University of Applied Sciences and Arts Northwestern Switzerland
School of Engineering

Fast Lights

clab

Christian Seiler

Windisch, 07.12.2018

BSc Computer Sciences
iCompetence
Brühlmattstrasse 1
CH-5610 Wohlen
T +41 79 289 31 93
christian.seiler@students.fhnw.ch
www.fhnw.ch

Inhaltsverzeichnis

1 Einleitung

Als Abschlussprojekt des Moduls clab (Computerlabor) galt es ein Projekt in Verwendung des Microcontroller Arduino zu gestalten. Mein Projekt mit dem Titel "Fast Lights" ist in den Ursprüngen des eigentlichen Projekts mit Thomas Frey, Temperaturmessung mit LCD-Anzeige, entstanden. Dies hauptsächlich durch die Unterschiede in Wissensstand und Erfahrung mit Microcontroller und der Arduino Umgebung zwischen meinem Teamkollegen und mir. Mit anderen Worten, es kam Langeweile auf und ich begann mit meinen eigenen Arduino-Komponenten zu basteln.

2 Vorgehen

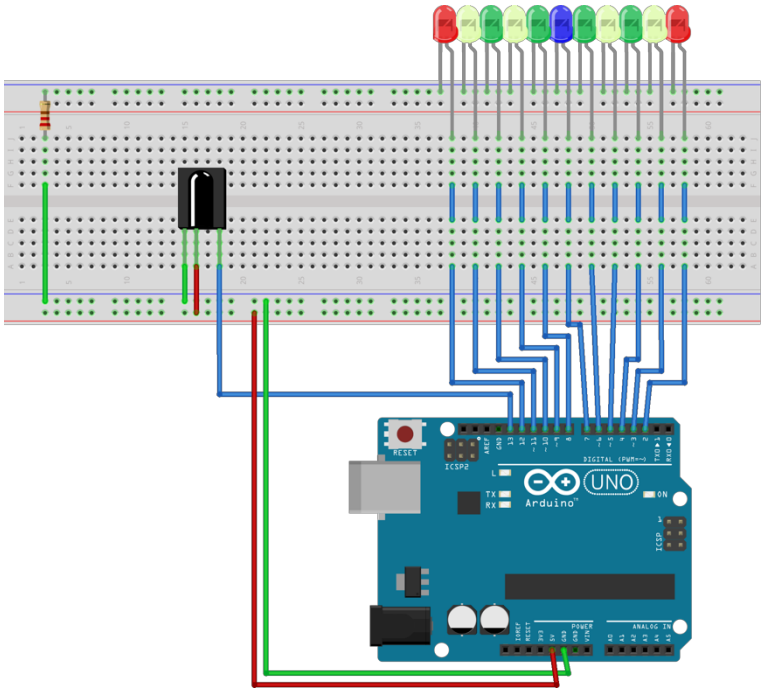
Die Entwicklung von der ursprünglichen Tickerei zum fertigen Prototypen kann grob in drei Abschnitte geteilt werden.

1. LEDs und IR Remote
2. Einfaches Reaktionsspiel
3. Double Player und OLED Display

3 Aufbau

3.1 LEDs und IR Remote

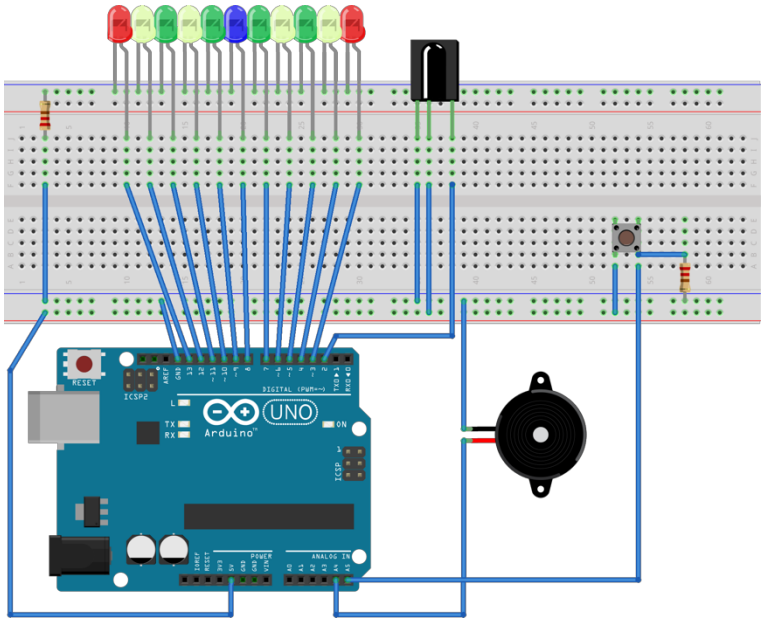
Die erste Phase bestand daraus eine Reihe von LED über die IR Remote zu kontrollieren. Dabei sind die LEDs aufgereiht und einzeln mit einem digitalen Port des Arduino UNO verbunden. Weiter ist der IR Empfänger an einen digitalen Port gebunden, wodurch die Eingaben durch die Fernbedienung ausgelesen werden.



Diese Eingaben werden anschliessend in verschiedene Aktionen übersetzt. Die erste dieser Aktionen ist das nacheinander Aufleuchten der LEDs von einer Seite zur

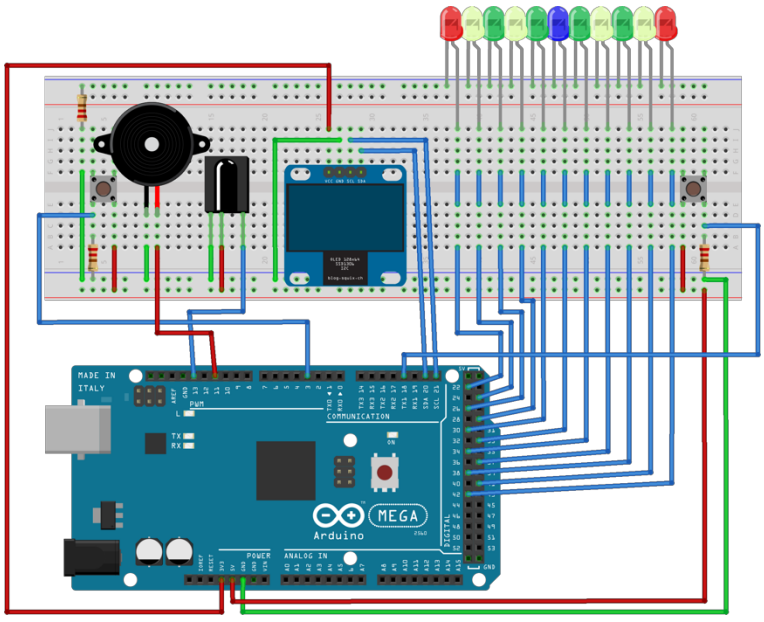
anderen und wieder zurück. Eine weitere Aktion ist das Aufleuchten der LEDs von links nach rechts. Sobald die letzte LED, rechts, erreicht wurde, begann die erste LED, links, die Sequenz wieder von neuem. Diese Aktion ist auch spiegelverkehrt vorhanden. Als letzte Aktion ist das zufällige Aufleuchten einer LED.

3.2 Einfaches Reaktionsspiel



Die Zufallssequenz der LEDs folgte mit der Idee eines Reaktionsspiels. Mit Druckknopf und Buzzer ist dies auch kurzerhand umgesetzt. Beide Komponenten sind über einem analogen Port mit dem Arduino verbunden. Wenn nun der Knopf gedrückt wird, wenn die mittlere LED (blau) leuchtet, folgt ein kurzer Ton. Die LEDs leuchten, bereits seit Beginn der Entwicklung, während 250ms. Noch während dieser Entwicklungsphase war klar, dass dies nicht das Endstadium sein kann. Ich wollte mehr! Das Problem war allerdings, dass alle digitalen Ports des Arduino belegt sind und die analogen Ports für diese Verwendung nicht geeignet sind.

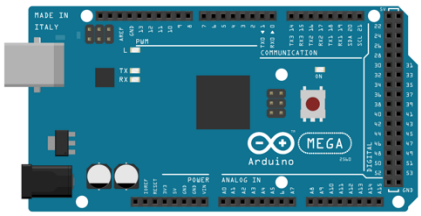
3.3 Double Player und OLED Display



Die Lösung lag im Upgrade des Arduino UNO auf den grösseren Arduino MEGA. Die nun viel grössere Anzahl digitaler Ports erlaubte nun die weitere Entwicklung des Spiels. Nachdem das bisherige Projekt auf den neuen Microcontroller angepasst war, konnte das Spiel mit einem zweiten Pushbutton auf zwei Spieler erweitert werden. Zusätzlich kann der Spielstand auf dem OLED Display dargestellt werden.

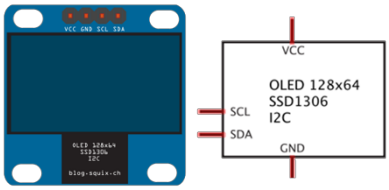
3.4 Verwendete Elemente

Arduino MEGA 2560 — 1 Stk



ATmega2560 Microcontroller
54 digitale I/O Pins
16 analoge Input Pins
16MHz

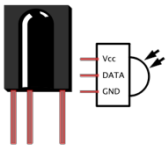
OLED Display — 1 Stk



128x64 Pixel
Monochrome

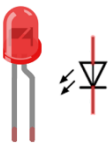
Library: Adafruit_GFX & Adafruit_SSD1306

IR Receiver & IR Remote Control — 1 Stk



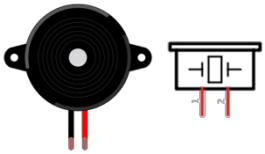
Library: IRremote & IRremoteInt

LEDs — 11 Stk

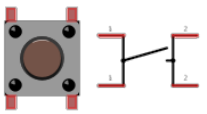


Verschiedene Farben

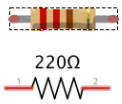
Piezo speaker — 1 Stk



Pushbutton — 2 Stk



220Ω Resistor — 3 Stk



4 Herausforderungen

4.1 Upgrade von UNO auf MEGA

Nachdem die Entscheidung gefallen war vom Arduino UNO auf MEGA zu wechseln, musste neben den physischen Komponenten auch der Code angepasst werden. Obwohl beide Geräte ähnlich aufgebaut sind, gibt es doch einige Unterschiede. Wo die Verbindungskabel für die LEDs beim UNO noch alle sequenziell in einer Reihe gesteckt werden konnten, haben diese beim MEGA keinen Platz mehr. Dies ist durch die zweireihige Anordnung der Ports 22 bis 53. Dieses Platzproblem konnte allerdings durch eine Zick-Zack-Belegung behoben werden, was allerdings den Code etwas verkompliziert.

4.2 Auslesen der Fernbedienung

Der Ausgabewert der Fernbedienung war anfangs ein grosses Fragezeichen. Obwohl man denken könnte, dass man nur alle Knöpfe durchdrücken und einfach ablesen könne was der entsprechende Wert ist, waren diese Werte anfangs jenseits von konsistent. Dies hat sich allerdings mit einigen Tests eingespielt.

4.3 Spielstatus

In der letzten Entwicklungsphase gab es vier Spielstati: *SINGLEPLAYER*, *DOUBLEPLAYER*, *WIN*, *FINISHED*. Das Spiel beginnt mit dem *FINISHED* Status und wartet auf einen Spielstart via der Fernbedienung. Während des Spiels ist entweder *SINGLEPLAYER* oder *DOUBLEPLAYER* aktiv. Sobald ein Spieler 10 Punkte hat, wechselt das Spiel auf WIN. Es wird eine Siegesmelodie abgespielt und wechselt schliesslich zurück auf *FINISHED*. Hier war die Herausforderung, dass der Spielstatus an vielen verschiedenen Orten im Spiel verwendet wird und anfangs war das Problem wann wie welcher Status gesetzt wird und dass kein Branch vergessen wird. So kam es, dass die Siegesmelodie im Loop abgespielt wurde, weil der Status nicht gewechselt wurde.

4.4 Scrolling Text

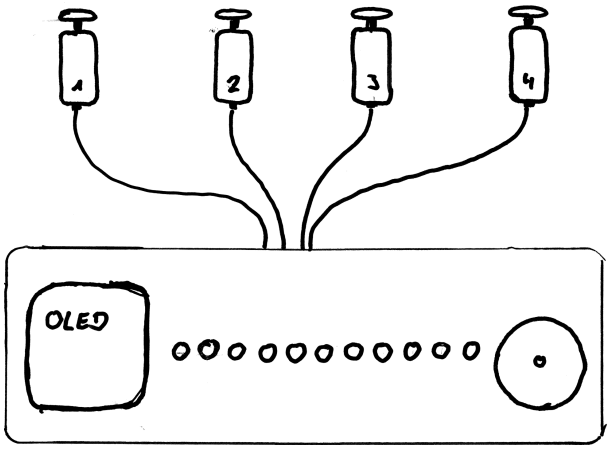
Das OLED Display ist mit 128 x 64 Pixeln sehr klein und der Text wird aktuell mit Schriftgrösse 1 (7 Pixel) angezeigt damit der gewünschte Text Platz hat. Die für das Display verwendete Library unterstützt Scrolling Text, aber doch nicht. Alles was länger als die Displaybreite ist, wird einfach abgesägt.

5 Lessons learned

Die Kombination von Elektrotechnik und deren Programmierung ist spannend aber nicht einfach. Insbesondere wenn der Code mehrere hundert Zeilen erreicht. Das Refactoring des Codes und Einführen von C++ Dateien war schwieriger als anfänglich gedacht. Vor allem das Erstellen einer eigenen C++ Klasse für das Display war nicht wirklich einfach, allerdings hat dies das Verständnis über die Klasse in C++ gestärkt.

6 Was man sonst noch tun könnte

Es gibt vordergründig drei Möglichkeiten, wie das Spiel in einem nächsten Schritt weiterentwickelt werden könnte:



Durch ein **Case** und Batterie würde das Spiel portabel einsetzbar und wäre zum Beispiel an einem Familienfest ein Hingucker mit grossem Unterhaltungsfaktor. Insbesondere können so Kinder mit der Technik vertraut gemacht werden. Durch das Entfernen der Druckknöpfe vom Breadboard und das Anbringen dieser mittels eines (einsteckbaren) Kabels, erlaubt man, noch mehr Spieler einzubringen.

Aktuell ist die **Geschwindigkeit** der LEDs auf 250 ms festgelegt. Dies ist eine relativ kurze Zeit, um den Knopf zu drücken, zu schnell für die meisten Kinder. Um das Spiel stufengerecht zu machen, könnte diese Geschwindigkeit mittels verschiedener Schwierigkeitsstufen über die Fernbedienung gesteuert werden. Weiter wäre auch eine Spielform möglich, bei der die Geschwindigkeit stetig gesteigert und dabei die Zeit gemessen wird, bis ein Spieler daneben drückt.

Wie bereits erwähnt, ist der Text auf dem Display sehr klein. Somit wäre die Implementation eines **Scrolling Texts** eine Lösung, um den Text grösser abzubilden.

7 Source Code

Die verwendeten Libraries können in der Arduino IDE mit dem Library Manager geladen werden. Dies sind:

- Adafruit GFX Library (v. 1.3.4) von Adafruit
- Adafruit SSD1306 (v. 1.2.9) von Adafruit
- IRremote (v.2.2.3) von shirriff

Der komplette Source Code ist hier erhältlich: https://github.com/Christian-Seiler/Fast-Lights

8 Quellen

OLED I2c Display	http://bit.ly/2B0IX9p
Writing a Library for Arduino	http://bit.ly/2QJTDTM
Class code and header files	http://bit.ly/2SAxDIt

9 Anhang

9.1 FastLights.in

```
#include <IRremote.h>
#include <IRremoteInt.h>
#include "Direction.cpp"
#include "Game.cpp"
#include "Remote.cpp"
#include "Display.h"

#define RIGHTBUTTON   3
#define LEFTBUTTON    18
#define BUZZER        11
#define REMOTE        13

int pointRight = 0;
int pointLeft = 0;

// Lamp things
const int LAMPS [] = {22, 25, 26, 29, 30, 33, 34, 37, 38, 41, 42};
const int COUNT = (sizeof(LAMPS) / sizeof(LAMPS[0]));
int lamp;
const int maxPoints = 10;

// Timing
int delayMS = 200;
const int buzzMS = 300;

// IR stuff
IRrecv ir(REMOTE);
decode_results results;

Game game = FINISHED;
Direction dir = DIR_LEFT;
bool pressed = false;
Remote previous;
Display display;

void setup() {
  Serial.begin(9600);
  display.init();

  // IR Remote setup
  ir.enableIRIn();
  ir.blink13(false);

  // light up each led at startup
  for (int i : LAMPS) {
    pinMode(i, OUTPUT);
    digitalWrite(i, HIGH);
    delay(200);
    digitalWrite(i, LOW);
  }
```

```
  // Pushbutton setup
  pinMode(LEFTBUTTON, INPUT);
  pinMode(RIGHTBUTTON, INPUT);
  attachInterrupt(digitalPinToInterrupt(LEFTBUTTON),
                  leftButtonPressed,
                  CHANGE);
  attachInterrupt(digitalPinToInterrupt(RIGHTBUTTON),
                  rightButtonPressed,
                  CHANGE);

  // give me a break
  delay(2000);
}

void loop() {
  // get the current remote button
  if (ir.decode(&results)) { ir.resume(); }

  if (game == FINISHED) {
    display.showInstructions();
    LED(game);
    game = setGame();
  } else if (game == WIN) {
    displayScore();
  } else {
    LED(game);
    displayScore();
  }
}

// Interrupt routine for the left pushbutton (aka player two)
void leftButtonPressed() {
  if (lamp == COUNT / 2 && pressed == false) {
    pressed = true;
    tone(BUZZER, 392, buzzMS);
    pointLeft += 1;
    if (pointLeft >= maxPoints) {
      game = WIN;
    }
  }
}

// Interrupt routine for the right pushbutton (aka player one)
void rightButtonPressed() {
  if (lamp == COUNT / 2 && pressed == false) {
    pressed = true;
    tone(BUZZER, 440, buzzMS);
    pointRight += 1;
    if (pointRight >= maxPoints) {
      game = WIN;
    }
  }
}
```

```
  // start game if remote has been pressed
Game setGame() {
  Remote remote = (Remote) results.value;
  if (remote != previous && remote == ONE) {
    previous = remote;
    display.setSinglePlayerGame();
    delay(2000);
    return SINGLEPLAYER;
  } else if (remote != previous && remote == TWO) {
    previous = remote;
    display.setDoublePlayerGame();
    delay(2000);
    return DOUBLEPLAYER;
  }
  return game;
}

void LED(Game game) {
  // turn off the lamp that is on
  digitalWrite(LAMPS[lamp], LOW);

  if (game == FINISHED) {
    // light up led left to right to left
    if (dir == DIR_LEFT) {
      if (lamp == COUNT-1) {
        dir = DIR_RIGHT;
        lamp -= 1;
      } else {
        lamp += 1;
      }
    } else {
      if (lamp == 0) {
        dir = DIR_LEFT;
        lamp += 1;
      } else {
        lamp -= 1;
      }
    }
  } else {
    // game is on
    // get a random number between 0 and the number of lamps
    int n = random(COUNT);
    // if the random lamp was the previous lamp, choose another
    lamp = n == lamp ? n + random(-1, 2) : n;
    pressed = false;
  }

  // turn on lamp
  digitalWrite(LAMPS[lamp], HIGH);
  delay(delayMS);
}
```

```
// Display Score for each state (single, double, win)
void displayScore() {
  if (game == SINGLEPLAYER) {
    display.showSingleScore(pointRight);
  } else if (game == DOUBLEPLAYER) {
    display.showDoubleScore(pointRight, pointLeft);
  } else if (game == WIN) {
    finish();
    display.showFinalScore(pointRight, pointLeft);
    delay(2000);
    game = FINISHED;
  }
}

// reset score and play win melody
void finish() {
  pointRight = 0;
  pointLeft = 0;
  tone(BUZZER, 415, 300);
  delay(350);
  tone(BUZZER, 349, 500);
  delay(350);
  tone(BUZZER, 466, 700);
  delay(700);
}
```

9.2 Remote.cpp

```
// Return values of the IR Remote
enum Remote : unsigned long {
  CHNEG =   0xFFA25D,
  CH =      0xFF629D,
  CHPLUS =  0xFFE21D,
  LEFT =    0xFF22DD,
  RIGHT =   0xFF02FD,
  PLAY =    0xFFC23D,
  MINUS =   0xFFE01F,
  PLUS =    0xFFA857,
  EQ =      0xFF906F
  ONEPLUS = 0xFF9867,
  TWOPLUS = 0xFFB04F,
  ZERO =    0XFF6897,
  ONE =     0xFF30CF,
  TWO =     0xFF18E7,
  THREE =   0xFF7A85,
  FOUR =    0xFF10EF,
  FIVE =    0xFF38C7,
  SIX =     0xFF5AA5,
  SEVEN =   0xFF42BD,
  EIGHT =   0xFF4AB5,
  NINE =    0xFF52AD
};
```

9.3 Display.h

```
#ifndef Display_h
#define Display_h

#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

class Display {
  public:
    Display();
    void init(void);
    void showInstructions(void);
    void showSingleScore(int pointRight);
    void showDoubleScore(int pointRight, int pointLeft);
    void showFinalScore(int pointRight, int pointLeft);
    void setSinglePlayerGame(void);
    void setDoublePlayerGame(void);
  private:
    Adafruit_SSD1306 display;
};
#endif
```

9.4 Display.cpp

```
#include "Display.h"

#define OLED_SCL 21
#define OLED_SDA 20
#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels
#define OLED_RESET 4

Display::Display(void) {
  Adafruit_SSD1306 display(SCREEN_WIDTH,
                           SCREEN_HEIGHT,
                           &Wire,
                           -1,
                           400000UL,
                           100000UL);
}

// initializing oled display
void Display::init() {
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("Copyright (c) 2018\nChristian\nSeiler\nServices");
  display.setCursor(0, 56);
  display.print("christianseiler.ch");
  display.display();
}
```

```
void Display::showInstructions() {
  display.clearDisplay();
  display.setCursor(0, 0);
  display.println("Welcome to\nFast Lights!");
  display.setCursor(0, 30);
  display.println("Press:");
  display.println("1 for Single Player");
  display.println("2 for Double Player");
  display.display();
}

// Display Score for each state (single, double, win)
void Display::showSingleScore(int pointRight) {
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("Score");
  display.setCursor(0, 20);
  display.print("Right");
  display.setCursor(100, 20);
  display.print(pointRight);
  display.display();
}

void Display::showDoubleScore(int pointRight, int pointLeft) {
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("Score");
  display.setCursor(0, 20);
  display.print("Left");
  display.setCursor(100, 20);
  display.print(pointLeft);
  display.setCursor(0, 40);
  display.print("Right");
  display.setCursor(100, 40);
  display.print(pointRight);
  display.display();
}
void Display::showFinalScore(int pointRight, int pointLeft) {
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("The Winner is");
  display.setCursor(0, 20);
  display.print(pointLeft > pointRight ? "LEFT" : "RIGHT");
  display.display();
}

void Display::setSinglePlayerGame(void) {
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("Playing");
  display.setCursor(20, 0);
  display.print("Single Player");
}
```

```
void Display::setDoublePlayerGame(void) {
  display.clearDisplay();
  display.setCursor(0, 0);
  display.print("Playing");
  display.setCursor(20, 0);
  display.print("Double Player");
}
```

9.5 Game.cpp

```
// Game status
enum Game : int {
  FINISHED = 0,
  SINGLEPLAYER = 1,
  DOUBLEPLAYER = 2,
  WIN = 3
};
```

9.6 Direction.cpp

```
enum Direction {
  DIR_RIGHT,
  DIR_LEFT
};
```