung kommt, ist fließt Strom an den jeweiligen Pin des Arduinos. Der Zeigefinger ist soll also an Pin2 anliegen und soll bei Kontakt mit dem Daumen das Event

„Laufen“ bzw. die w-Taste simulieren (gleiches Prinzip für die anderen zwei Finger). Um den Anschluss/ die Funktion des Accelerometers kümmere ich mich später, ich will zunächst erstmals die Buttons zum laufen kriegen.

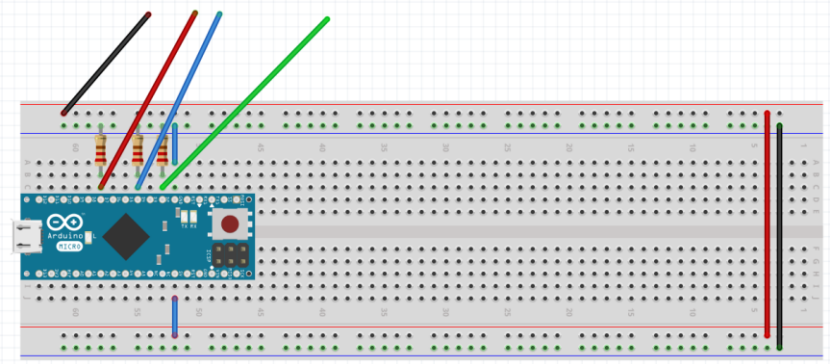
2. *Löten*

- Als nächstes habe ich angefangen, die benötigten Drähte/ Kabel/ Kupferfolien zusammenzulöten, und diese dann auch gleich am Handschuh angebracht (dabei habe ich auch gleich geschaut, dass die verwendeten Litzen lange genug sind, dass man später auch etwas Freiraum in der Armbewegung hat/ falls immernoch zu kurz, kann man einfach noch eine Verlängerung zusammenlöten). Für das Accelerometer habe ich auch gleich drei Stecker verlängert, die ich später für die Daten der x, y, und z-Achse benötige


3. *Schaltung für die Buttons planen und umsetzen*

- Jetzt habe ich versucht, die Schaltung/ Anschlüsse am Steckbrett umzusetzen. Das ging relativ schnell, da die Finger ja jetzt eigentlich einfach nur als Schalter angesehen werden können, die bei Berührung (mit der Spannungsquelle, also dem Daumen) zu einem gewissen Pin Strom leiten sollen, ansonsten nicht. Wichtig war hier aber wieder, die Pull-Down Widerstände nicht zu vergessen, dass kein „schwebender“ Zustand an den Pins vorliegt, wenn kein Strom anliegt, sondern diese klar definiert 0V empfangen.
- Zeigefinger (Grüner Draht) soll an Pin 2, Mittelfinger (Blauer Draht) an Pin 4, Ringfinger (Roter Draht) an Pin 7, Daumen (schwarzer Draht) an Vcc



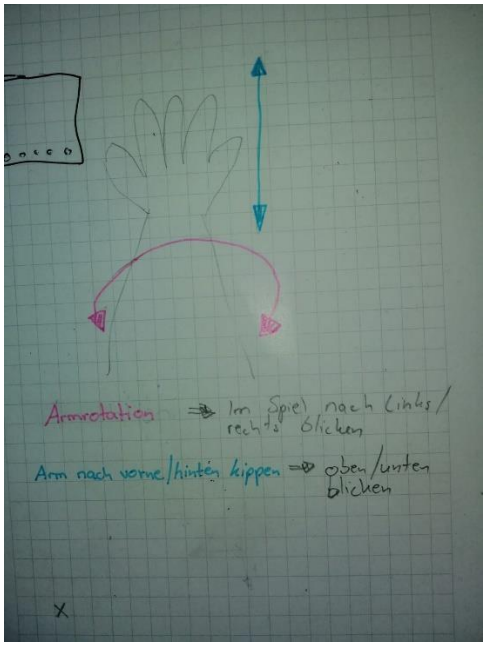
4. *Code für die Fingerbuttons*

- Die Methode checkFingerInput() wird in jedem loop-Durchlauf aufgerufen
- Hier werden die drei Werte für die Finger ausgelesen und dann die passende Aktion ausgeführt
- Ich habe mich für drei aufeinanderfolgende if-else if Schleifen entschieden, weil so mehrere Aktionen gleichzeitig ausgeführt werden können. Ich kann also z.B. in Minecraft gleichzeitig vorwärts laufen und einen Block setzen etc.
- Für die Aktion Block setzen wurde ein Timer eingebaut, sonst ist es relativ unkontrollierbar, wie viele Blöcke man setzt. So dauert es erst immer 200ms bis der Spieler einen neuen Block setzen kann.
- **Anmerkung**: Anfangs konnte man Blöcke nicht abbauen. Die Taste funktionierte zwar, allerdings wurde sie nicht lange genug gedrückt, sondern immer nur ganz kurz und dann sofort wieder released (siehe Video Zwischenupdate.mp4). Das lag daran, dass die click() Funktion der Maus verwendet wurde, die nur einmal Klickt und sofort wieder released. Diese musste lediglich Gegen die press() Methode ausgetauscht werden, bei der man dann selbst bestimmen konnte, wann wieder released werden soll.

```
void checkFingerInput(){
  walk = digitalRead(POINTERFINGER);
  attack = digitalRead(MIDDLEFINGER);
  setBlock = digitalRead(RINGFINGER);
  if(walk == HIGH && ! isWalking){
    isWalking = true;
    Keyboard.press(KEY_WALK);
  }
  else if(walk == LOW){
    isWalking = false;
    Keyboard.release(KEY_WALK);
  }
  if(attack == HIGH && ! isAttacking) {
    isAttacking = true;
    Mouse.press();
  }
  else if(attack == LOW){
    isAttacking = false;
    Mouse.release();
  }
  if(millis() - lastMillis2 >= SETBLOCK_DELAY){
    lastMillis2 = millis();
    if(setBlock == HIGH) {
      Mouse.click(MOUSE_RIGHT);
    }
  }
}
```

5. *Accelerometer verstehen und testen*

- Für mein Eingabegerät will ich, dass…
  … der Spieler nach links/ rechts blickt, wenn der Arm nach rechts/ links rotiert wird
  … der Spieler nach oben/ unten blick, wenn der Arm nach vorne/ hinten gekippt wird
  ➔ Dafür will ich das Accelerometer hernehmen



Notizen zur Funktion des Accelerometers:

- damit kann man die Beschleunigung auf die 3 Raumachsen (x,y,z) messen → Ausrichtung steht auf dem Acc mit drauf.
- in unserem Fall 3-5V Inputspannung akzeptabel
- 3 Pins für jede Achse als Output
- Wenn z-Achse nach oben schaut, wirkt die Gravitation auch nur auf die Z-Achse
  ➔ Resultat der eingehenden Spannung: x: 344, y: 338, z: 416
  ➔ 416 entspricht ca. 2V
  ➔ 340 entspricht ca. 1.65V
- Wenn y-Achse nach oben schaut: x: 348, y: 405, z: 346
- Wenn x-Achse nach oben schaut: x: 409, y: 335, z: 346
  ➔ Wenn x, y- oder z-Achse nach unten schauen, sind Werte von ca. 265 (1.3V) zu erwarten
  ➔ Das sind allerdings nur die Werte, die wir bekommen, wenn sich der Sensor in Ruhe befindet. Bei Bewegung wirkt dementsprechend mehr oder weniger Kraft auf die Achsen
  ➔ Im freien Fall wird für einen kurzen Moment keine Beschleunigung an den 3 Achsen gemessen/ bzw. Werte von ca. 335
- Wenn ich Sensor entlang der x-Achse um 45 Grad kippe, erhalte ich für y- und z- nahezu identische Werte, da die Kraft auf beide Achsen gleich ist

Anschließen des Accelerometers an den Arduino/ ans Breadboard:

- Relativ intuitiv, Eingang für Vcc, Ausgang für Ground, sowie drei Ausgänge für x, y und z-Wert vorhanden. Die Werte müssen einfach nur an passende analoge Pins des Arduinos angeschlossen werden (ich wähle A1, A2, A3)

Lösung:

- Ich baue das Accelerometer so auf den Handschuh, dass im Normalfall (ausgestreckter Arm, keine Rotation oder Neigung) die z-Achse nach oben schaut
- Die x-Achse hätte dann einen Werte von ungefähr 340, die y-Achse genauso
- Wenn ich jetzt meine Hand nach links/rechts rotiere, ändert sich theoretisch nur der Wert der x-Achse.
- Wenn ich meine Hand nach vorne oder hinten neige, dann ändert sich nur der y-Achsen-Wert
- Sobald ein Threshold-Wert überschritten wird/ ich also genug Änderung an der x- oder y-Achse habe, bewege ich die Maus in die entsprechende Richtung (z-Achse kann komplett ignoriert werden)
- Damit ich die richtige Mausgeschwindigkeit finden konnte, die sich einigermaßen benutzen lässt, habe ich einfach herumprobiert, bis ich einen meiner Meinung nach passenden Wert hatte, der nicht zu einer kompletten Übersteuerung führt, aber auch nicht ewig langsam ist.

```
void moveMouse(int x, int y){
  if(x > UPPER_THRES){
    Mouse.move(-MOUSE_SPEED, 0, 0); // move mouse right
  }
  else if(x < LOWER_THRES){
    Mouse.move(MOUSE_SPEED, 0, 0); // move mouse left
  }
  if(y > UPPER_THRES){
    Mouse.move(0, MOUSE_SPEED, 0); // move mouse down
  }
  else if(y < LOWER_THRES){
    Mouse.move(0, -MOUSE_SPEED, 0); // move mouse up
  }
}
```

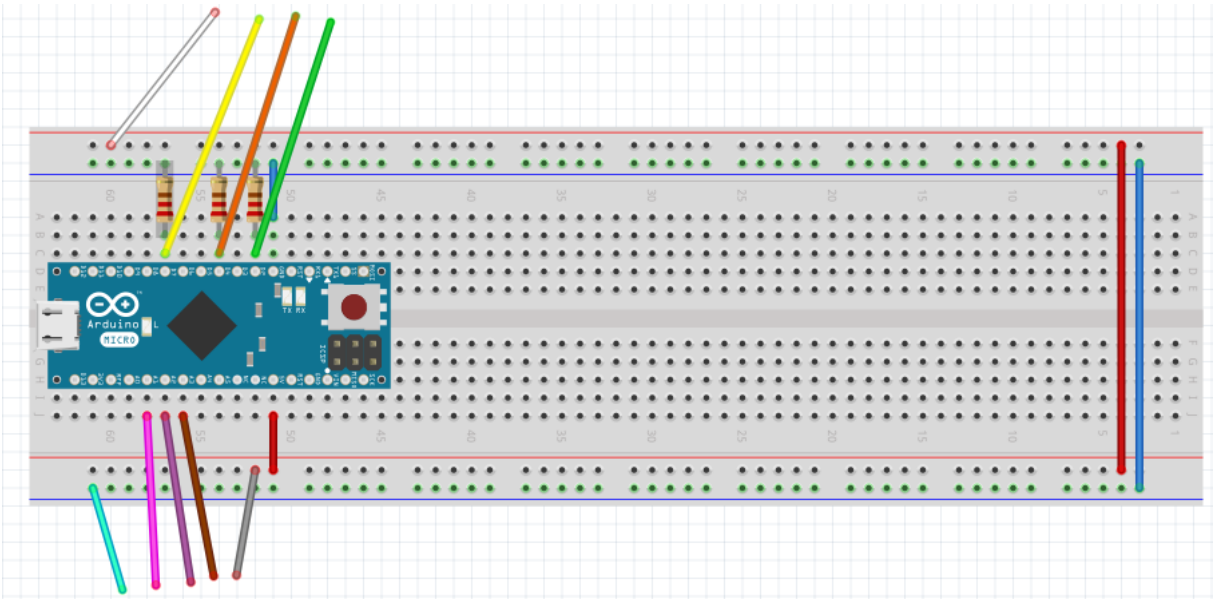
Notizen:

- Zunächst hatte ich versucht, in jedem loop() Durchlauf einen Wert für x und y auszulesen und diesen dann vom vorherigen x und y Wert zu subtrahieren, um die Differenz zum vorherigen Zustand zu berechnen und dann den Mauscursor basierend auf diesen Werten zu bewegen. Allerdings gab es hier mehrere Probleme. Zum einen hatte der Curser selbst im Ruhestand des Sensor „herumgewackelt“, da immer wieder kleinste Änderungen an dem Accelerometer wahrgenommen wurden (siehe Video „Sensor Update 1.mp4“). Das Problem konnte sich einigermaßen gut beheben lassen, indem ich eine Art Deadzone in den Code eingebaut habe, damit der Sensor erst um einen gewissen Grad bewegt werden musste, damit die Maus reagiert. Das hat auch geklappt (siehe Video „Sensor Update 2.mp4“), allerdings war die ganze Steuerung mit dieser Methodik allgemein sehr unzuverlässig und schwer bedienbar. Deshalb habe ich mich dann dazu entschieden. Die Maus immer dann in eine Richtung zu bewegen, sobald ein bestimmter Threshold-Wert für x und y überschritten wird.

6. *Zusammenfügen aller Elemente*

- Der finale Code, sowie ein Video zum Resultat sind im Anhang zu finden
- Zunächst wurde das Accelerometer am Handschuh angebracht (mit Klebeband), möglich so, dass die z-Achse bei ausgestrecktem Arm senkreckt zum Boden/ nach oben zeigt

- Abkleben offener Lötstellen der zusammengelöteten Litzen (hatte zu dem Zeitpunkt noch keine Schrumpfschläuche)
- Anschließend wurde alles ans Breadboard angesteckt



➔ Weiß: Daumen
➔ Gelb: Ringfinger
➔ Orange: Mittelfinger
➔ Grün: Zeigefinger
➔ Cyan: Ground für Accelerometer
➔ Rosa: x-Wert des Accelerometers
➔ Lila: y-Wert des Accelerometers
➔ Braun: z-Wert des Accelerometers (wird allerdings nicht benötigt)
➔ Grau: Spannungsversorgung für Accelerometer