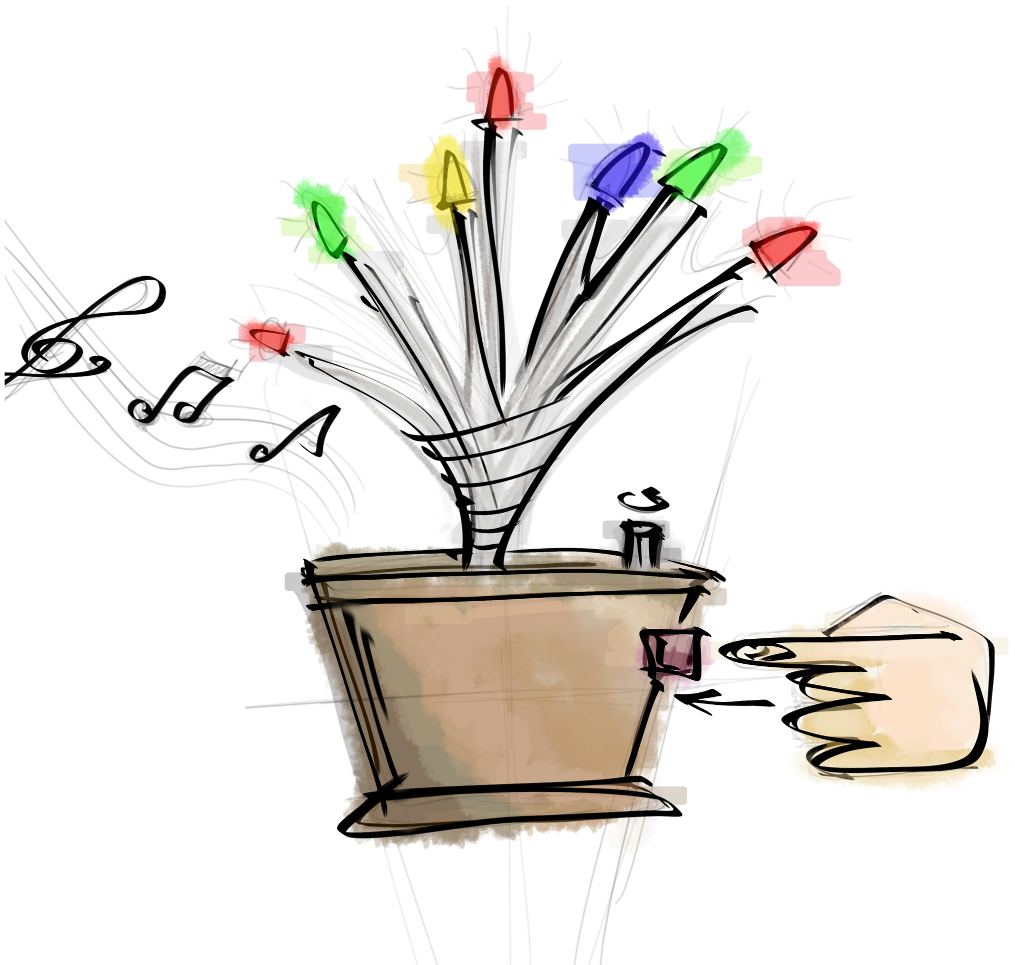
Musical Tree

Mureşan Bianca-Maria

January 2019



Description

The project is about making an electronic tree that comes out of a pot with 20 branches and at the end of every branch there is a color LED (four colors in total). On the pot there is a button and for each press the current melody will change (three melodies in total). Also there will be a potentiometer for volume adjustment.

Input and output data

Input data:

- button state (Low/High).

Output data:

- the three melodies (the frequencies on buzzer);
- the light on the branches (the voltage on the LEDs).

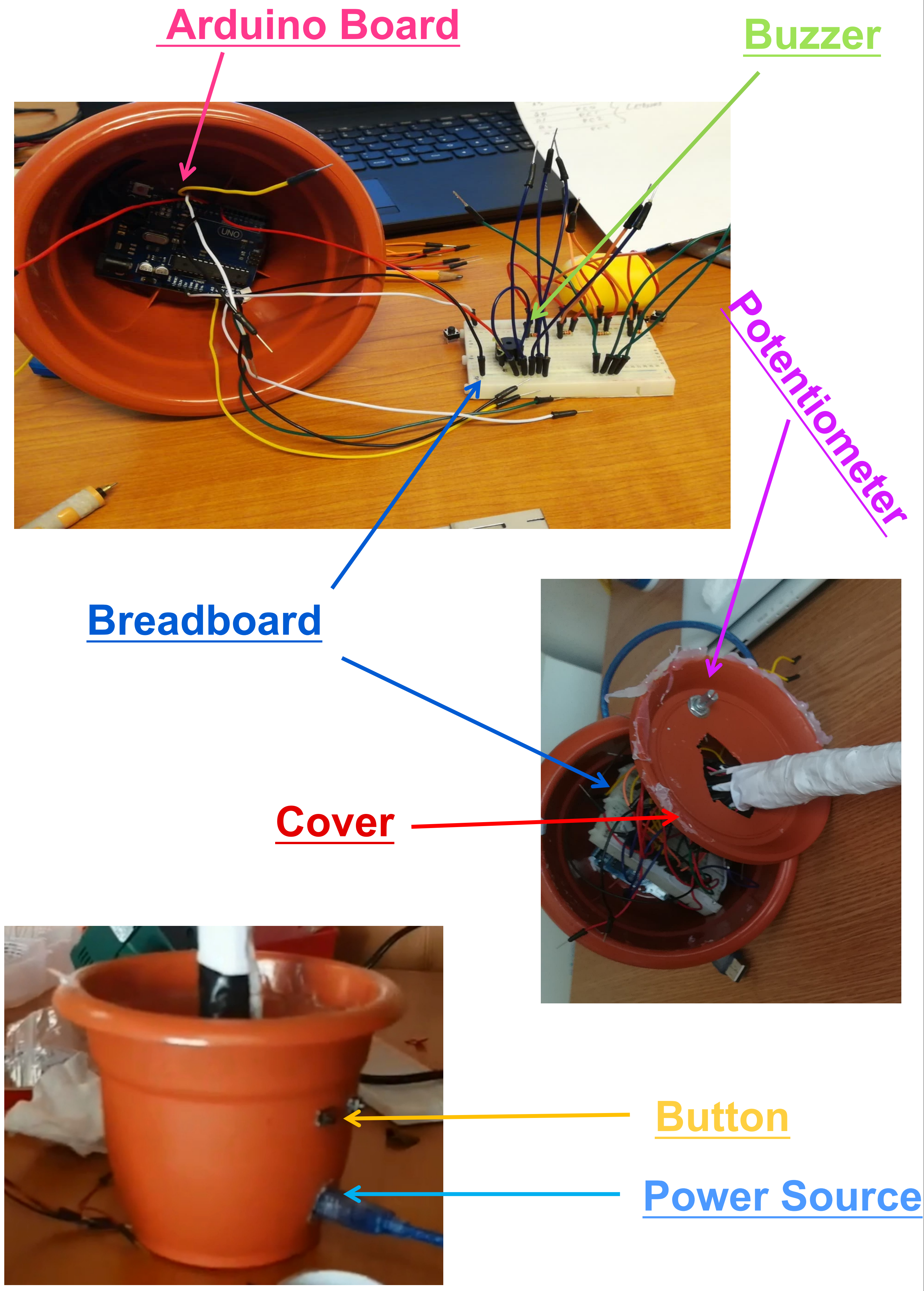
Physical Design

The components:

- an Arduino board (ATmega328P);
- a buzzer and a potentiometer;
- a button;
- 5 LEDs of 4 colors each (red, blue, green and yellow);
- a breadboard, resistors and wires.

Pot Design

To make the tree according to the description, the Arduino board is glued on the bottom of the pot. On the pot’s side there are two holes, one for the power source and one to connect the button. The breadboard is placed over the Arduino. The buzzer is connected to the breadboard. Over it, there is a protective cover for the circuits. Also, the potentiometer is placed on the cover.


Arduino Board
Buzzer
Potentiometer
Breadboard
Cover
Button
Power Source

Buzzer Design

The melodies’ notes are played with the help of the buzzer and the potentiometer. The buzzer plays the note frequencies and the potentiometer adjusts the volume.

The potentiometer’s pins are connected this way:

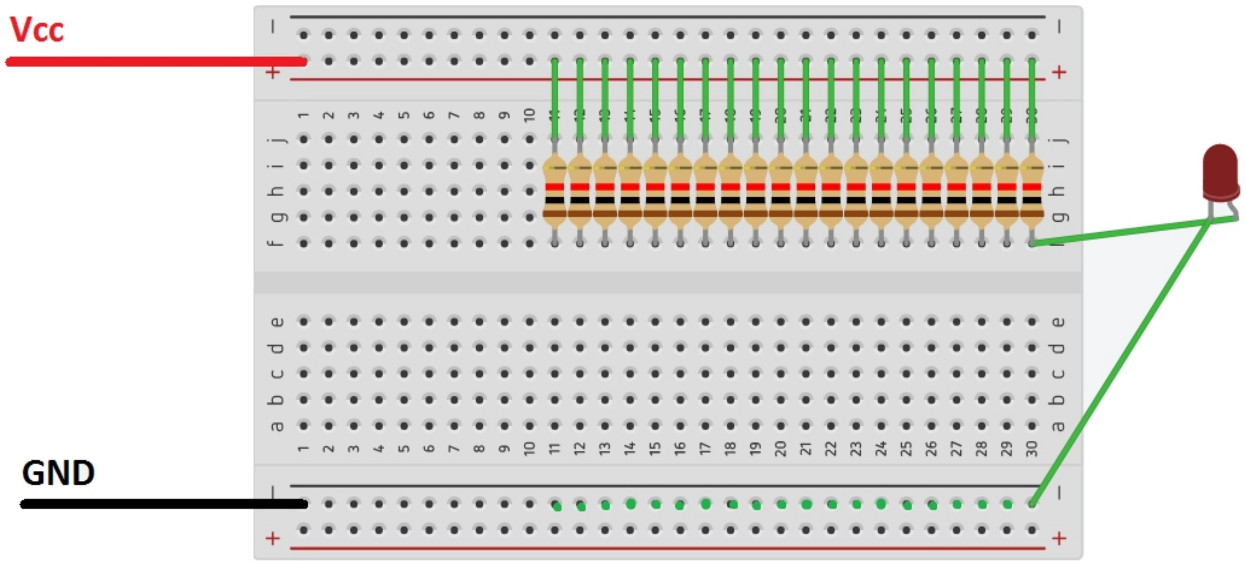
- one to the GND;
- one to the buzzer;
- one to the PB1 pin of the Arduino board.

Button Design

The button is placed on the pot’s side, above the power source hole. The pot is drilled in four points in which the button's pins are introduced. Two of these pins are connected at GND and Vcc on the breadboard and the third pin is connected at the PD2 on the Arduino board to verify its state (Low/High).

LEDs Design

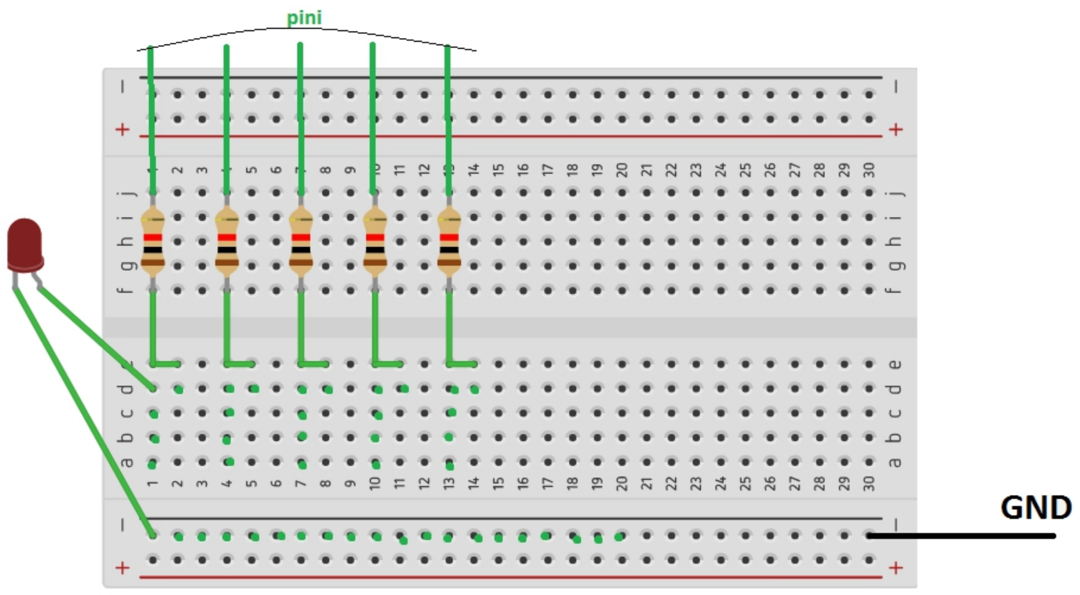
<u>First version</u>: LEDs and resistors in parallel, connected at Vcc



We assume that each resistor is connected to a LED, like the last resistor from the picture above. The voltage from resistor-LED assembly in series is equal with Vcc (5V) and the voltage on the LEDs varies depending on the resistors value.

In this version, the LEDs are always ON as long as the circuit is powered. We can't control them as we wish.

Second version: LEDs and resistors in parallel, connected at a control pin



We assume that we have four different control signal for four pins on the Arduino board to control five resistor-LED assembly in series, like in the picture above. Each of these four circuits of five resistor-LED assembly in series represents a color(red, green, yellow and blue). The voltage of the resistor-LED assembly in series is equal with the control signal voltage when is ON (5V) and the LEDs voltage varies depending on resistor values.

This version allow us to control each color set, but we have too little current on a command pin (max 40mA).

This current is then split on the five resistor-LED assembly in series, so we have maximum 8mA on a branch, but we need minimum 10mA to turn on a LED..

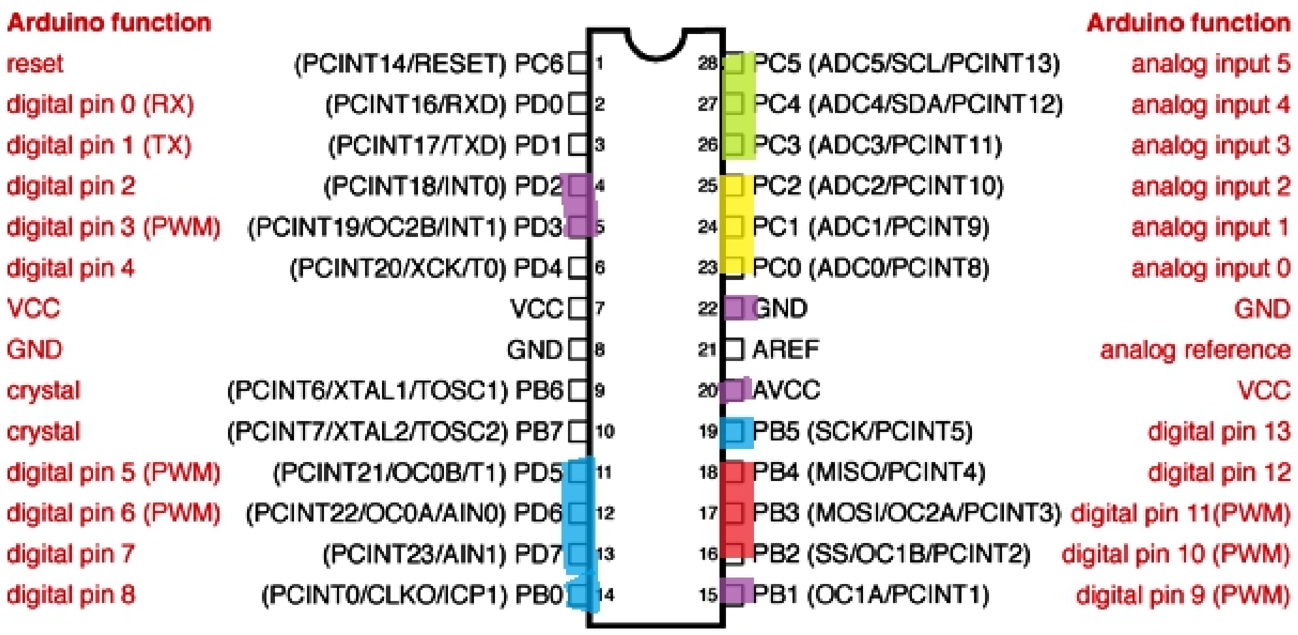
Third version: All in series

This version assumes that we have three command pins for three branches: two LEDs and one resistor in series on two branches and one LED with one resistor on the third branch for each color, except the blue LEDs. The blue LEDs consume too much power and we can't connect them in series if we want them to shine at their full potential. In conclusion, we need five command pins for five blue LED-resistor assembly in series. To find out the value for each resistors I used the Ohm's law and searched for the voltage for turning on each LED depending on its color.

Therefore, we need three pins for red, yellow, green sets and five pins for blue sets, in total fourteen pins to control the LEDs outputs.

That being said, this version will remain the final version.

Pin mapping

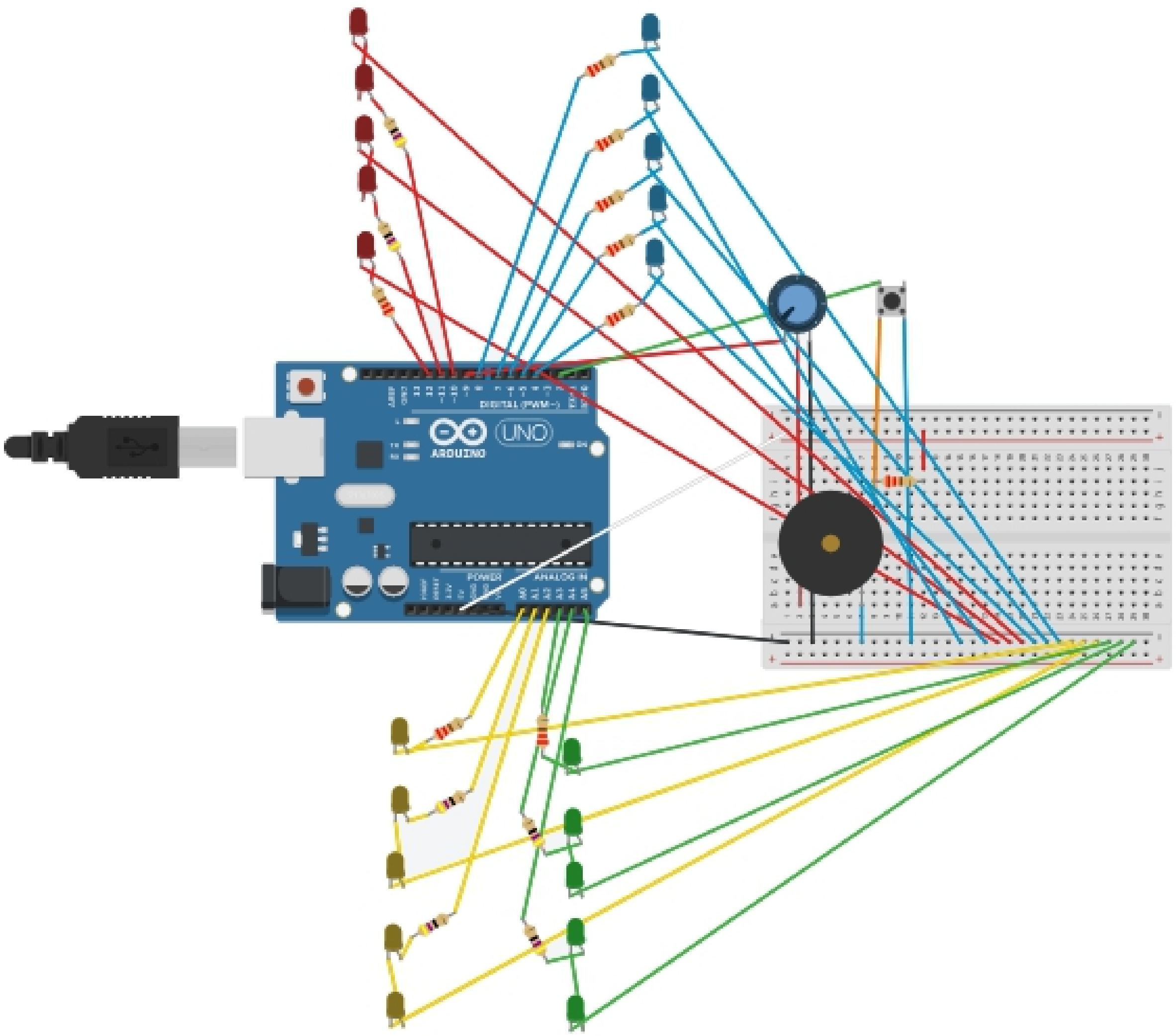


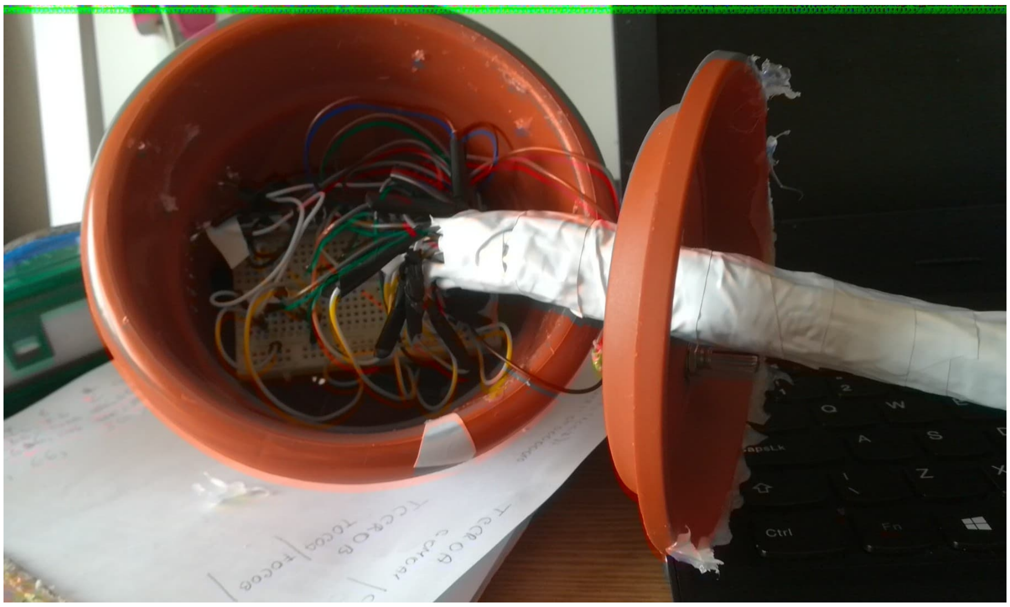
Digital Pins 11,12 & 13 are used by the ICSP header for MOSI, MISO, SCK connections (Atmega168 pins 17,18 & 19). Avoid low-impedance loads on these pins when using the ICSP header.

The blue, red, yellow and green colored pins are the pins chosen for LEDs outputs and the purple ones are chosen for:

- PB1 - the buzzer command signal. For this specific command pin we need to create PWM signal as output to play the frequency and the tempo of the melodies' notes. So we need a pin with OCR1A or OCR2B (from PD3 pin).
- PD2 - this pin is reading the signal input from the button.

Final Physical Design





The code

Timer0 initialization:

```
void timer0_init() {
SREG = 1<<7;  // Global Interrupt Enable

                          //normal port operation,
TCCR0A = 0b00000010;      // OC0A si OC0B disconnect,
TCCR0B = 0b00000011;      //CTC with OCRA,
                          //64 prescaler

TCNT0 = 0;  //timer0 counting register
OCR0A = 250;  //16.000.000/64=250.000/250=1.000Hz
              //=>T=1ms
TIMSK0 |= 0b00000010;     //Enables internal
}                         //output comparison interrupts
                          // for OCR0A
```

Interrupts initialization:

```
void interrupt_init(){
EIMSK=0b00000011;  //acvtivate the external interrupts

EIRCA=0b00001010;  //positive edge INT0 and INT1
}                  //(PD2)
```

Buzzer initialization:

```
void buzzer_init(){
  DDRB |= 0b00000010; //PB1 is set as an output
                      //for the buzzer's PWM signal

  TCCR1A =  0b01000000; // the setings for the timer as
  TCCR1B =  0b00001010; //CTC, clk/8 prescaler,
  TCCR1C =  0b00000000; //toogle OC1A compare
                        //match
 }
```

Button initialization:

```
void button_int(){
   DDRD &=~ 0b00000100; //PD2 - switch the melody
}
```

LEDs initialization:

```
void  LEDs_init(){
DDRC |= 0b00000111;  //yellow output
DDRC |= 0b00111000;  //green output
DDRB |= 0b00111101;  //red output
DDRD |= 0b11100000;  //iblue output
}
```

Global Variables:

1. All the notes frequencies that we use (ex: float C4 = 261.63;);
2. The melodies' tempo (int sec;);
3. The arrays which contain all the notes in order and the arrays which contain timings for each note depending on the tempo;
4. The milliseconds(long int ms = 0;), the counter for looping the melodies arrays (int i =0;) and the TOP variable used for frequency(long int TOP;);
5. The counter for switching the melodies arrays (int contor= -1;).

Main:

```
int main(){
  timer0_init();
  buzzer_init();
  button_int();
  LEDs_init();
  interrupt_init();

  while(1){
  }
}
```

Interrupts:

```
ISR(INT0_vect){
  contor++;
  i=0;
  ms=0;
  if(contor==3)
      contor=0;
} //It's activating at
  button press (PD2)
  and the program is
  stopping and running
  the interrupt.
```

PWM signal:

To generate an output signal in CTC mode, the OC0 pin has to be set as digital output. The output mode for OC0 can be set to toggle every time it’s finding a equality between TCNT0 and OCR0. The signal frequency is defined by the following equation:

$$f_{OC0} = \frac{f_{clk\_I/O}}{2 \cdot N \cdot (1 + OCR0)}$$

where the fclk_I/O is the internal sistem clock frequency, N is the prescaler value(1, 8, 64, 256, 1024) and ORC0 is the register value.

In our case the equation above becomes :

TOP = 1000000/frequency
OCR1A = (TOP+1)/2