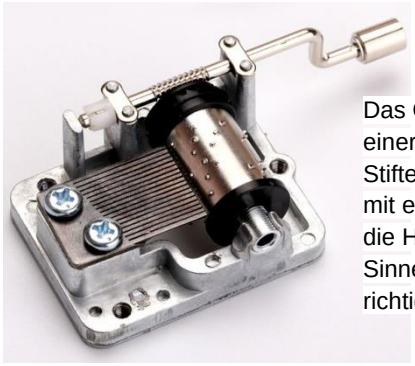
# Glockenspiel (Eine kleine Projektskizze)

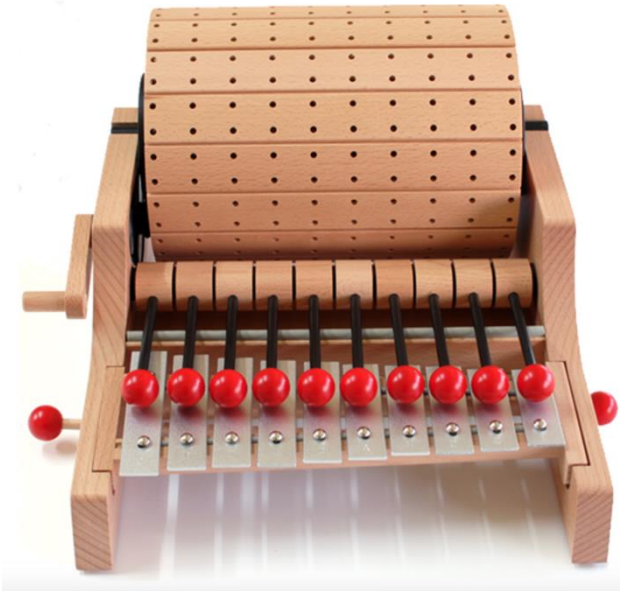
Die Idee dazu lieferte das Gloggomobil im Kinderzimmer, das vor vielen Jahren meine Töchter, und heute die Enkelkinder begeistert.

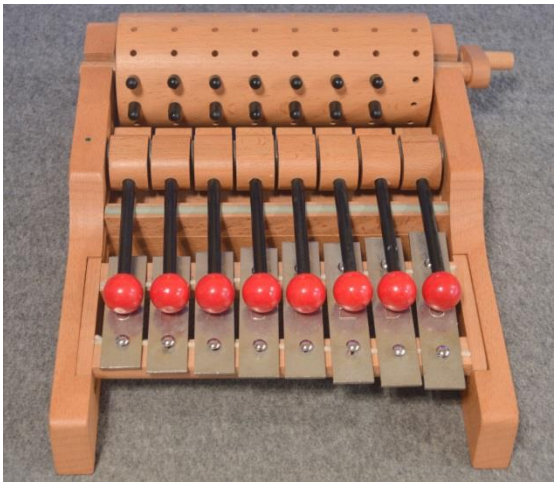


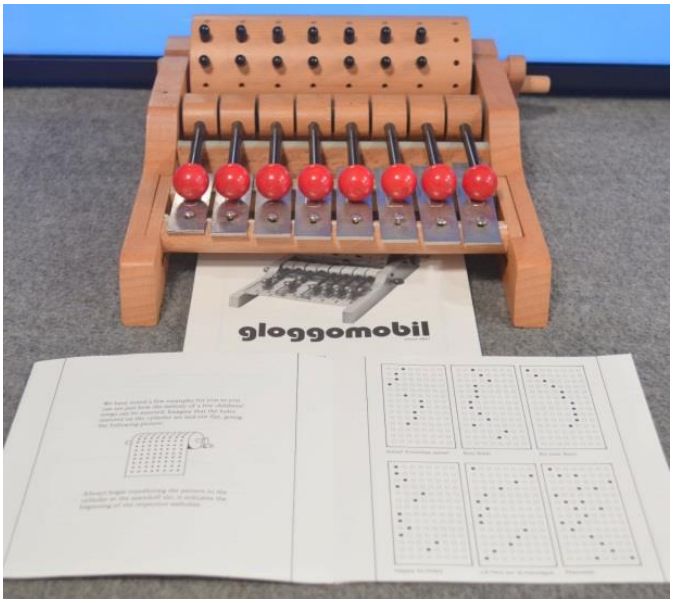
Das Gloggomobil ist nach dem Prinzip einer Spieluhr gebaut. Die schwarzen Stifte in den Löchern der Walze heben mit einem einfachen Drehmechanismus die Hämmerchen und lassen sie (im Sinne der programmierten Melodie) zum richtigen Zeitpunkt fallen.

Wird die Drehbewegung von einem Motor-Antrieb übernommen, wird aus der Spieluhr ein automatisches Glockenspiel.

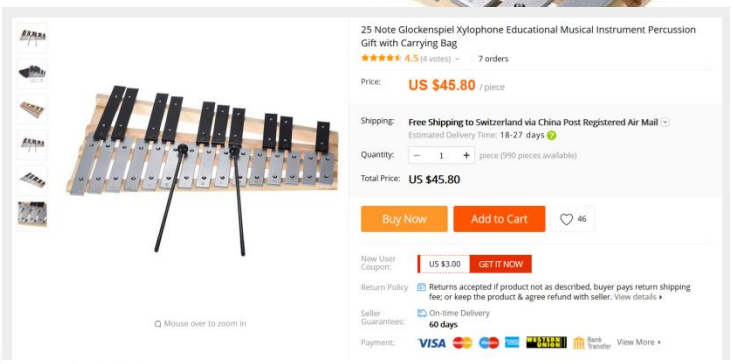
Im vorliegenden Projekt geht es darum, die Funktionsweise des Glockenspiels mit modernen Bausteinen der Mikrocomputertechnik so abzubilden, dass auch hier das Endprodukt in der Lage ist, vorprogrammierte Melodien automatisch abspielen zu lassen.
Im digitalisierten Modell werden die Melodien binär gespeichert und können beliebig verändert werden. Ein Micro-Controller steuert mit Hilfe dieser binären Informatinen kleine Servo-Motoren, die die Hämmerchen im richtigen Zeitpunkt abheben und fallen lassen.

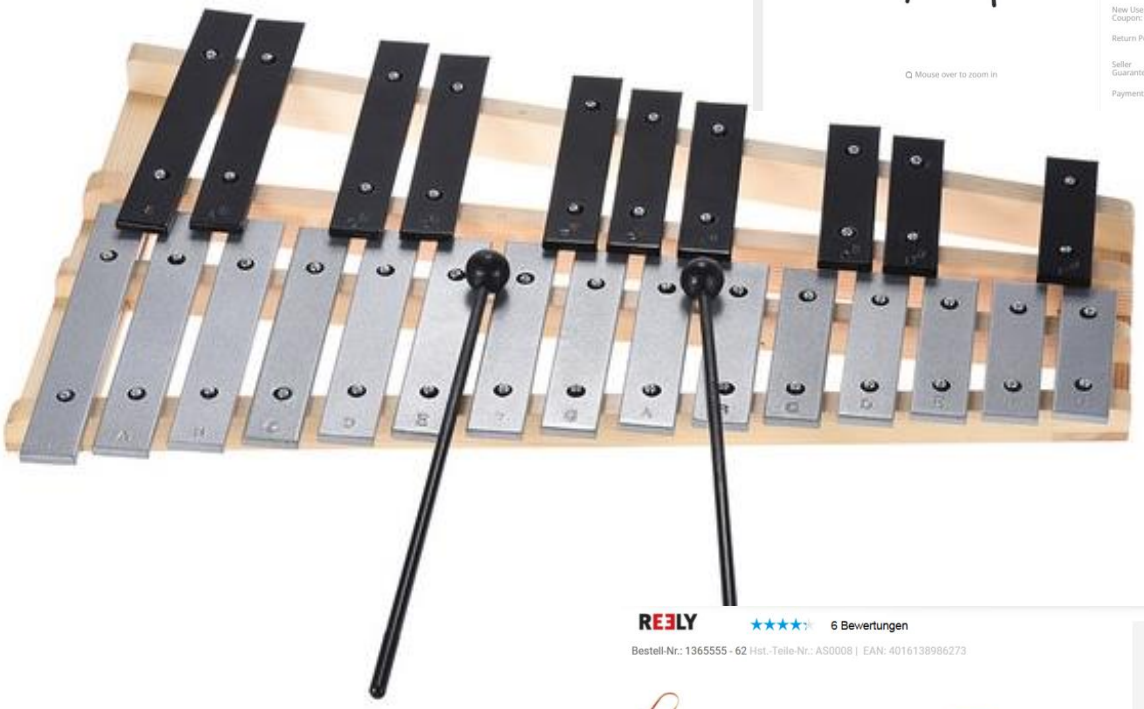


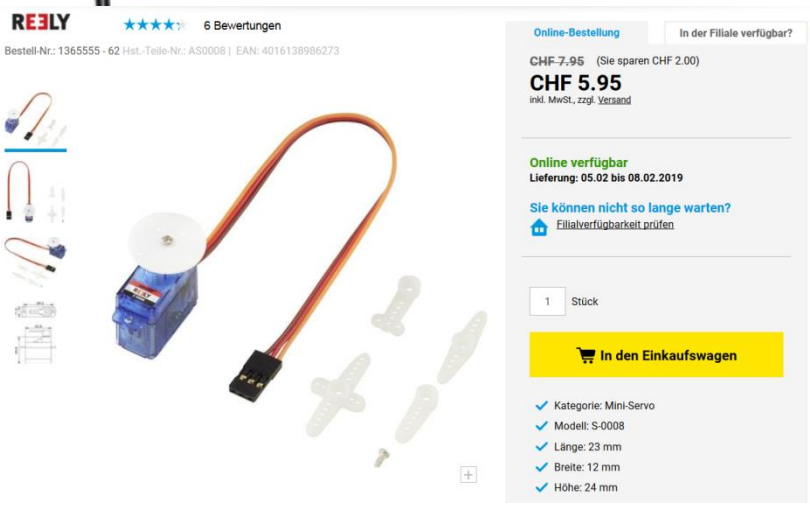


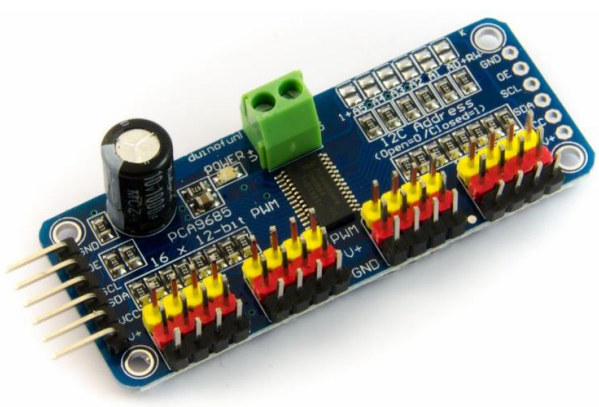


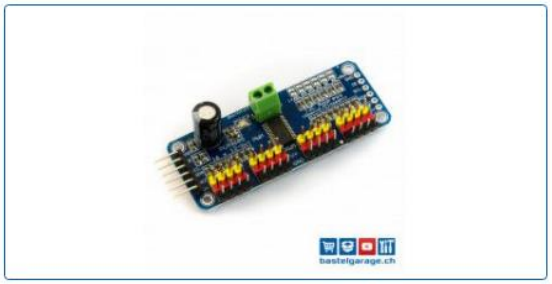
Materialien




16-Kanal PWM / Servo Treiber I2C (PCA9685)

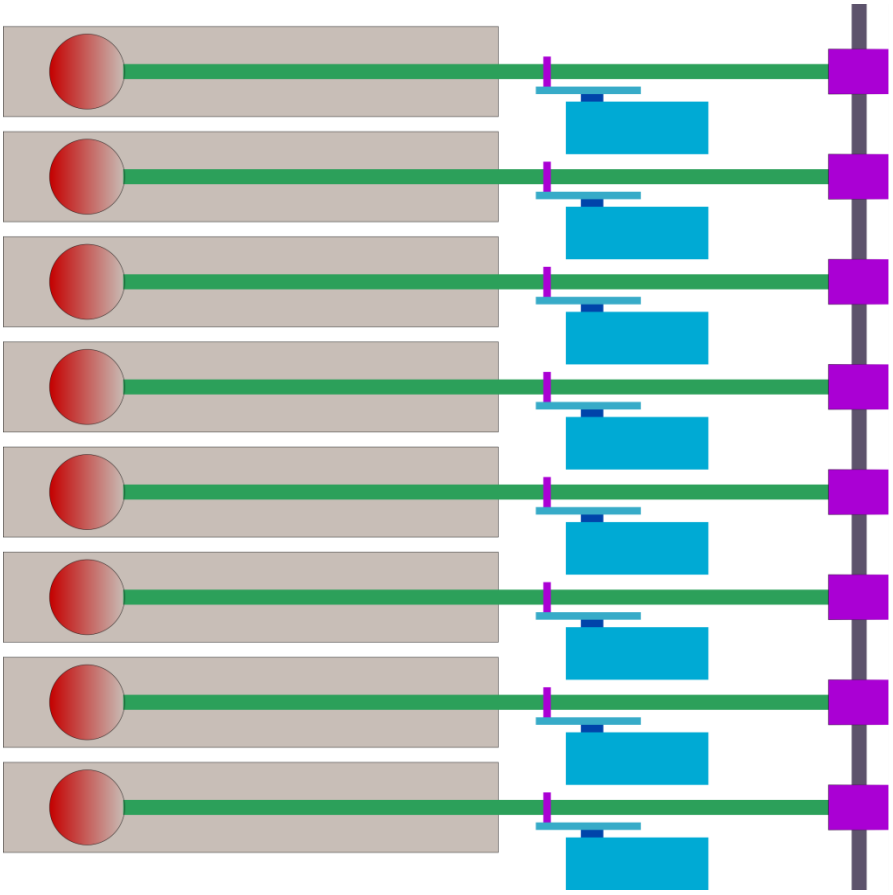
Artikelnr. 420049

Verfügbarkeit Lagernd

11,90CHF

Netto 11,05CHF

Konzeptionelle Entwürfe

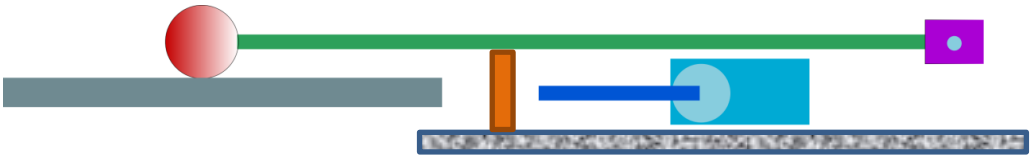


## Der Schlägel fällt im freien Fall

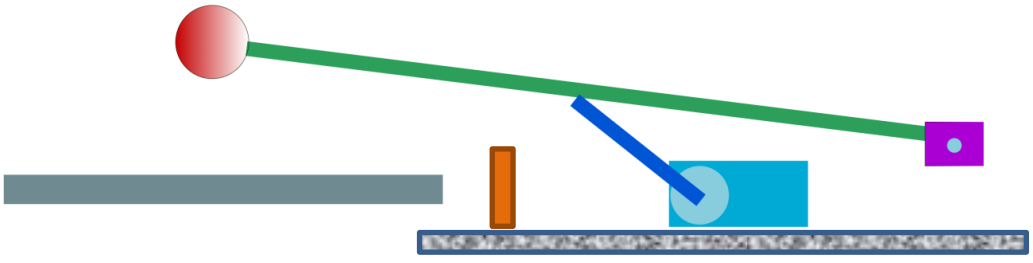
Ein erster Entwurf basiert auf der Grundidee des Gloggomobils. Über jeder Metallplatte ruht ein Schlägel auf einer vorgegebenen Höhe. Die Schlägel sind am Fusspunkt mit einer langen Stange verbunden und in vertikaler Richtung frei beweglich. Die Neigung des Schlägels in der Ruhestellung wird vom Servo-Motor bestimmt.
Soll ein Ton erklingen macht der entsprechende Servo eine schnelle Drehung von etwa 45 Grad. Danach fällt die Kugel – beschleunigt durch ihr Eigengewicht – auf die Metallplatte. Ob der Schlägel unmittelbar danach wieder in die Ruhestellung zurück-geführt werden muss (Vibrationen) soll der praktische Versuch zeigen.


## Der Schlägel wird vollständig vom Servo-Motor kontrolliert

Denkbar wäre auch, dass der Schlägel fest mit der Achse des Servo-Motors verbunden ist und der Servo die Bewegung des Spielers nachahmt. Damit könnte der Klangrhythmus und die Klangstärke präziser programmiert werden.
Auch hier muss der praktische Versuch zeigen, was technisch möglich und mit vertretbarem Aufwand machbar ist.

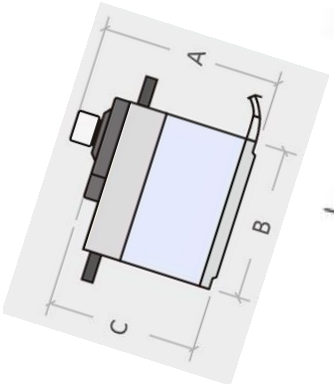
Die definitiven Dimensionen des Schlägels werden im praktischen Versuch ermittelt. Sowohl die Länge des Arms wie auch das Gewicht der Kugel sind massgeblich für die Erzeugung der Tonschwingung.

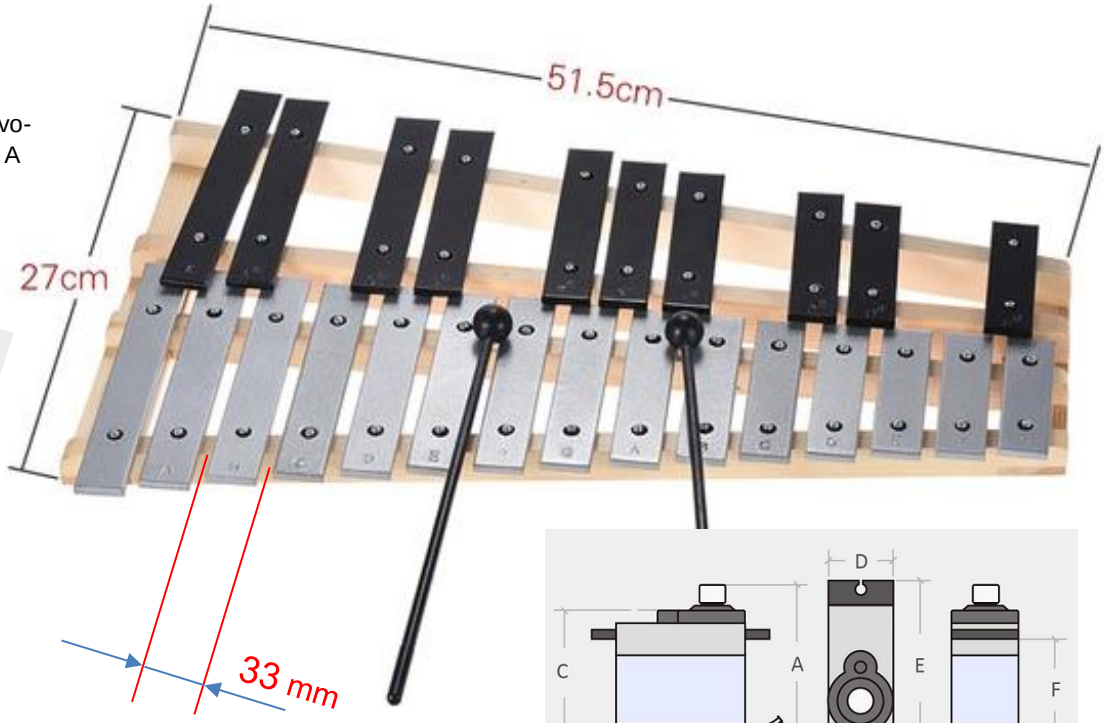


Möglicherweise muss mit einer Abfederung verhindert werden, dass der Schlägel aufspringt und ein zweites Mal auf die Metallplatte fällt.

## Kritische Grössen

Die besondere Lage der Servo-Motoren erfordert eine Höhe A von weniger als 30 mm.



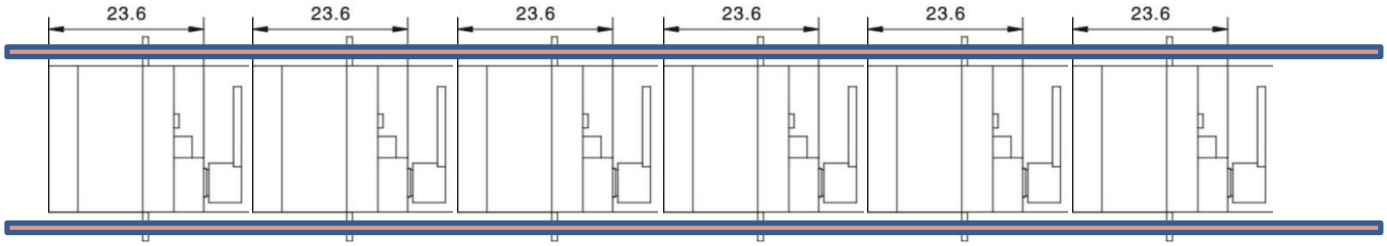







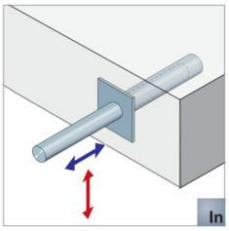
### Prinzip der Befestigung

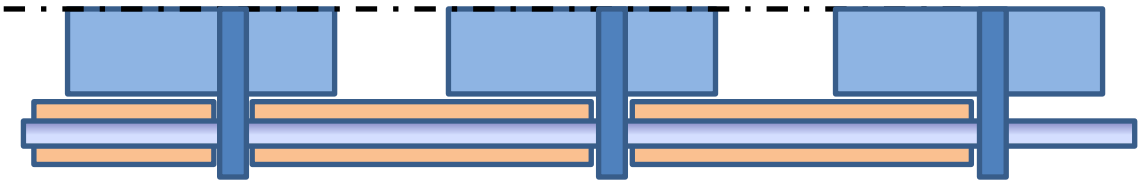
Die Servomotoren werden mit zwei Metallstäben fixiert, die durch die beiden Halterungslaschen geführt werden.




### Ausrichtung und Justierung

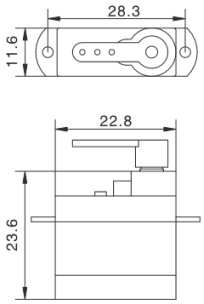
Zwischen den Halterungslaschen werden (Kunstsstoff-) Hülsen eingesetzt, die eine genaue seitliche Ausrichtung garantieren.





### Die technischen Merkmale des Servomotors

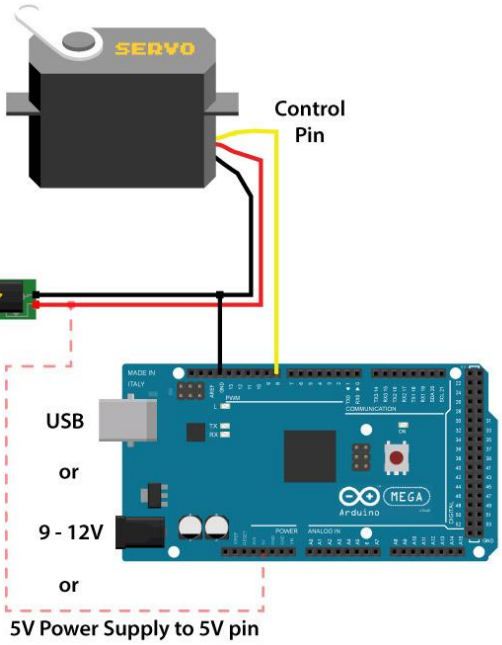
| | |
|---|---|
| Servo-Typ / Servo type | Micro servo |
| Servo-Technologie / Servo technology | Analog / Analogue servo |
| Getriebe-Material / Gear box | Kunststoff / Plastic |
| Stell-Moment / Torque at 4.8 V / 6.0 V | 10 / 11 Ncm |
| Stell-Zeit / Actuation time at 4.8 V / 6.0 V | 0.14sec / 0.12sec (60°) |
| Gewicht / Weight | 8g |
| Stecksystem / Connector system | JR |

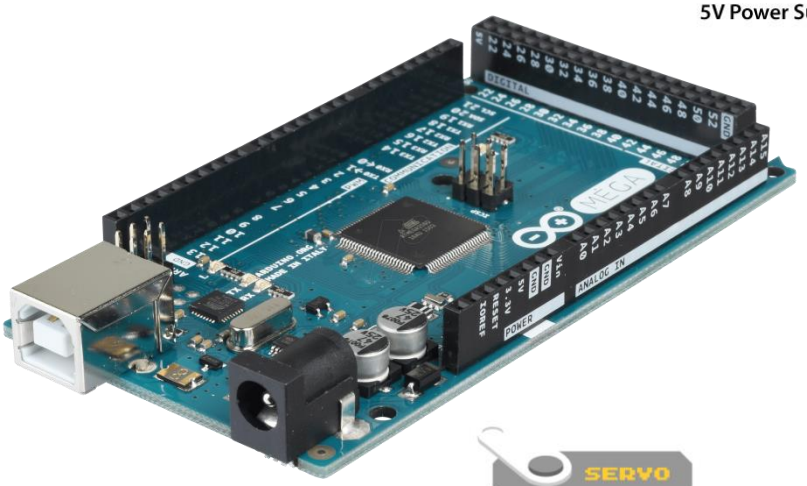


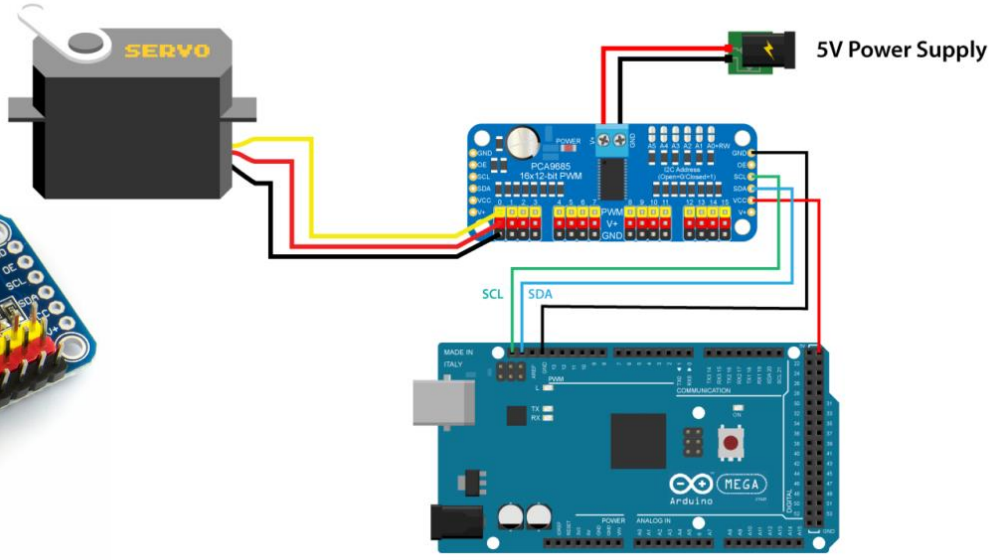

UNIT: MM

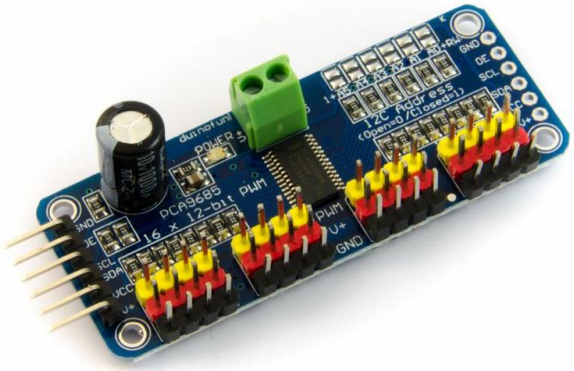
## Die Komponenten der Steuerung

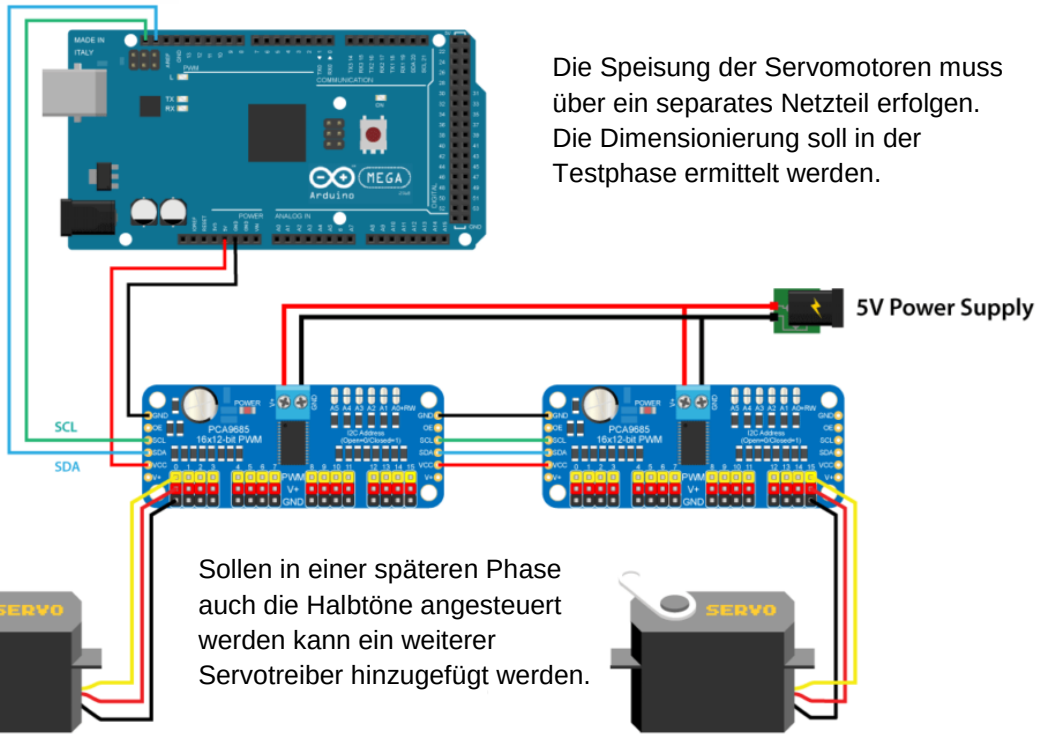
In der Entwicklungsphase soll ein Arduino MEGA zum Einsatz kommen, weil dieser über die in der Testphase erforderlichen Schnittstellen verfügt. Es ist aber durchaus möglich, diesen bei der Realisierung des Projektes durch einen Arduino UNO zu ersetzen.








Die Speisung der Servomotoren muss über ein separates Netzteil erfolgen. Die Dimensionierung soll in der Testphase ermittelt werden.

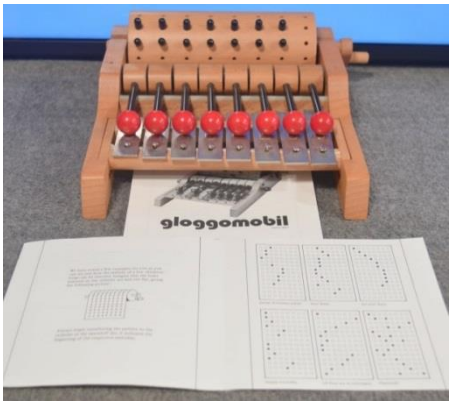
Sollen in einer späteren Phase auch die Halbtöne angesteuert werden kann ein weiterer Servotreiber hinzugefügt werden.

## Skizze eines Programmentwurfs

So etwa wird das Programm zur Ansteuerung der Servo-Motoren aussehen…

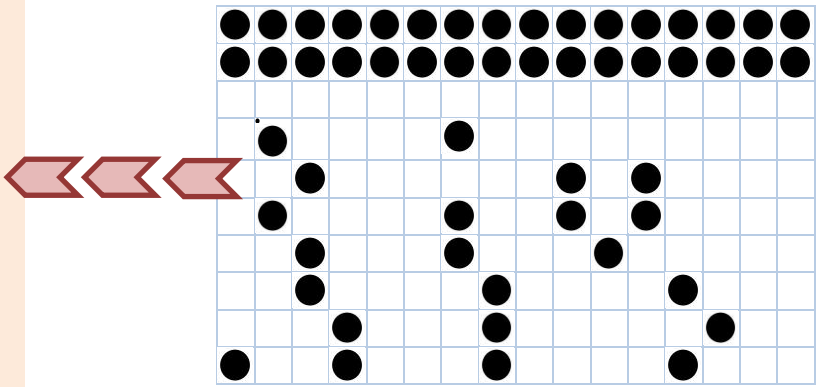
```
/*
  PCA9685 PWM Servo Driver Example
  pca9685-servomotor-demo.ino
  Demonstrates use of 16 channel I2C PWM driver board with 4 servo motors
  Uses Adafruit PWM library
  Uses 4 potentiometers for input

  DroneBot Workshop 2018
  https://dronebotworkshop.com
*/

// Include Wire Library for I2C Communications
#include <Wire.h>

// Include Adafruit PWM Library
#include <Adafruit_PWMServoDriver.h>

#define MIN_PULSE_WIDTH       650
#define MAX_PULSE_WIDTH       2350
#define FREQUENCY             50

Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();

// Define Potentiometer Inputs

int controlA = A0;
int controlB = A1;
int controlC = A2;
int controlD = A3;

// Define Motor Outputs on PCA9685 board

int motorA = 0;
int motorB = 4;
int motorC = 8;
int motorD = 12;

void setup()
{
  pwm.begin();
  pwm.setPWMFreq(FREQUENCY);
}


void moveMotor(int controlIn, int motorOut)
{
  int pulse_wide, pulse_width, potVal;

  // Read values from potentiometer
  potVal = analogRead(controlIn);

  // Convert to pulse width
  pulse_wide = map(potVal, 0, 1023, MIN_PULSE_WIDTH, MAX_PULSE_WIDTH);
  pulse_width = int(float(pulse_wide) / 1000000 * FREQUENCY * 4096);

  //Control Motor
  pwm.setPWM(motorOut, 0, pulse_width);

}

void loop() {

  //Control Motor A
  moveMotor(controlA, motorA);

  //Control Motor B
  moveMotor(controlB, motorB);

  //Control Motor C
  moveMotor(controlC, motorC);

  //Control Motor D
  moveMotor(controlD, motorD);


}
```

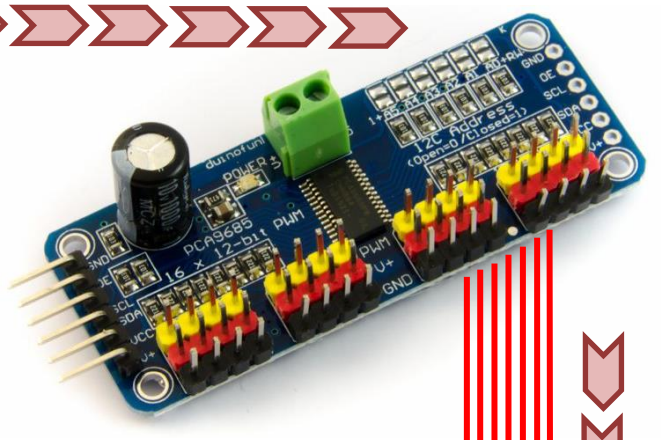
## Prinzip der Steuerlogik

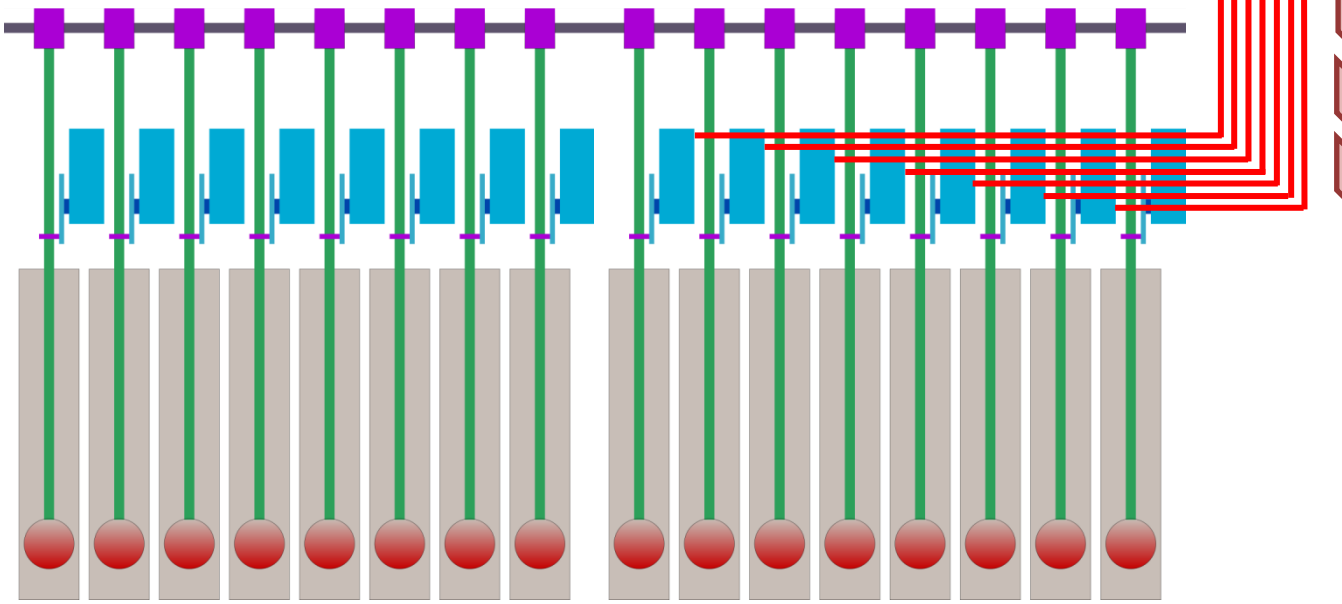



Die Melodie wird aus dem Notenblatt in eine Hilfstabelle übertragen, die für jeden Klangkörper festlegt, zu welchem Zeitpunkt der Schlag ausgeführt werden soll. Dies geschieht von Hand, oder (in einer späteren Phase) mit Hilfe eines Programms.
Diese Noteninformationen werden dem Computer in Form einer Datenmatrix übergeben, wo sie für die Ansteuerung der Servo-Motoren verwendet werden.

```
int matrix[n][16] = {
  {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
  {1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1}
  {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}
  {0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,0}
  {0,0,1,0,0,0,0,0,0,1,0,1,0,0,0,0}
  {0,1,0,0,0,0,0,0,0,1,0,1,0,0,0,0}
  {0,0,1,0,0,0,1,0,0,0,1,0,0,0,0,0}
  {0,0,1,0,0,0,0,1,0,0,0,0,1,0,0,0}
  {0,0,0,1,0,0,0,1,0,0,0,0,0,1,0,0}
  {1,0,0,1,0,0,0,1,0,0,0,0,1,0,0,0}
};
```

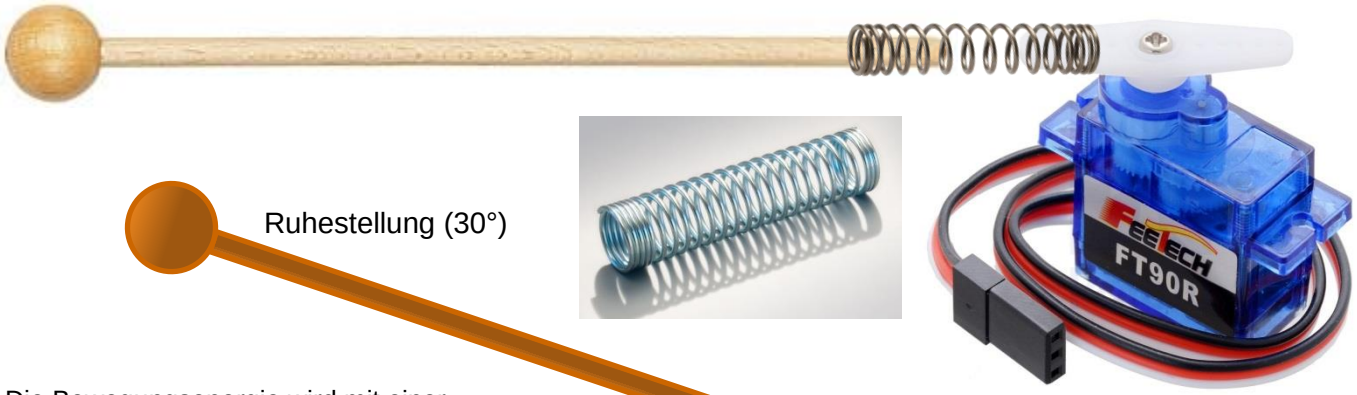


# Befestigung des Schlägels (Prinzip)

Die Drehbewegung des Servo-Armes erfolgt mit der grösstmöglichen Beschleunigung.



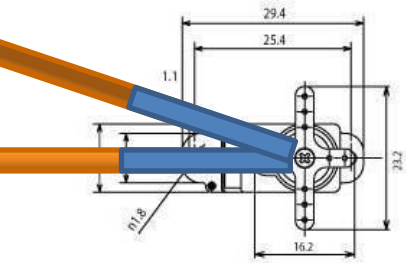





Ruhestellung (30°)

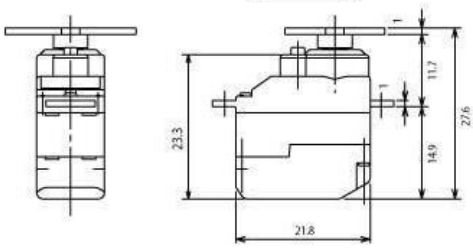
Die Bewegungsenergie wird mit einer Druckfeder verzögert und gespeichert.

Tonerzeugung (0°)

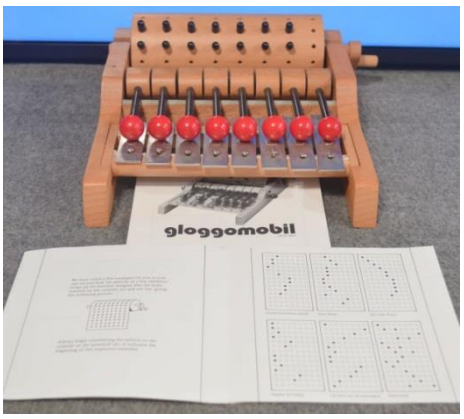


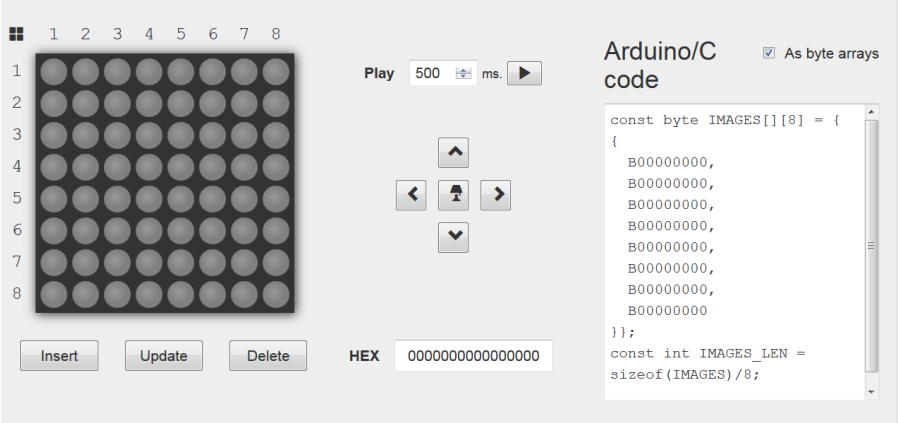

Die Normalsituation eines Schlägels ist die Ruhestellung, in einem 30° Winkel zur horizontalen Grundfläche.

Erkennt die Servosteuerung, dass ein Klangkörper aktiviert werden soll, wird der Servo-Motor des entsprechenden Schlägels von der Ruheposition (30°) direkt (also mit grösstmöglicher Geschwindigkeit) in die Aktionsposition (0°) gedreht. Danach wird Der Servo-Arm in kleinen Schritten (5°..?) wieder in die Ruhestellung zurückgeführt.

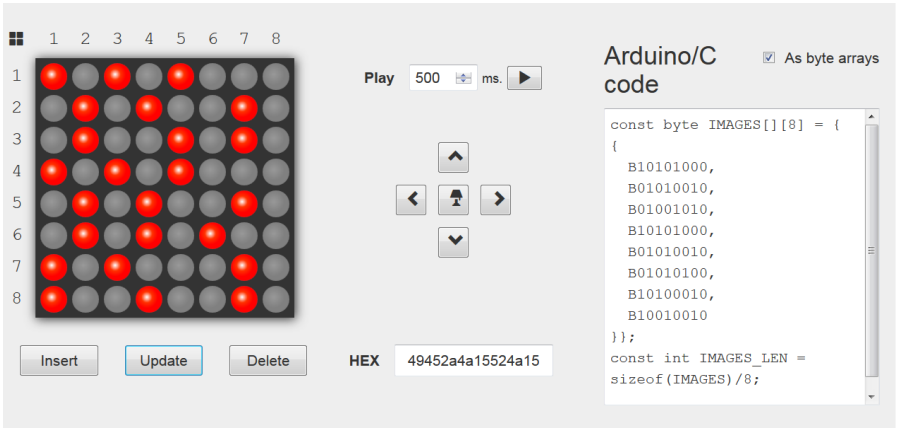


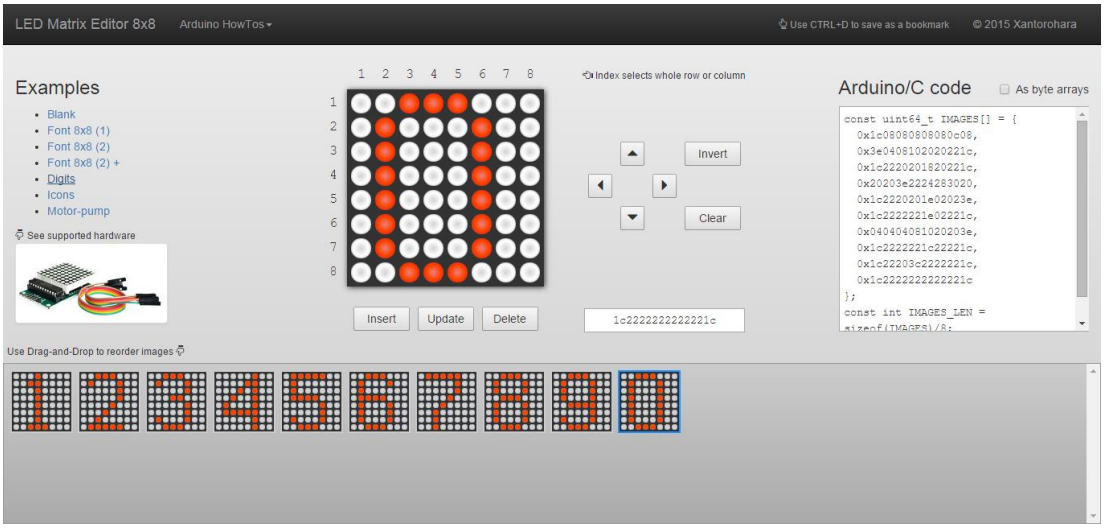

## Hilfsmittel zur Erstellung der Melodie-Vorlagen



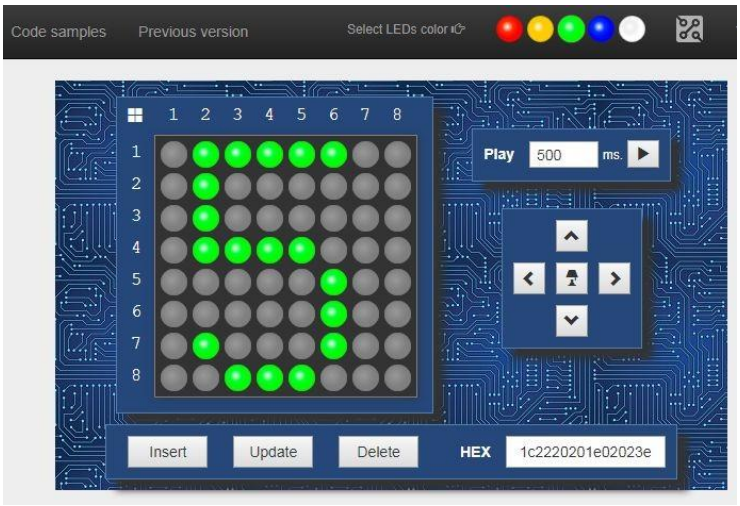

```
// Offsets for rows/cols
const int start_rows = 38;
const int start_cols = 22;

// Amount of rows/cols
const int num_rows = 8;
const int num_cols = 8;

// 1-Dimensional Array, Bits represent LEDs
const char image[num_rows] ={B11111111,\
                             B10000011,\
                             B10000101,\
                             B10001001,\
                             B10010001,\
                             B10100001,\
                             B11000001,\
                             B11111111};

// Bitmask for logical AND
char bitmask = B00000001;

void setup()
{
```




Vielleicht lässt sich auf der Grundlage von Matrix-Editoren ein Werkzeug erstellen, das als eigentliches Noten-Programm verwendet werden kann.

Matrix-Editoren werden bei LED-Displays für Textdarstellungen verwendet, die als sogenannte Lauflichter angezeigt werden. Sie erzeugen einen Code, der direkt in das Arduino-Programm hineinkopiert werden kann.



```
// Offsets for rows/cols
const int start_rows = 38;
const int start_cols = 22;

// Amount of rows/cols
const int num_rows = 8;
const int num_cols = 8;

// 1-Dimensional Array, Bits represent LEDs
const char image[num_rows] ={B11111111,\
                             B10000011,\
                             B10000101,\
                             B10001001,\
                             B10010001,\
                             B10100001,\
                             B11000001,\
                             B11111111};

// Bitmask for logical AND
char bitmask = B00000001;

void setup()
{
    // Set up rows
    for(int i = 0; i < num_rows; i++)
    {
        pinMode(start_rows + (2*i), OUTPUT);
        digitalWrite(start_rows + (2*i), LOW);
    }

    // Set up cols
    for(int i = 0; i < num_cols; i++)
    {
        pinMode(start_cols + (2*i), OUTPUT);
        digitalWrite(start_cols + (2*i), LOW);
    }
}

void loop()
{
    // Repeat displaying a single image
    for(int i = 0; i < 500; i++)
    {
        // Iterate rows
        for(int r = 0; r < num_rows; r++)
        {
            // Switch on a row
            digitalWrite(start_rows + 2*r, HIGH);

            // Iterate Cols
            for(int c = 0; c < num_cols; c++)           {               // Check which co
                    digitalWrite(start_cols + 2*c, HIGH);
                else
                    digitalWrite(start_cols + 2*c, LOW);
            }

            // Iterate Cols
            for(int c = 0; c < num_cols; c++)
            {
                // Switch off cols again
                digitalWrite(start_cols + 2*c, LOW);
            }

            // Switch off row again
            digitalWrite(start_rows + 2*r, LOW);
        }
    }
}
```