

# Pian Electric

**Autor:** [Stefanidis Stefan](#)

## Introducere

Proiectul consta intr-un pian electric cu 13 clape ale caror sunete sunt redade de un buzzer prin apasarea a 13 butoane. La apasarea fiecarui buton, se aprinde un led RGB, fiecare nota muzicala avand asociata cate o culoare. Totodata, exista posibilitatea de a inregistra pana la 50 de note muzicale si de a canta automat notele inregistrate prin apasarea unui singur buton. Se afiseaza la Serial Monitor informatii legate de pianul electric. [Aici](#) se afla proiectul de fata implementat si simulat in Tinkercad®, cu mentiunea precizata la [Hardware Design](#).

## Descriere generala

In imaginea de mai jos se afla schema bloc a proiectului. Se pot observa cele **3 intrari analogice** (cele marcate cu puncte) si cele **3 intrari digitale** (cele marcate cu linie continua). Pe fiecare dintre cele 3 intrari analogice se obtin informatii de la cate 4 butoane care activeaza cate o nota muzicala. A 13-a nota muzicala este activata de un buton al carui input este citit de la o intrare digitala. Input-ul pentru butoanele de recording si play sunt si ele citite de la cate o intrare digitala, asemenea.

Totodata, se pot observa cele **4 iesiri digitale**, 3 dintre ele fiind pentru led-ul RGB, iar ultima pentru buzzer-ul pasiv. In plus, programul afiseaza in permanenta mesaje pe **Serial Monitor** despre instructiunile de utilizare a pianului electric si notele muzicale inregistrate si cantate.



## Hardware Design

Piese necesare acestui proiect sunt:

- Arduino
- Breadboard mare
- Breadboard mic
- Buzzer pasiv
- Led RGB
- 15 Butoane
- 18 Rezistente 220Ω
- Fire de tip jumper

Mai jos este redată schema electrică a proiectului. Aceasta este generată de către Tinkercad®. Singura diferență dintre proiectul de față și schema electrică de mai jos este aceea că pe Tinkercad® s-a folosit un led RGB cu catod comun, pe când în acest proiect s-a folosit un led RGB cu anod comun.



## Software Design

Inițial am implementat și simulat proiectul pe Tinkercad®, după care am folosit Arduino IDE ca mediu de dezvoltare. La schimbarea mediului de dezvoltare, singura modificare făcută a fost inversarea codului culorilor pentru led-ul RGB, din motivele menționate în secțiunea [Hardware Design](#).

### Detalii implementare

Au fost implementate următoarele funcții:

- void **setup()** care setează pinii, vectorul în care sunt păstrate notele muzicale înregistrate și afișează instrucțiunile de utilizare ale pianului electric
- void **setColor**(int redVal, int greenVal, int blueVal) care setează culoarea led-ului RGB; dacă se înregistrează, atunci led-ul luminează intermitent
- const char\* **getNote**(float freq) care denumește nota muzicală specifică unei frecvențe
- void **playNote**(float freq) care face buzzer-ul să cânte o notă muzicală, o afișează la Serial Monitor și o înregistrează dacă este cazul; dacă s-a depășit numărul maxim de note muzicale ce pot fi înregistrate (50) se afișează un mesaj de eroare la Serial Monitor
- int **debounceAnalogButton**(int buttonState, int buttonPin) care face debouncing pentru butoanele corespunzătoare intrărilor analogice
- void **readAnalogButtons**(int analogButtonsPin) care citește o intrare analogică și determină ce buton s-a apăsător; fiecare dintre cele 4 butoane specifice unei intrări analogice are câte un prag de tensiune în funcție de care se determină de către aplicație ce buton s-a apăsător
- boolean **debounceDigitalButton**(boolean buttonState, int buttonPin) care face debouncing pentru butoanele corespunzătoare intrărilor digitale
- void **readDigitalButtonLastNote**() care citește intrarea digitală corespunzătoare ultimei note muzicale și o înregistrează dacă este cazul
- void **readDigitalButtonRec**() care citește intrarea digitală corespunzătoare butonului de recording; funcția porneste și închide înregistrarea și afișează notele muzicale înregistrate dacă acestea există
- void **readDigitalButtonPlay**() care citește intrarea digitală corespunzătoare butonului de play; funcția cântă notele înregistrate și le afișează la Serial Monitor, iar dacă nu există note muzicale înregistrate, se afișează un mesaj de eroare
- void **loop**() care execută la nesfârșit funcțiile de mai sus

**La fiecare pornire a înregistrării se golește vectorul în care sunt păstrate notele muzicale înregistrate. Butonul de play poate fi apăsător oricând în timpul înregistrării, caz în care vor**

fi cantate notele muzicale inregistrate pana in momentul respectiv, si dupa inregistrare, caz in care vor fi cantate toate notele muzicale inregistrate.

## Cod Sursa

```
/* INPUT PINS CONSTANTS */

const int analogFirstButtonsPin = A0;
const int analogSecondButtonsPin = A1;
const int analogLastButtonsPin = A2;
const int lastNotePin = 2;
const int recordPin = 3;
const int playPin = 4;

/* OUTPUT PINS CONSTANTS */

const int redPin = 9;
const int greenPin = 10;
const int bluePin = 11;
const int pianoPin = 5;

/* MUSICAL NOTES CONSTANTS */

const int MAX_NOTES_SIZE = 50;
const int noteDuration = 250;
const float C = 261.6;
const float C_sharp = 277.2;
const float D = 293.7;
const float D_sharp = 311.1;
const float E = 329.6;
const float F = 349.2;
const float F_sharp = 370;
const float G = 392;
const float G_sharp = 415.3;
const float A = 440;
const float A_sharp = 466.2;
const float B = 493.9;
const float C2 = 523.2;

/* BUTTONS CONSTANTS */

const int analogValueFirstThreshold = 100;
const int analogValueSecondThreshold = 200;
const int analogValueThirdThreshold = 300;
const int analogValueFourthThreshold = 500;
const int analogPressedThreshold = 2;
const int debounceDuration = 50;

/* ANALOG BUTTONS VARIABLES */
```

```
int analogButtonState = 0;
boolean analogButtonPressed = false;
int pressedTimes = 0;

/* DIGITAL BUTTONS VARIABLES */

boolean digitalButtonState = LOW;
boolean digitalNoteButtonPressed = false;
boolean digitalRecButtonPressed = false;
boolean digitalPlayButtonPressed = false;

/* RECORDING VARIABLES */

boolean record = false;
float recNotes[50];
int recIndex = 0;

void setup()
{
  Serial.begin(9600);
  pinMode(lastNotePin, INPUT_PULLUP);
  pinMode(recordPin, INPUT_PULLUP);
  pinMode(playPin, INPUT_PULLUP);
  pinMode(pianoPin, OUTPUT);
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  for (int i = 0; i < MAX_NOTES_SIZE; i++) {
    recNotes[i] = 0;
  }
  Serial.println("Press the piano keys to play. To start recording, press
the red button.");
  Serial.println("To end recording, press the red button again. If you want
to listen to");
  Serial.println("the recording, press the green button. You can also press
the green button");
  Serial.println("even while recording if you need to listen to the notes
you have recorded so far.");
  Serial.println();
}

/* function used to change the led color */
void setColor(int redVal, int greenVal, int blueVal)
{
  /* the led used is common anode, so the values are reversed */
  analogWrite(redPin, 255 - redVal);
  analogWrite(greenPin, 255 - greenVal);
  analogWrite(bluePin, 255 - blueVal);
  if (record) {
    /* if record is on, the led should blink */
  }
}
```

```
    delay(noteDuration);
    analogWrite(redPin, 255);
    analogWrite(greenPin, 255);
    analogWrite(bluePin, 255);
    delay(noteDuration);
  }
}

/* function used to write to Serial Monitor */
const char* getNote(float freq) {
  if (freq == C) {
    return "C";
  } else if (freq == C_sharp) {
    return "C#";
  } else if (freq == D) {
    return "D";
  } else if (freq == D_sharp) {
    return "D#";
  } else if (freq == E) {
    return "E";
  } else if (freq == F) {
    return "F";
  } else if (freq == F_sharp) {
    return "F#";
  } else if (freq == G) {
    return "G";
  } else if (freq == G_sharp) {
    return "G#";
  } else if (freq == A) {
    return "A";
  } else if (freq == A_sharp) {
    return "A#";
  } else if (freq == B) {
    return "B";
  } else if (freq == C2) {
    return "C2";
  }
}

/* function used for the analog buttons */
void playNote(float freq)
{
  pressedTimes = 0;
  if (!analogButtonPressed) {
    /* once pressed, the piano key should sing only for noteDuration
    milliseconds */
    analogButtonPressed = true;
    Serial.println(getNote(freq));
    tone(pianoPin, freq, noteDuration);
    delay(noteDuration);
    /* id record is on, save the note */
  }
}
```

```
    if (record && recIndex < MAX_NOTES_SIZE) {
        recNotes[recIndex] = freq;
        recIndex++;
    }
    if (recIndex == MAX_NOTES_SIZE) {
        Serial.println("Maximum number of notes has been reached! No longer
recording.\n");
    }
}
}

/* function used to debounce the analog buttons */
int debounceAnalogButton(int buttonState, int buttonPin)
{
    int currentState = analogRead(buttonPin);
    if (buttonState != currentState) {
        delay(debounceDuration);
        currentState = analogRead(buttonPin);
    }
    return currentState;
}

/* function used to read data from the analog buttons */
void readAnalogButtons(int analogButtonsPin)
{
    int analogValue = debounceAnalogButton(analogButtonState, analogButtonsPin);
    /* read the analog data and decide where it is coming from */
    if (analogValue > analogValueFirstThreshold && analogValue <=
analogValueSecondThreshold) {
        if (analogButtonsPin == analogFirstButtonsPin) {
            setColor(255, 0, 0);
            playNote(C);
        } else if (analogButtonsPin == analogSecondButtonsPin) {
            setColor(0, 255, 0);
            playNote(E);
        } else {
            setColor(0, 0, 255);
            playNote(G_sharp);
        }
    } else if (analogValue > analogValueSecondThreshold && analogValue <=
analogValueThirdThreshold ) {
        if (analogButtonsPin == analogFirstButtonsPin) {
            setColor(255, 50, 0);
            playNote(C_sharp);
        } else if (analogButtonsPin == analogSecondButtonsPin) {
            setColor(0, 255, 128);
            playNote(F);
        } else {
            setColor(128, 0, 255);
            playNote(A);
        }
    }
}
```

```

    } else if (analogValue > analogValueThirdThreshold && analogValue <=
analogValueFourthThreshold ) {
        if (analogButtonsPin == analogFirstButtonsPin) {
            setColor(255, 255, 0);
            playNote(D);
        } else if (analogButtonsPin == analogSecondButtonsPin) {
            setColor(0, 255, 255);
            playNote(F_sharp);
        } else {
            setColor(255, 0, 255);
            playNote(A_sharp);
        }
    } else if (analogValue > analogValueFourthThreshold ) {
        if (analogButtonsPin == analogFirstButtonsPin) {
            setColor(128, 255, 0);
            playNote(D_sharp);
        } else if (analogButtonsPin == analogSecondButtonsPin) {
            setColor(0, 128, 255);
            playNote(G);
        } else {
            setColor(238, 130, 238);
            playNote(B);
        }
    } else {
        /* if the read data is 0 wait to see if the button is pressed or not */
        pressedTimes++;
        if (pressedTimes > analogPressedThreshold) {
            /* if not, turn down the buzzer */
            analogButtonPressed = false;
            noTone(pianoPin);
        }
    }
}

/* function used to debounce the digital buttons */
boolean debounceDigitalButton(boolean buttonState, int buttonPin)
{
    boolean currentState = digitalRead(buttonPin);
    if (buttonState != currentState) {
        delay(debounceDuration);
        currentState = digitalRead(buttonPin);
    }
    return currentState;
}

/* function used to read data from the digital button for the last piano key */
void readDigitalButtonLastNote()
{
    boolean lastNotePinVal = debounceDigitalButton(digitalButtonState,
lastNotePin);
}

```

```
if (lastNotePinVal == LOW) {
    setColor(255, 0, 0);
    if (!digitalNoteButtonPressed) {
        /* once pressed, the piano key should only sing for noteDuration
milliseconds */
        digitalNoteButtonPressed = true;
        Serial.println("C2");
        tone(pianoPin, C2, noteDuration);
        delay(noteDuration);
        /* if record is on, save the note */
        if (record && recIndex < MAX_NOTES_SIZE) {
            recNotes[recIndex] = C2;
            recIndex++;
        }
        if (recIndex == MAX_NOTES_SIZE) {
            Serial.println("Maximum number of notes has been reached! No longer
recording.\n");
        }
    }
} else {
    digitalNoteButtonPressed = false;
    noTone(pianoPin);
}
}

/* function used to read data from the digital button for recording */
void readDigitalButtonRec()
{
    boolean recordPinVal = debounceDigitalButton(digitalButtonState, recordPin);
    if (recordPinVal == LOW) {
        if (!digitalRecButtonPressed) {
            digitalRecButtonPressed = true;
            record = !record;
            /* if recording is turned on clear the buffer */
            if (record) {
                Serial.println("Currently recording...\n");
                recIndex = 0;
                for (int i = 0; i < MAX_NOTES_SIZE; i++) {
                    recNotes[i] = 0;
                }
            }
            else if (recNotes[0] != 0) {
                /* if recording is turned off and there are notes recorded, print
them to serial monitor */
                Serial.println("Recording has stopped. Here are your notes:");
                for (int i = 0; i < MAX_NOTES_SIZE; i++) {
                    if (recNotes[i] != 0) {
                        Serial.print(getNote(recNotes[i]));
                    }
                    if ((i + 1) < MAX_NOTES_SIZE && recNotes[i + 1] != 0) {
                        Serial.print(", ");
                    }
                }
            }
        }
    }
}
```

```

    }
    Serial.println("\n");
} else {
    /* if recording is turned off and there are no recorded notes */
    Serial.println("Recording has stopped. No notes have been recorded.
\n");
}
}
} else {
    digitalRecButtonPressed = false;
}
}

/* function used to read data from the digital button for playing the
recording */
void readDigitalButtonPlay()
{
    boolean playPinVal = debounceDigitalButton(digitalButtonState, playPin);
    if (playPinVal == LOW) {
        if (!digitalPlayButtonPressed) {
            /* once pressed, the play button should play the recorded notes */
            digitalPlayButtonPressed = true;
            if (recNotes[0] != 0) {
                Serial.println("Playing the following notes:");
                for (int i = 0; i < MAX_NOTES_SIZE && recNotes[i] != 0; i++) {
                    Serial.print(getNote(recNotes[i]));
                    if ((i + 1) < MAX_NOTES_SIZE && recNotes[i + 1] != 0) {
                        Serial.print(", ");
                    }
                    tone(pianoPin, recNotes[i], noteDuration);
                    delay(2 * noteDuration);
                }
                Serial.println("\n");
            } else {
                /* or warn the player there are no notes to be played */
                Serial.println("No notes to play!\n");
            }
        }
    } else {
        digitalPlayButtonPressed = false;
        noTone(pianoPin);
    }
}

void loop()
{
    readAnalogButtons(analogFirstButtonsPin);
    readAnalogButtons(analogSecondButtonsPin);
    readAnalogButtons(analogLastButtonsPin);

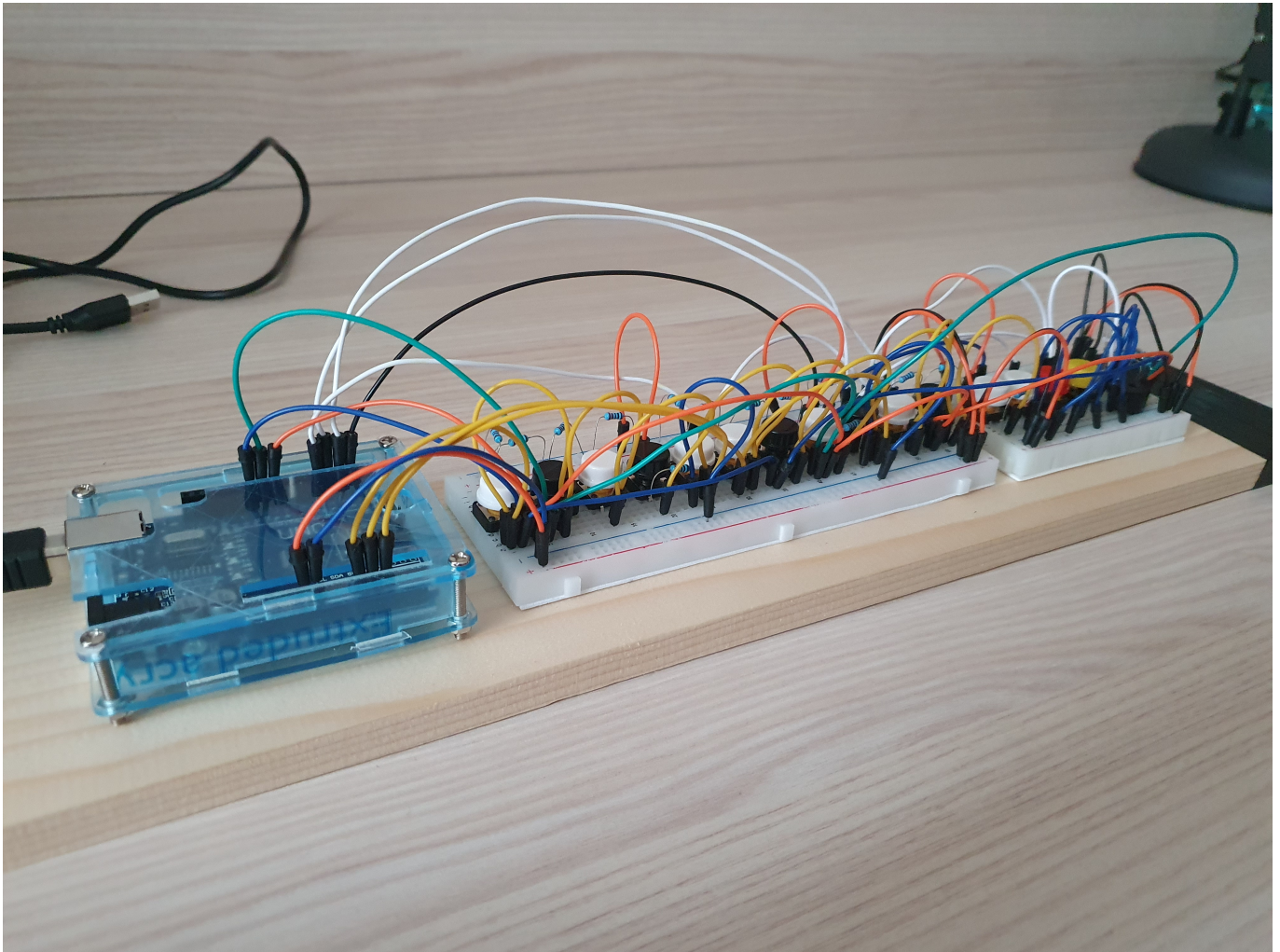
    readDigitalButtonLastNote();
}

```

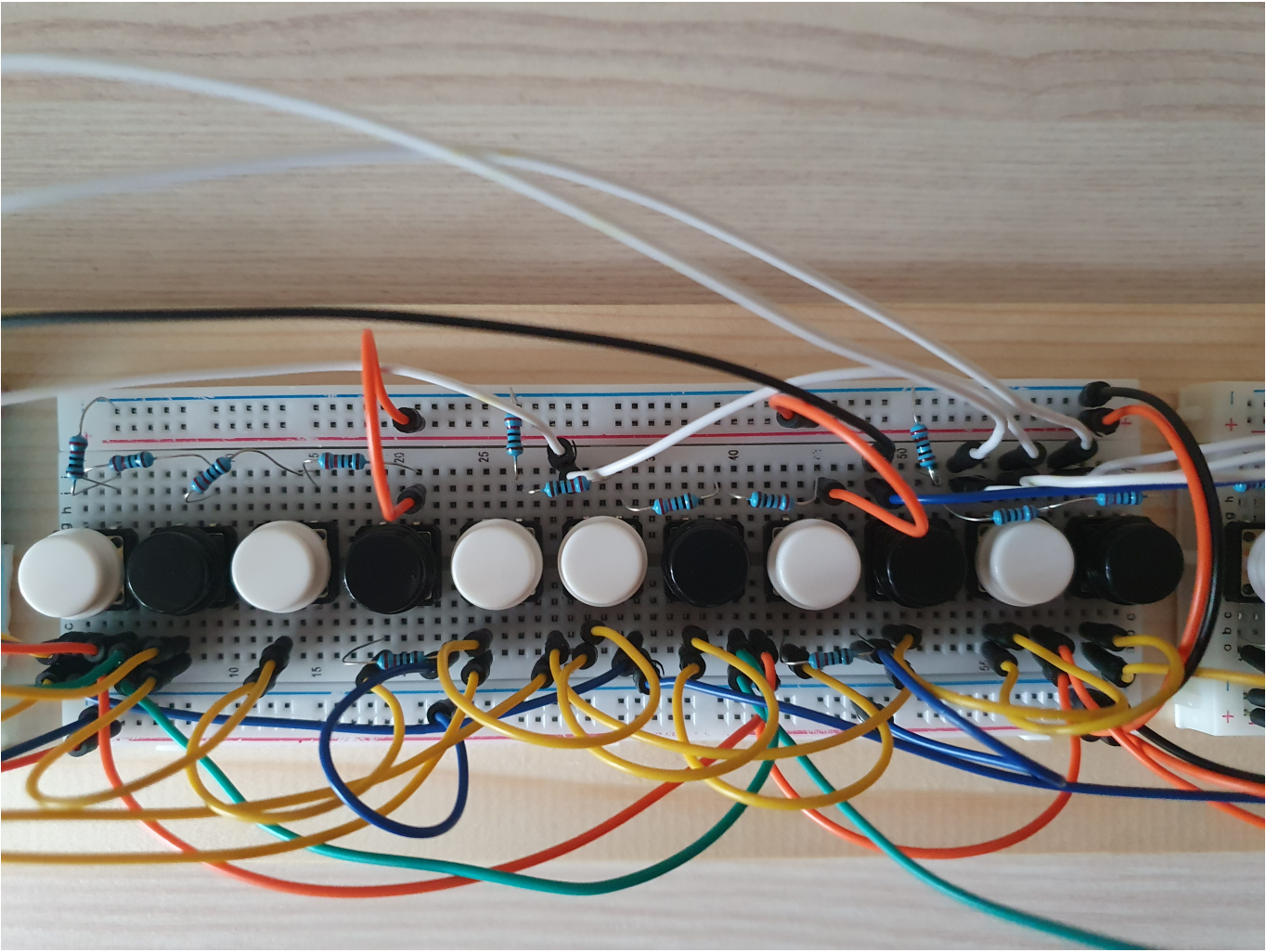
```
readDigitalButtonRec();  
readDigitalButtonPlay();  
}
```

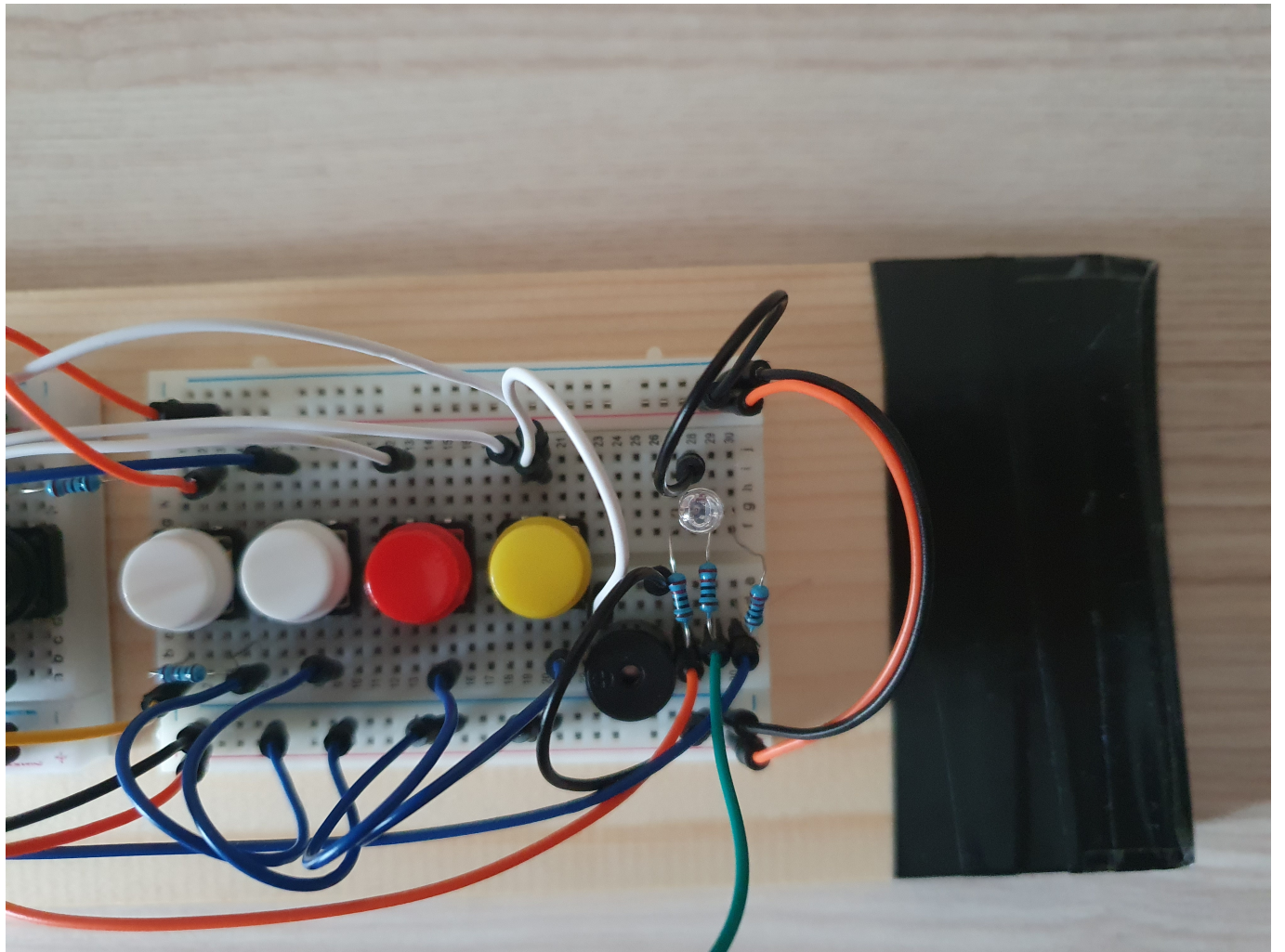
## Rezultate Obținute

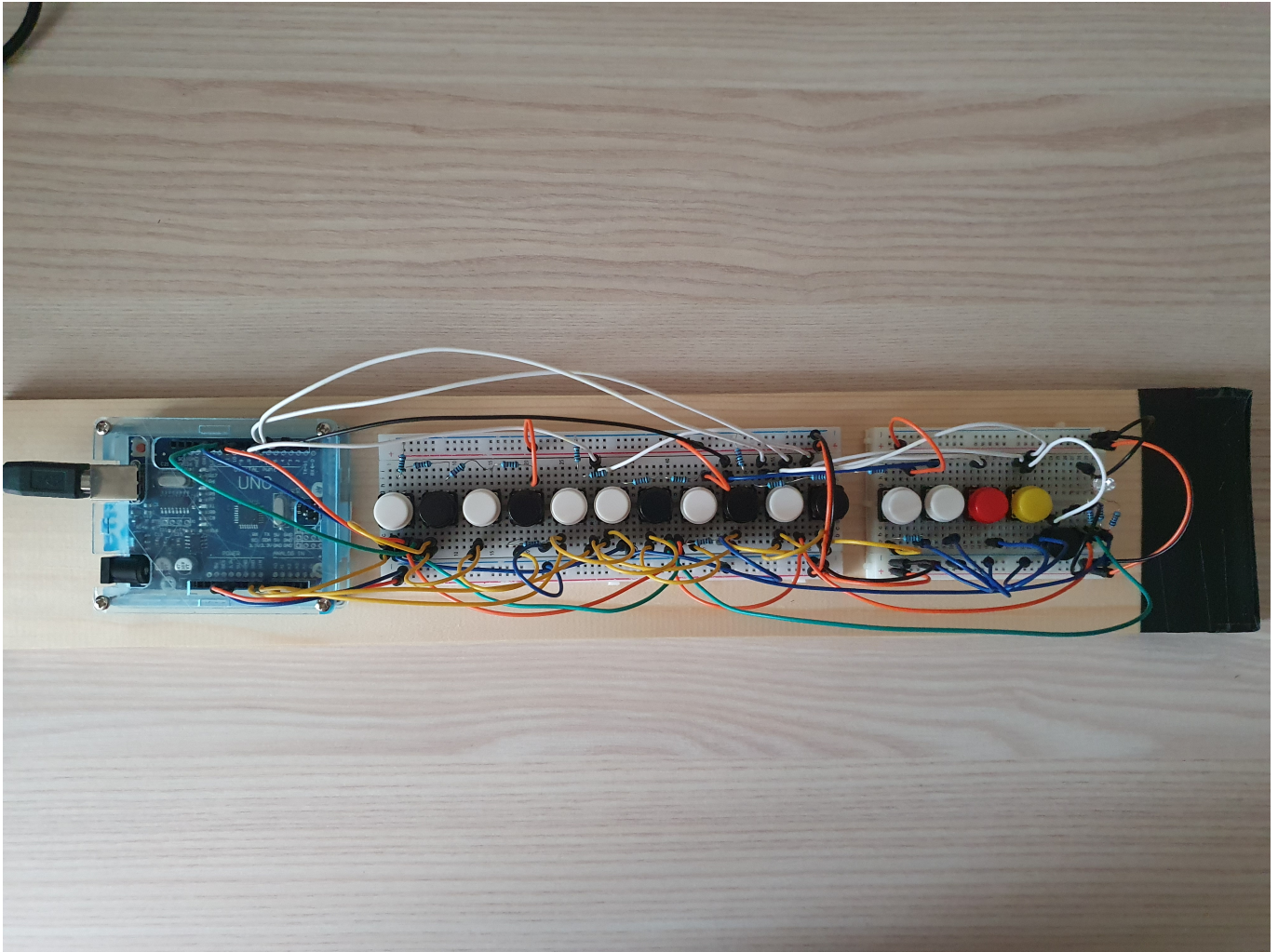
Așa arata proiectul de fata implementat:











## Concluzii

Consider ca proiectul de fata este util pentru mai buna intelegere a modului de utilizare al unui Arduino, in special functiile `analogRead`, `analogWrite` si `digitalRead`. De asemenea, proiectul este folositor pentru studiul modului in care s-au citit date provenite de la mai multe butoane pe o singura intrare analogica si in care s-a facut debouncing pentru toate butoanele. In plus, se poate observa ca pianul este proiectat in asa fel incat sa fie inregistrata o singura apasare de buton chiar daca se tine apasat respectivul buton.

## Download

- [Arhiva cu codul folosit si README](#)
- [Schema electrica a proiectului](#)
- [Pagina de fata](#)

## Jurnal

- **Vineri, 21 aprilie 2022, 6:19 PM:** sectiunile [Titlu](#) si [Introducere](#) terminate; sectiunile [Descriere generala](#) (adaugare schema bloc) si [Hardware Design](#) (adaugare lista de piese) partial terminate
- **Vineri, 21 aprilie 2022, 8:32 PM:** sectiunea [Descriere generala](#) terminata (adaugare explicatii)
- **Luni, 2 mai 2022, 9:26 PM:** sectiunea [Hardware Design](#) terminata (adaugare schema electrica)
- **Luni, 2 mai 2022, 10:44 PM:** sectiunile [Software Design](#) si [Resurse](#) terminate
- **Marti, 3 mai 2022, 12:36 AM:** sectiunea [Download](#) terminata; sectiunea [Rezultate Obtinute](#) partial terminata (adaugare text)
- **Marti, 3 mai 2022, 8:27 AM:** sectiunea [Rezultate Obtinute](#) terminata (adaugare poze)
- **Miercuri, 4 mai 2022, 7:04 AM:** sectiunea [Concluzii](#) terminata

## Resurse

### Resurse Software

- Buzzer pasiv: <https://microdaz.com/what-is-a-passive-buzzer/>
- Debounce butoane: <https://www.electronics-lab.com/project/arduino-button-debounce-tutorial/>

### Resurse Hardware

- Led RGB cu anod comun:  
<https://www.hackster.io/techmirtz/using-common-cathode-and-common-anode-rgb-led-with-arduino-7f3aa9>
- Citire mai multe butoane pe acelasi pin analogic:  
<https://www.instructables.com/How-to-Multiple-Buttons-on-1-Analog-Pin-Arduino-Tu/>

From:  
<http://ocw.cs.pub.ro/courses/> - **CS Open CourseWare**

Permanent link:  
<http://ocw.cs.pub.ro/courses/pm/prj2022/arosca/pian-electric>



Last update: **2022/05/04 08:04**