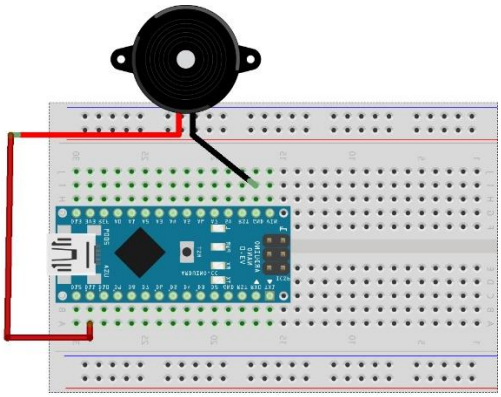
My project is a “Jukebox”. it means I have programmed an Arduino NANO 33 IoT that drives a speaker that plays buzzer tones at a certain rhythm at a certain tempo so that you create a song, musical version based on the song. You can choose which track to play in the Blynk app.

Below you will find the connection diagram of the NANO with the speaker.



This project currently features 13 tracks. Each track has a button on Blynk that you can press to play it. You can press several in succession to create a queue.

Below is the code that goes into the Arduino NANO 33 IoT.

```
// The information to conect to Blynk
#define BLYNK_TEMPLATE_ID          "TMPLISiHrdYF"
#define BLYNK_DEVICE_NAME          "Quickstart Device"
#define BLYNK_AUTH_TOKEN           "3ngKr3laoFwAccfrT4GhAMzGFsUMnTf_"
#define BLYNK_PRINT Serial

char auth[] = BLYNK_AUTH_TOKEN;

// The information to conect to WIFI
char ssid[] = "MSI";
char pass[] = "123456789";

//------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------


// Setting the value of every song on 0
int Hp = 0;
int zel = 0;
int nok = 0;
int gt = 0;
int sleeps = 0;
int lull = 0;
int tet = 0;
int swars = 0;
int strek = 0;
int panther = 0;
int up = 0;
int cat = 0;
int song = 0;

//------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------


// The included libraries
#include <SPI.h>
#include <WiFiNINA.h>
#include <BlynkSimpleWiFiNINA.h>

//------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------


BlynkTimer timer;
```

```
// This function is called every time the Virtual Pin 1 state changes
BLYNK_WRITE(V1)
{
  // Set incoming value from pin V1 to a variable
  int Hp = param.asInt();

  // Print the changing value for visual control
  Serial.println(Hp);

  // When the value is 1 start the HarryPottor function
  if (Hp = 1)
  {
    Serial.print("Harry Potter");
    HarryPottor();
  }
}

BLYNK_WRITE(V2)
{
  int zel = param.asInt();
  Serial.println(zel);
  if (zel == 1 )
  {
    Serial.print("Zelda");
    zelda();
  }
}


BLYNK_WRITE(V3)
{
  int nok = param.asInt();
  Serial.println(nok);
  if (nok == 1)
  {
    Serial.print("Nokia");
    Nokia();
  }
}


BLYNK_WRITE(V4)
{
  int gt = param.asInt();
  Serial.println(gt);
  if (gt == 1)
  {
    Serial.print("got");
    GameOfThrones();
  }
}
```

```
BLYNK_WRITE(V5)
{
  int sleeps = param.asInt();
  Serial.println(sleeps);
  if (sleeps == 1)
  {
    Serial.print("lion");
    TheLionSleepsTonight();
  }
}


BLYNK_WRITE(V6)
{
  int lull = param.asInt();
  Serial.println(lull);
  if (lull == 1)
  {
    Serial.print("zelda LUll");
    ZeldaLullaby();
  }
}


BLYNK_WRITE(V7)
{
  int tet = param.asInt();
  Serial.println(tet);
  if (tet == 1)
  {
    Serial.print("tet");
    Tetris();
  }
}


BLYNK_WRITE(V8)
{
  int swars = param.asInt();
  Serial.println(swars);
  if (swars == 1)
  {
    Serial.print("war");
    StarWars();
  }
}


BLYNK_WRITE(V9)
{
```

```
  int strek = param.asInt();
  Serial.println(strek);
  if (strek == 1)
  {
    Serial.print("trek");
    StarTrek();
  }
}


BLYNK_WRITE(V10)
{
  int panther = param.asInt();
  Serial.println(panther);
  if (panther == 1)
  {
    Serial.print("pink");
    PinkPanther();
  }
}


BLYNK_WRITE(V11)
{
  int up = param.asInt();
  Serial.println(up);
  if (up == 1)
  {
    Serial.print("rickroll");
    RickRoll();
  }
}


BLYNK_WRITE(V12)
{
  int cat = param.asInt();
  Serial.println(cat);
  if (cat == 1)
  {
    Serial.print("Keyboard");
    KeyboardCat();
  }
}


BLYNK_WRITE(V13)
{
  int song = param.asInt();
  Serial.println(song);
  if (song == 1)
  {
```

```
    Serial.print("jiggly");
    JigglyPuffSong();
  }
}

//-------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------

// Conecting to the WIFI and printing the stage
void setup_wifi()
{
  delay(10);

  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, pass);

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

//-------------------------------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------------------
//-------------------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------

// Set the notes to be visual readable
#define NOTE_B0  31
#define NOTE_C1  33
#define NOTE_CS1 35
#define NOTE_D1  37
#define NOTE_DS1 39
#define NOTE_E1  41
#define NOTE_F1  44
#define NOTE_FS1 46
#define NOTE_G1  49
#define NOTE_GS1 52
```

```
#define NOTE_A1  55
#define NOTE_AS1 58
#define NOTE_B1  62
#define NOTE_C2  65
#define NOTE_CS2 69
#define NOTE_D2  73
#define NOTE_DS2 78
#define NOTE_E2  82
#define NOTE_F2  87
#define NOTE_FS2 93
#define NOTE_G2  98
#define NOTE_GS2 104
#define NOTE_A2  110
#define NOTE_AS2 117
#define NOTE_B2  123
#define NOTE_C3  131
#define NOTE_CS3 139
#define NOTE_D3  147
#define NOTE_DS3 156
#define NOTE_E3  165
#define NOTE_F3  175
#define NOTE_FS3 185
#define NOTE_G3  196
#define NOTE_GS3 208
#define NOTE_A3  220
#define NOTE_AS3 233
#define NOTE_B3  247
#define NOTE_C4  262
#define NOTE_CS4 277
#define NOTE_D4  294
#define NOTE_DS4 311
#define NOTE_E4  330
#define NOTE_F4  349
#define NOTE_FS4 370
#define NOTE_G4  392
#define NOTE_GS4 415
#define NOTE_A4  440
#define NOTE_AS4 466
#define NOTE_B4  494
#define NOTE_C5  523
#define NOTE_CS5 554
#define NOTE_D5  587
#define NOTE_DS5 622
#define NOTE_E5  659
#define NOTE_F5  698
#define NOTE_FS5 740
#define NOTE_G5  784
#define NOTE_GS5 831
#define NOTE_A5  880
#define NOTE_AS5 932
#define NOTE_B5  988
#define NOTE_C6  1047
```

```
#define NOTE_CS6 1109
#define NOTE_D6  1175
#define NOTE_DS6 1245
#define NOTE_E6  1319
#define NOTE_F6  1397
#define NOTE_FS6 1480
#define NOTE_G6  1568
#define NOTE_GS6 1661
#define NOTE_A6  1760
#define NOTE_AS6 1865
#define NOTE_B6  1976
#define NOTE_C7  2093
#define NOTE_CS7 2217
#define NOTE_D7  2349
#define NOTE_DS7 2489
#define NOTE_E7  2637
#define NOTE_F7  2794
#define NOTE_FS7 2960
#define NOTE_G7  3136
#define NOTE_GS7 3322
#define NOTE_A7  3520
#define NOTE_AS7 3729
#define NOTE_B7  3951
#define NOTE_C8  4186
#define NOTE_CS8 4435
#define NOTE_D8  4699
#define NOTE_DS8 4978
#define REST 0

//--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------


int HarryPottor()
{
  // The Speed of the song change this to make the song slower or faster
  int tempo = 144;

  // The pin where the speaker is conected too
  int buzzer = 11;

  // Notes of the moledy followed by the duration.
  // a 4 means a quarter note, 8 an eighteenth , 16 sixteenth, so on
  // !!negative numbers are used to represent dotted notes,
  // so -4 means a dotted quarter note, that is, a quarter plus an eighteenth!!
  int melody[] = {


    // Hedwig's theme from the Harry Potter Movies
    // Socre from https://musescore.com/user/3811306/scores/4906610
```

```
  REST, 2, NOTE_D4, 4,
  NOTE_G4, -4, NOTE_AS4, 8, NOTE_A4, 4,
  NOTE_G4, 2, NOTE_D5, 4,
  NOTE_C5, -2,
  NOTE_A4, -2,
  NOTE_G4, -4, NOTE_AS4, 8, NOTE_A4, 4,
  NOTE_F4, 2, NOTE_GS4, 4,
  NOTE_D4, -1,
  NOTE_D4, 4,

  NOTE_G4, -4, NOTE_AS4, 8, NOTE_A4, 4, //10
  NOTE_G4, 2, NOTE_D5, 4,
  NOTE_F5, 2, NOTE_E5, 4,
  NOTE_DS5, 2, NOTE_B4, 4,
  NOTE_DS5, -4, NOTE_D5, 8, NOTE_CS5, 4,
  NOTE_CS4, 2, NOTE_B4, 4,
  NOTE_G4, -1,
  NOTE_AS4, 4,

  NOTE_D5, 2, NOTE_AS4, 4,//18
  NOTE_D5, 2, NOTE_AS4, 4,
  NOTE_DS5, 2, NOTE_D5, 4,
  NOTE_CS5, 2, NOTE_A4, 4,
  NOTE_AS4, -4, NOTE_D5, 8, NOTE_CS5, 4,
  NOTE_CS4, 2, NOTE_D4, 4,
  NOTE_D5, -1,
  REST, 4, NOTE_AS4, 4,

  NOTE_D5, 2, NOTE_AS4, 4,//26
  NOTE_D5, 2, NOTE_AS4, 4,
  NOTE_F5, 2, NOTE_E5, 4,
  NOTE_DS5, 2, NOTE_B4, 4,
  NOTE_DS5, -4, NOTE_D5, 8, NOTE_CS5, 4,
  NOTE_CS4, 2, NOTE_AS4, 4,
  NOTE_G4, -1,

};

// Sizeof gives the number of bytes, each int value is composed of two bytes (16 bits)
// There are two values per note (pitch and duration), so for each note there are four bytes
int notes = sizeof(melody) / sizeof(melody[0]) / 2;

// This calculates the duration of a whole note in ms (60s/tempo)*4 beats
int wholenote = (60000 * 4) / tempo;

int divider = 0, noteDuration = 0;


// iterate over the notes of the melody.
// Remember, the array is twice the number of notes (notes + durations)
for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {
```

```
    // Calculates the duration of each note
    divider = melody[thisNote + 1];
    if (divider > 0) {

      // Regular note, just proceed
      noteDuration = (wholenote) / divider;
    } else if (divider < 0) {

      // Dotted notes are represented with negative durations!!
      noteDuration = (wholenote) / abs(divider);
      noteDuration *= 1.5; // increases the duration in half for dotted notes
    }

    // We only play the note for 90% of the duration, leaving 10% as a pause
    tone(buzzer, melody[thisNote], noteDuration * 0.9);

    // Wait for the specief duration before playing the next note.
    delay(noteDuration);

    // Stop the waveform generation before the next note.
    noTone(buzzer);
  }
}

//---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
//---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

int zelda()
{
  int tempo = 88;
  int buzzer = 11;
  int melody[] = {

    NOTE_AS4, -2,  NOTE_F4, 8,  NOTE_F4, 8,  NOTE_AS4, 8, //1
    NOTE_GS4, 16,  NOTE_FS4, 16,  NOTE_GS4, -2,
    NOTE_AS4, -2,  NOTE_FS4, 8,  NOTE_FS4, 8,  NOTE_AS4, 8,
    NOTE_A4, 16,  NOTE_G4, 16,  NOTE_A4, -2,
    REST, 1,

    NOTE_AS4, 4,  NOTE_F4, -4,  NOTE_AS4, 8,  NOTE_AS4, 16,  NOTE_C5, 16, NOTE_D5, 16, NOTE_DS5, 16, //7
    NOTE_F5, 2,  NOTE_F5, 8,  NOTE_F5, 8,  NOTE_F5, 8,  NOTE_FS5, 16, NOTE_GS5, 16,
    NOTE_AS5, -2,  NOTE_AS5, 8,  NOTE_AS5, 8,  NOTE_GS5, 8,  NOTE_FS5, 16,
    NOTE_GS5, -8,  NOTE_FS5, 16,  NOTE_F5, 2,  NOTE_F5, 4,

    NOTE_DS5, -8, NOTE_F5, 16, NOTE_FS5, 2, NOTE_F5, 8, NOTE_DS5, 8, //11
    NOTE_CS5, -8, NOTE_DS5, 16, NOTE_F5, 2, NOTE_DS5, 8, NOTE_CS5, 8,
    NOTE_C5, -8, NOTE_D5, 16, NOTE_E5, 2, NOTE_G5, 8,
    NOTE_F5, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 8, NOTE_F4, 16, NOTE_F4, 8,
```

```
    NOTE_AS4, 4,  NOTE_F4, -4,  NOTE_AS4, 8,  NOTE_AS4, 16,  NOTE_C5, 16, NOTE_D5, 16, NOTE_DS5, 16, //15
    NOTE_F5, 2,  NOTE_F5, 8,  NOTE_F5, 8,  NOTE_F5, 8,  NOTE_FS5, 16, NOTE_GS5, 16,
    NOTE_AS5, -2, NOTE_CS6, 4,
    NOTE_C6, 4, NOTE_A5, 2, NOTE_F5, 4,
    NOTE_FS5, -2, NOTE_AS5, 4,
    NOTE_A5, 4, NOTE_F5, 2, NOTE_F5, 4,

    NOTE_FS5, -2, NOTE_AS5, 4,
    NOTE_A5, 4, NOTE_F5, 2, NOTE_D5, 4,
    NOTE_DS5, -2, NOTE_FS5, 4,
    NOTE_F5, 4, NOTE_CS5, 2, NOTE_AS4, 4,
    NOTE_C5, -8, NOTE_D5, 16, NOTE_E5, 2, NOTE_G5, 8,
    NOTE_F5, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 16, NOTE_F4, 8, NOTE_F4, 16, NOTE_F4, 8

  };

  int notes = sizeof(melody) / sizeof(melody[0]) / 2;

  int wholenote = (60000 * 4) / tempo;

  int divider = 0, noteDuration = 0;

  for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

    divider = melody[thisNote + 1];
    if (divider > 0) {
      noteDuration = (wholenote) / divider;
    } else if (divider < 0) {
      noteDuration = (wholenote) / abs(divider);
      noteDuration *= 1.5;
    }

    tone(buzzer, melody[thisNote], noteDuration * 0.9);
    delay(noteDuration);
    noTone(buzzer);
  }
}

//-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

int Nokia()
{
  int tempo = 180;
  int buzzer = 11;
  int melody[] = {
```

```
    NOTE_E5, 8, NOTE_D5, 8, NOTE_FS4, 4, NOTE_GS4, 4,
    NOTE_CS5, 8, NOTE_B4, 8, NOTE_D4, 4, NOTE_E4, 4,
    NOTE_B4, 8, NOTE_A4, 8, NOTE_CS4, 4, NOTE_E4, 4,
    NOTE_A4, 2,
  };

  int notes = sizeof(melody) / sizeof(melody[0]) / 2;

  int wholenote = (60000 * 4) / tempo;

  int divider = 0, noteDuration = 0;

  for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

    divider = melody[thisNote + 1];
    if (divider > 0) {
      noteDuration = (wholenote) / divider;
    } else if (divider < 0) {
      noteDuration = (wholenote) / abs(divider);
      noteDuration *= 1.5;
    }

    tone(buzzer, melody[thisNote], noteDuration * 0.9);
    delay(noteDuration);
    noTone(buzzer);
  }
}

//--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

int GameOfThrones()
{
  int tempo = 85;
  int buzzer = 11;
  int melody[] = {

    NOTE_G4, 8, NOTE_C4, 8, NOTE_DS4, 16, NOTE_F4, 16, NOTE_G4, 8, NOTE_C4, 8, NOTE_DS4, 16, NOTE_F4, 16, //1
    NOTE_G4, 8, NOTE_C4, 8, NOTE_DS4, 16, NOTE_F4, 16, NOTE_G4, 8, NOTE_C4, 8, NOTE_DS4, 16, NOTE_F4, 16,
    NOTE_G4, 8, NOTE_C4, 8, NOTE_E4, 16, NOTE_F4, 16, NOTE_G4, 8, NOTE_C4, 8, NOTE_E4, 16, NOTE_F4, 16,
    NOTE_G4, 8, NOTE_C4, 8, NOTE_E4, 16, NOTE_F4, 16, NOTE_G4, 8, NOTE_C4, 8, NOTE_E4, 16, NOTE_F4, 16,
    NOTE_G4, -4, NOTE_C4, -4, //5

    NOTE_DS4, 16, NOTE_F4, 16, NOTE_G4, 4, NOTE_C4, 4, NOTE_DS4, 16, NOTE_F4, 16, //6
    NOTE_D4, -1, //7 and 8
    NOTE_F4, -4, NOTE_AS3, -4,
```

```
  NOTE_DS4, 16, NOTE_D4, 16, NOTE_F4, 4, NOTE_AS3, -4,
  NOTE_DS4, 16, NOTE_D4, 16, NOTE_C4, -1, //11 and 12

  //repeats from 5
  NOTE_G4, -4, NOTE_C4, -4, //5

  NOTE_DS4, 16, NOTE_F4, 16, NOTE_G4, 4, NOTE_C4, 4, NOTE_DS4, 16, NOTE_F4, 16, //6
  NOTE_D4, -1, //7 and 8
  NOTE_F4, -4, NOTE_AS3, -4,
  NOTE_DS4, 16, NOTE_D4, 16, NOTE_F4, 4, NOTE_AS3, -4,
  NOTE_DS4, 16, NOTE_D4, 16, NOTE_C4, -1, //11 and 12
  NOTE_G4, -4, NOTE_C4, -4,
  NOTE_DS4, 16, NOTE_F4, 16, NOTE_G4, 4,  NOTE_C4, 4, NOTE_DS4, 16, NOTE_F4, 16,

  NOTE_D4, -2, //15
  NOTE_F4, -4, NOTE_AS3, -4,
  NOTE_D4, -8, NOTE_DS4, -8, NOTE_D4, -8, NOTE_AS3, -8,
  NOTE_C4, -1,
  NOTE_C5, -2,
  NOTE_AS4, -2,
  NOTE_C4, -2,
  NOTE_G4, -2,
  NOTE_DS4, -2,
  NOTE_DS4, -4, NOTE_F4, -4,
  NOTE_G4, -1,

  NOTE_C5, -2, //28
  NOTE_AS4, -2,
  NOTE_C4, -2,
  NOTE_G4, -2,
  NOTE_DS4, -2,
  NOTE_DS4, -4, NOTE_D4, -4,
  NOTE_C5, 8, NOTE_G4, 8, NOTE_GS4, 16, NOTE_AS4, 16, NOTE_C5, 8, NOTE_G4, 8, NOTE_GS4, 16,
NOTE_AS4, 16,
  NOTE_C5, 8, NOTE_G4, 8, NOTE_GS4, 16, NOTE_AS4, 16, NOTE_C5, 8, NOTE_G4, 8, NOTE_GS4, 16,
NOTE_AS4, 16,

  REST, 4, NOTE_GS5, 16, NOTE_AS5, 16, NOTE_C6, 8, NOTE_G5, 8, NOTE_GS5, 16, NOTE_AS5, 16,
  NOTE_C6, 8, NOTE_G5, 16, NOTE_GS5, 16, NOTE_AS5, 16, NOTE_C6, 8, NOTE_G5, 8, NOTE_GS5,
16, NOTE_AS5, 16,
 };

 int notes = sizeof(melody) / sizeof(melody[0]) / 2;

 int wholenote = (60000 * 4) / tempo;

 int divider = 0, noteDuration = 0;

 for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

  divider = melody[thisNote + 1];
  if (divider > 0) {
```

```
      noteDuration = (wholenote) / divider;
    } else if (divider < 0) {
      noteDuration = (wholenote) / abs(divider);
      noteDuration *= 1.5;
    }

    tone(buzzer, melody[thisNote], noteDuration * 0.9);
    delay(noteDuration);
    noTone(buzzer);
  }
}

//-----------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------

int TheLionSleepsTonight()
{
  int tempo = 122;
  int buzzer = 11;
  int melody[] = {

    NOTE_F4, 4, NOTE_G4, 4, NOTE_A4, 8, NOTE_G4, 4, NOTE_A4, 8, //1
    NOTE_AS4, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_F4, 4, NOTE_G4, 8,
    NOTE_A4, 4, NOTE_C4, 8, NOTE_C4, 4, NOTE_C4, 8, NOTE_C4, 4,
    NOTE_C4, 1, //1st ending

    NOTE_F4, 4, NOTE_G4, 4, NOTE_A4, 8, NOTE_G4, 4, NOTE_A4, 8, //repeats from 1
    NOTE_AS4, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_F4, 4, NOTE_G4, 8,
    NOTE_A4, 4, NOTE_C4, 8, NOTE_C4, 4, NOTE_C4, 8, NOTE_C4, 4,
    NOTE_C4, -2,  REST, -8, NOTE_A4, 16, //2nd ending

    NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8,
NOTE_A4, 16, //6
    NOTE_AS4, -8, NOTE_AS4, 16, NOTE_AS4, -8, NOTE_AS4, 16, NOTE_AS4, -8, NOTE_AS4, 16,
NOTE_AS4, -8, NOTE_AS4, 16,
    NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8,
NOTE_A4, 16,
    NOTE_G4, -8, NOTE_G4, 16, NOTE_G4, -8, NOTE_G4, 16, NOTE_G4, -8, NOTE_G4, 16, NOTE_G4, -8,
NOTE_G4, 16,

    NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8,
NOTE_A4, 16, //10
    NOTE_AS4, -8, NOTE_AS4, 16, NOTE_AS4, -8, NOTE_AS4, 16, NOTE_AS4, -8, NOTE_AS4, 16,
NOTE_AS4, -8, NOTE_AS4, 16,
    NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8, NOTE_A4, 16, NOTE_A4, -8,
NOTE_A4, 16,
    NOTE_G4, -8, NOTE_G4, 16, NOTE_G4, -8, NOTE_G4, 16, NOTE_G4, -8, NOTE_G4, 16, NOTE_G4, -8,
NOTE_G4, 16,

    NOTE_F4, 4, NOTE_G4, 4, NOTE_A4, 8, NOTE_G4, 4, NOTE_A4, 8, //14
```

```
NOTE_AS4, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_F4, 4, NOTE_G4, 8,
NOTE_A4, 4, NOTE_G4, 4, NOTE_F4, 4, NOTE_A4, 4,
NOTE_G4, 1,
NOTE_C5, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_A4, 4, NOTE_C5, 8,
NOTE_AS4, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_F4, 4, NOTE_G4, 8,
NOTE_A4, 4, NOTE_G4, 4, NOTE_F4, 4, NOTE_A4, 4,
NOTE_G4, 1,

NOTE_C5, 1, //22
NOTE_C5, 4, NOTE_AS4, 8, NOTE_C5, 8, NOTE_AS4, 2,
NOTE_A4, 4, NOTE_C4, 8, NOTE_C4, 4, NOTE_C4, 8, NOTE_C4, 4,
NOTE_C4, 1,

REST, 4, NOTE_A4, 8, NOTE_G4, 8, NOTE_F4, 8, NOTE_E4, 8, NOTE_D4, 8, NOTE_C4, 8,
NOTE_D4, 1,
REST, 4, NOTE_A4, 8, NOTE_G4, 8, NOTE_F4, 8, NOTE_E4, 8, NOTE_D4, 8, NOTE_C4, 8,
NOTE_D4, 1,

NOTE_F4, 4, NOTE_G4, 4, NOTE_A4, 8, NOTE_G4, 4, NOTE_A4, 8, //repeats from 14
NOTE_AS4, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_F4, 4, NOTE_G4, 8,
NOTE_A4, 4, NOTE_G4, 4, NOTE_F4, 4, NOTE_A4, 4,
NOTE_G4, 1,
NOTE_C5, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_A4, 4, NOTE_C5, 8,
NOTE_AS4, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_F4, 4, NOTE_G4, 8,
NOTE_A4, 4, NOTE_G4, 4, NOTE_F4, 4, NOTE_A4, 4,
NOTE_G4, 1,

NOTE_C5, 1, //22
NOTE_C5, 4, NOTE_AS4, 8, NOTE_C5, 8, NOTE_AS4, 2,
NOTE_A4, 4, NOTE_C4, 8, NOTE_C4, 4, NOTE_C4, 8, NOTE_C4, 4,
NOTE_C4, 1,

REST, 4, NOTE_A4, 8, NOTE_G4, 8, NOTE_F4, 8, NOTE_E4, 8, NOTE_D4, 8, NOTE_C4, 8,
NOTE_D4, 1,
REST, 4, NOTE_A4, 8, NOTE_G4, 8, NOTE_F4, 8, NOTE_E4, 8, NOTE_D4, 8, NOTE_C4, 8,
NOTE_D4, 1,

NOTE_F4, 4, NOTE_G4, 4, NOTE_A4, 8, NOTE_G4, 4, NOTE_A4, 8, //30
NOTE_AS4, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_F4, 4, NOTE_G4, 8,
NOTE_A4, 4, NOTE_C4, 8, NOTE_C4, 4, NOTE_C4, 8, NOTE_C4, 4,
NOTE_C4, 1,

NOTE_F4, 4, NOTE_G4, 4, NOTE_A4, 8, NOTE_G4, 4, NOTE_A4, 8, //repeats from 14 (again)
NOTE_AS4, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_F4, 4, NOTE_G4, 8,
NOTE_A4, 4, NOTE_G4, 4, NOTE_F4, 4, NOTE_A4, 4,
NOTE_G4, 1,
NOTE_C5, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_A4, 4, NOTE_C5, 8,
NOTE_AS4, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_F4, 4, NOTE_G4, 8,
NOTE_A4, 4, NOTE_G4, 4, NOTE_F4, 4, NOTE_A4, 4,
NOTE_G4, 1,

NOTE_C5, 1, //22
```

```
  NOTE_C5, 4, NOTE_AS4, 8, NOTE_C5, 8, NOTE_AS4, 2,
  NOTE_A4, 4, NOTE_C4, 8, NOTE_C4, 4, NOTE_C4, 8, NOTE_C4, 4,
  NOTE_C4, 1,

  REST, 4, NOTE_A4, 8, NOTE_G4, 8, NOTE_F4, 8, NOTE_E4, 8, NOTE_D4, 8, NOTE_C4, 8,
  NOTE_D4, 1,
  REST, 4, NOTE_A4, 8, NOTE_G4, 8, NOTE_F4, 8, NOTE_E4, 8, NOTE_D4, 8, NOTE_C4, 8,
  NOTE_D4, 1,

  NOTE_F4, 4, NOTE_G4, 4, NOTE_A4, 8, NOTE_G4, 4, NOTE_A4, 8, //30
  NOTE_AS4, 4, NOTE_A4, 4, NOTE_G4, 8, NOTE_F4, 4, NOTE_G4, 8,
  NOTE_A4, 4, NOTE_C4, 8, NOTE_C4, 4, NOTE_C4, 8, NOTE_C4, 4,
  NOTE_C4, 1,

 };

 int notes = sizeof(melody) / sizeof(melody[0]) / 2;

 int wholenote = (60000 * 4) / tempo;

 int divider = 0, noteDuration = 0;

 for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

  divider = melody[thisNote + 1];
  if (divider > 0) {
   noteDuration = (wholenote) / divider;
  } else if (divider < 0) {
   noteDuration = (wholenote) / abs(divider);
   noteDuration *= 1.5;
  }

  tone(buzzer, melody[thisNote], noteDuration * 0.9);
  delay(noteDuration);
  noTone(buzzer);
 }
}

//-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------------------------------------------------------------

int ZeldaLullaby()
{
 int tempo = 108;
 int buzzer = 11;
 int melody[] = {

  NOTE_E4, 2, NOTE_G4, 4,
  NOTE_D4, 2, NOTE_C4, 8, NOTE_D4, 8,
  NOTE_E4, 2, NOTE_G4, 4,
```

```
  NOTE_D4, -2,
  NOTE_E4, 2, NOTE_G4, 4,
  NOTE_D5, 2, NOTE_C5, 4,
  NOTE_G4, 2, NOTE_F4, 8, NOTE_E4, 8,
  NOTE_D4, -2,
  NOTE_E4, 2, NOTE_G4, 4,
  NOTE_D4, 2, NOTE_C4, 8, NOTE_D4, 8,
  NOTE_E4, 2, NOTE_G4, 4,
  NOTE_D4, -2,
  NOTE_E4, 2, NOTE_G4, 4,

  NOTE_D5, 2, NOTE_C5, 4,
  NOTE_G4, 2, NOTE_F4, 8, NOTE_E4, 8,
  NOTE_F4, 8, NOTE_E4, 8, NOTE_C4, 2,
  NOTE_F4, 2, NOTE_E4, 8, NOTE_D4, 8,
  NOTE_E4, 8, NOTE_D4, 8, NOTE_A3, 2,
  NOTE_G4, 2, NOTE_F4, 8, NOTE_E4, 8,
  NOTE_F4, 8, NOTE_E4, 8, NOTE_C4, 4, NOTE_F4, 4,
  NOTE_C5, -2,

 };

 int notes = sizeof(melody) / sizeof(melody[0]) / 2;
 int wholenote = (60000 * 4) / tempo;

 int divider = 0, noteDuration = 0;

 for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

  divider = melody[thisNote + 1];
  if (divider > 0) {
   noteDuration = (wholenote) / divider;
  } else if (divider < 0) {
   noteDuration = (wholenote) / abs(divider);
   noteDuration *= 1.5;
  }

  tone(buzzer, melody[thisNote], noteDuration * 0.9);
  delay(noteDuration);
  noTone(buzzer);
 }
}

//--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

int Tetris()
{
 int tempo = 108;
 int buzzer = 11;
```

```
  int melody[] = {

    NOTE_E4, 2, NOTE_G4, 4,
    NOTE_D4, 2, NOTE_C4, 8, NOTE_D4, 8,
    NOTE_E4, 2, NOTE_G4, 4,
    NOTE_D4, -2,
    NOTE_E4, 2, NOTE_G4, 4,
    NOTE_D5, 2, NOTE_C5, 4,
    NOTE_G4, 2, NOTE_F4, 8, NOTE_E4, 8,
    NOTE_D4, -2,
    NOTE_E4, 2, NOTE_G4, 4,
    NOTE_D4, 2, NOTE_C4, 8, NOTE_D4, 8,
    NOTE_E4, 2, NOTE_G4, 4,
    NOTE_D4, -2,
    NOTE_E4, 2, NOTE_G4, 4,

    NOTE_D5, 2, NOTE_C5, 4,
    NOTE_G4, 2, NOTE_F4, 8, NOTE_E4, 8,
    NOTE_F4, 8, NOTE_E4, 8, NOTE_C4, 2,
    NOTE_F4, 2, NOTE_E4, 8, NOTE_D4, 8,
    NOTE_E4, 8, NOTE_D4, 8, NOTE_A3, 2,
    NOTE_G4, 2, NOTE_F4, 8, NOTE_E4, 8,
    NOTE_F4, 8, NOTE_E4, 8, NOTE_C4, 4, NOTE_F4, 4,
    NOTE_C5, -2,

  };

  int notes = sizeof(melody) / sizeof(melody[0]) / 2;

  // this calculates the duration of a whole note in ms
  int wholenote = (60000 * 4) / tempo;

  int divider = 0, noteDuration = 0;

  for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

    divider = melody[thisNote + 1];
    if (divider > 0) {
      noteDuration = (wholenote) / divider;
    } else if (divider < 0) {
      noteDuration = (wholenote) / abs(divider);
      noteDuration *= 1.5;
    }

    tone(buzzer, melody[thisNote], noteDuration * 0.9);
    delay(noteDuration);
    noTone(buzzer);
  }
}

//----------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------
```

```
//------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

int StarWars()
{
  int tempo = 108;
  int buzzer = 11;
  int melody[] = {

    NOTE_AS4, 8, NOTE_AS4, 8, NOTE_AS4, 8, //1
    NOTE_F5, 2, NOTE_C6, 2,
    NOTE_AS5, 8, NOTE_A5, 8, NOTE_G5, 8, NOTE_F6, 2, NOTE_C6, 4,
    NOTE_AS5, 8, NOTE_A5, 8, NOTE_G5, 8, NOTE_F6, 2, NOTE_C6, 4,
    NOTE_AS5, 8, NOTE_A5, 8, NOTE_AS5, 8, NOTE_G5, 2, NOTE_C5, 8, NOTE_C5, 8, NOTE_C5, 8,
    NOTE_F5, 2, NOTE_C6, 2,
    NOTE_AS5, 8, NOTE_A5, 8, NOTE_G5, 8, NOTE_F6, 2, NOTE_C6, 4,

    NOTE_AS5, 8, NOTE_A5, 8, NOTE_G5, 8, NOTE_F6, 2, NOTE_C6, 4, //8
    NOTE_AS5, 8, NOTE_A5, 8, NOTE_AS5, 8, NOTE_G5, 2, NOTE_C5, -8, NOTE_C5, 16,
    NOTE_D5, -4, NOTE_D5, 8, NOTE_AS5, 8, NOTE_A5, 8, NOTE_G5, 8, NOTE_F5, 8,
    NOTE_F5, 8, NOTE_G5, 8, NOTE_A5, 8, NOTE_G5, 4, NOTE_D5, 8, NOTE_E5, 4, NOTE_C5, -8, NOTE_C5, 16,
    NOTE_D5, -4, NOTE_D5, 8, NOTE_AS5, 8, NOTE_A5, 8, NOTE_G5, 8, NOTE_F5, 8,

    NOTE_C6, -8, NOTE_G5, 16, NOTE_G5, 2, REST, 8, NOTE_C5, 8, //13
    NOTE_D5, -4, NOTE_D5, 8, NOTE_AS5, 8, NOTE_A5, 8, NOTE_G5, 8, NOTE_F5, 8,
    NOTE_F5, 8, NOTE_G5, 8, NOTE_A5, 8, NOTE_G5, 4, NOTE_D5, 8, NOTE_E5, 4, NOTE_C6, -8, NOTE_C6, 16,
    NOTE_F6, 4, NOTE_DS6, 8, NOTE_CS6, 4, NOTE_C6, 8, NOTE_AS5, 4, NOTE_GS5, 8, NOTE_G5, 4, NOTE_F5, 8,
    NOTE_C6, 1

  };

  int notes = sizeof(melody) / sizeof(melody[0]) / 2;
  int wholenote = (60000 * 4) / tempo;

  int divider = 0, noteDuration = 0;

  for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

    divider = melody[thisNote + 1];
    if (divider > 0) {
      noteDuration = (wholenote) / divider;
    } else if (divider < 0) {
      noteDuration = (wholenote) / abs(divider);
      noteDuration *= 1.5;
    }

    tone(buzzer, melody[thisNote], noteDuration * 0.9);
    delay(noteDuration);
    noTone(buzzer);
```

```
  }
}

//-----------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------

int StarTrek()
{
  int tempo = 80;
  int buzzer = 11;
  int melody[] = {

    NOTE_D4, -8, NOTE_G4, 16, NOTE_C5, -4,
    NOTE_B4, 8, NOTE_G4, -16, NOTE_E4, -16, NOTE_A4, -16,
    NOTE_D5, 2,

  };

  int notes = sizeof(melody) / sizeof(melody[0]) / 2;

  int wholenote = (60000 * 4) / tempo;

  int divider = 0, noteDuration = 0;

  for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

    divider = melody[thisNote + 1];
    if (divider > 0) {
      noteDuration = (wholenote) / divider;
    } else if (divider < 0) {
      noteDuration = (wholenote) / abs(divider);
      noteDuration *= 1.5;
    }

    tone(buzzer, melody[thisNote], noteDuration * 0.9);
    delay(noteDuration);
    noTone(buzzer);
  }
}

//-----------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------------------------------------
------------------------------------------------------------------------------------------

int PinkPanther()
{
  int tempo = 120;
  int buzzer = 11;
  int melody[] = {
```

```
  REST, 2, REST, 4, REST, 8, NOTE_DS4, 8,
  NOTE_E4, -4, REST, 8, NOTE_FS4, 8, NOTE_G4, -4, REST, 8, NOTE_DS4, 8,
  NOTE_E4, -8, NOTE_FS4, 8,  NOTE_G4, -8, NOTE_C5, 8, NOTE_B4, -8, NOTE_E4, 8, NOTE_G4, -8, NOTE_B4, 8,
  NOTE_AS4, 2, NOTE_A4, -16, NOTE_G4, -16, NOTE_E4, -16, NOTE_D4, -16,
  NOTE_E4, 2, REST, 4, REST, 8, NOTE_DS4, 4,

  NOTE_E4, -4, REST, 8, NOTE_FS4, 8, NOTE_G4, -4, REST, 8, NOTE_DS4, 8,
  NOTE_E4, -8, NOTE_FS4, 8,  NOTE_G4, -8, NOTE_C5, 8, NOTE_B4, -8, NOTE_G4, 8, NOTE_B4, -8, NOTE_E5, 8,
  NOTE_DS5, 1,
  NOTE_D5, 2, REST, 4, REST, 8, NOTE_DS4, 8,
  NOTE_E4, -4, REST, 8, NOTE_FS4, 8, NOTE_G4, -4, REST, 8, NOTE_DS4, 8,
  NOTE_E4, -8, NOTE_FS4, 8,  NOTE_G4, -8, NOTE_C5, 8, NOTE_B4, -8, NOTE_E4, 8, NOTE_G4, -8, NOTE_B4, 8,

  NOTE_AS4, 2, NOTE_A4, -16, NOTE_G4, -16, NOTE_E4, -16, NOTE_D4, -16,
  NOTE_E4, -4, REST, 4,
  REST, 4, NOTE_E5, -8, NOTE_D5, 8, NOTE_B4, -8, NOTE_A4, 8, NOTE_G4, -8, NOTE_E4, -8,
  NOTE_AS4, 16, NOTE_A4, -8, NOTE_AS4, 16, NOTE_A4, -8, NOTE_AS4, 16, NOTE_A4, -8, NOTE_AS4, 16, NOTE_A4, -8,
  NOTE_G4, -16, NOTE_E4, -16, NOTE_D4, -16, NOTE_E4, 16, NOTE_E4, 16, NOTE_E4, 2,

 };
 int notes = sizeof(melody) / sizeof(melody[0]) / 2;
 int wholenote = (60000 * 4) / tempo;
 int divider = 0, noteDuration = 0;
 for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

  divider = melody[thisNote + 1];
  if (divider > 0) {

   noteDuration = (wholenote) / divider;
  } else if (divider < 0) {
   noteDuration = (wholenote) / abs(divider);
   noteDuration *= 1.5;
  }

  tone(buzzer, melody[thisNote], noteDuration * 0.9);
  delay(noteDuration);
  noTone(buzzer);
 }
}

//------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
//------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

int RickRoll()
{
```

```
int tempo = 114;
int buzzer = 11;
int melody[] = {

  NOTE_D5, -4, NOTE_E5, -4, NOTE_A4, 4, //1
  NOTE_E5, -4, NOTE_FS5, -4, NOTE_A5, 16, NOTE_G5, 16, NOTE_FS5, 8,
  NOTE_D5, -4, NOTE_E5, -4, NOTE_A4, 2,
  NOTE_A4, 16, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 8, NOTE_D5, 16,
  NOTE_D5, -4, NOTE_E5, -4, NOTE_A4, 4, //repeat from 1
  NOTE_E5, -4, NOTE_FS5, -4, NOTE_A5, 16, NOTE_G5, 16, NOTE_FS5, 8,
  NOTE_D5, -4, NOTE_E5, -4, NOTE_A4, 2,
  NOTE_A4, 16, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 8, NOTE_D5, 16,
  REST, 4, NOTE_B4, 8, NOTE_CS5, 8, NOTE_D5, 8, NOTE_D5, 8, NOTE_E5, 8, NOTE_CS5, -8,
  NOTE_B4, 16, NOTE_A4, 2, REST, 4,

  REST, 8, NOTE_B4, 8, NOTE_B4, 8, NOTE_CS5, 8, NOTE_D5, 8, NOTE_B4, 4, NOTE_A4, 8, //7
  NOTE_A5, 8, REST, 8, NOTE_A5, 8, NOTE_E5, -4, REST, 4,
  NOTE_B4, 8, NOTE_B4, 8, NOTE_CS5, 8, NOTE_D5, 8, NOTE_B4, 8, NOTE_D5, 8, NOTE_E5, 8, REST,
8,
  REST, 8, NOTE_CS5, 8, NOTE_B4, 8, NOTE_A4, -4, REST, 4,
  REST, 8, NOTE_B4, 8, NOTE_B4, 8, NOTE_CS5, 8, NOTE_D5, 8, NOTE_B4, 8, NOTE_A4, 4,
  NOTE_E5, 8, NOTE_E5, 8, NOTE_E5, 8, NOTE_FS5, 8, NOTE_E5, 4, REST, 4,

  NOTE_D5, 2, NOTE_E5, 8, NOTE_FS5, 8, NOTE_D5, 8, //13
  NOTE_E5, 8, NOTE_E5, 8, NOTE_E5, 8, NOTE_FS5, 8, NOTE_E5, 4, NOTE_A4, 4,
  REST, 2, NOTE_B4, 8, NOTE_CS5, 8, NOTE_D5, 8, NOTE_B4, 8,
  REST, 8, NOTE_E5, 8, NOTE_FS5, 8, NOTE_E5, -4, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16,
NOTE_B4, 16,
  NOTE_FS5, -8, NOTE_FS5, -8, NOTE_E5, -4, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16, NOTE_B4,
16,

  NOTE_E5, -8, NOTE_E5, -8, NOTE_D5, -8, NOTE_CS5, 16, NOTE_B4, -8, NOTE_A4, 16, NOTE_B4, 16,
NOTE_D5, 16, NOTE_B4, 16, //18
  NOTE_D5, 4, NOTE_E5, 8, NOTE_CS5, -8, NOTE_B4, 16, NOTE_A4, 8, NOTE_A4, 8, NOTE_A4, 8,
  NOTE_E5, 4, NOTE_D5, 2, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16, NOTE_B4, 16,
  NOTE_FS5, -8, NOTE_FS5, -8, NOTE_E5, -4, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16, NOTE_B4,
16,
  NOTE_A5, 4, NOTE_CS5, 8, NOTE_D5, -8, NOTE_CS5, 16, NOTE_B4, 8, NOTE_A4, 16, NOTE_B4, 16,
NOTE_D5, 16, NOTE_B4, 16,

  NOTE_D5, 4, NOTE_E5, 8, NOTE_CS5, -8, NOTE_B4, 16, NOTE_A4, 4, NOTE_A4, 8, //23
  NOTE_E5, 4, NOTE_D5, 2, REST, 4,
  REST, 8, NOTE_B4, 8, NOTE_D5, 8, NOTE_B4, 8, NOTE_D5, 8, NOTE_E5, 4, REST, 8,
  REST, 8, NOTE_CS5, 8, NOTE_B4, 8, NOTE_A4, -4, REST, 4,
  REST, 8, NOTE_B4, 8, NOTE_B4, 8, NOTE_CS5, 8, NOTE_D5, 8, NOTE_B4, 8, NOTE_A4, 4,
  REST, 8, NOTE_A5, 8, NOTE_A5, 8, NOTE_E5, 8, NOTE_FS5, 8, NOTE_E5, 8, NOTE_D5, 8,

  REST, 8, NOTE_A4, 8, NOTE_B4, 8, NOTE_CS5, 8, NOTE_D5, 8, NOTE_B4, 8, //29
  REST, 8, NOTE_CS5, 8, NOTE_B4, 8, NOTE_A4, -4, REST, 4,
  NOTE_B4, 8, NOTE_B4, 8, NOTE_CS5, 8, NOTE_D5, 8, NOTE_B4, 8, NOTE_A4, 4, REST, 8,
  REST, 8, NOTE_E5, 8, NOTE_E5, 8, NOTE_FS5, 4, NOTE_E5, -4,
  NOTE_D5, 2, NOTE_D5, 8, NOTE_E5, 8, NOTE_FS5, 8, NOTE_E5, 4,
```

```
  NOTE_E5, 8, NOTE_E5, 8, NOTE_FS5, 8, NOTE_E5, 8, NOTE_A4, 8, NOTE_A4, 4,

  REST, -4, NOTE_A4, 8, NOTE_B4, 8, NOTE_CS5, 8, NOTE_D5, 8, NOTE_B4, 8, //35
  REST, 8, NOTE_E5, 8, NOTE_FS5, 8, NOTE_E5, -4, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16,
NOTE_B4, 16,
  NOTE_FS5, -8, NOTE_FS5, -8, NOTE_E5, -4, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16, NOTE_B4,
16,
  NOTE_E5, -8, NOTE_E5, -8, NOTE_D5, -8, NOTE_CS5, 16, NOTE_B4, 8, NOTE_A4, 16, NOTE_B4, 16,
NOTE_D5, 16, NOTE_B4, 16,
  NOTE_D5, 4, NOTE_E5, 8, NOTE_CS5, -8, NOTE_B4, 16, NOTE_A4, 4, NOTE_A4, 8,

  NOTE_E5, 4, NOTE_D5, 2, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16, NOTE_B4, 16, //40
  NOTE_FS5, -8, NOTE_FS5, -8, NOTE_E5, -4, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16, NOTE_B4,
16,
  NOTE_A5, 4, NOTE_CS5, 8, NOTE_D5, -8, NOTE_CS5, 16, NOTE_B4, 8, NOTE_A4, 16, NOTE_B4, 16,
NOTE_D5, 16, NOTE_B4, 16,
  NOTE_D5, 4, NOTE_E5, 8, NOTE_CS5, -8, NOTE_B4, 16, NOTE_A4, 4, NOTE_A4, 8,
  NOTE_E5, 4, NOTE_D5, 2, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16, NOTE_B4, 16,

  NOTE_FS5, -8, NOTE_FS5, -8, NOTE_E5, -4, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16, NOTE_B4,
16, //45
  NOTE_A5, 4, NOTE_CS5, 8, NOTE_D5, -8, NOTE_CS5, 16, NOTE_B4, 8, NOTE_A4, 16, NOTE_B4, 16,
NOTE_D5, 16, NOTE_B4, 16,
  NOTE_D5, 4, NOTE_E5, 8, NOTE_CS5, -8, NOTE_B4, 16, NOTE_A4, 4, NOTE_A4, 8,
  NOTE_E5, 4, NOTE_D5, 2, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16, NOTE_B4, 16,
  NOTE_FS5, -8, NOTE_FS5, -8, NOTE_E5, -4, NOTE_A4, 16, NOTE_B4, 16, NOTE_D5, 16, NOTE_B4,
16, //45

  NOTE_A5, 4, NOTE_CS5, 8, NOTE_D5, -8, NOTE_CS5, 16, NOTE_B4, 8, NOTE_A4, 16, NOTE_B4, 16,
NOTE_D5, 16, NOTE_B4, 16,
  NOTE_D5, 4, NOTE_E5, 8, NOTE_CS5, -8, NOTE_B4, 16, NOTE_A4, 4, NOTE_A4, 8,

  NOTE_E5, 4, NOTE_D5, 2, REST, 4
 };

 int notes = sizeof(melody) / sizeof(melody[0]) / 2;
 int wholenote = (60000 * 4) / tempo;

 int divider = 0, noteDuration = 0;

 for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

  divider = melody[thisNote + 1];
  if (divider > 0) {
   noteDuration = (wholenote) / divider;
  } else if (divider < 0) {
   noteDuration = (wholenote) / abs(divider);
   noteDuration *= 1.5;
  }
  tone(buzzer, melody[thisNote], noteDuration * 0.9);
  delay(noteDuration);
  noTone(buzzer);
```

```
 }
}

//-----------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------
//-----------------------------------------------------------------------------------------------------------------------------------
--------------------------------------------------------------------------------

int KeyboardCat()
{
 int tempo = 160;
 int buzzer = 11;
 int melody[] = {

  REST, 1,
  REST, 1,
  NOTE_C4, 4, NOTE_E4, 4, NOTE_G4, 4, NOTE_E4, 4,
  NOTE_C4, 4, NOTE_E4, 8, NOTE_G4, -4, NOTE_E4, 4,
  NOTE_A3, 4, NOTE_C4, 4, NOTE_E4, 4, NOTE_C4, 4,
  NOTE_A3, 4, NOTE_C4, 8, NOTE_E4, -4, NOTE_C4, 4,
  NOTE_G3, 4, NOTE_B3, 4, NOTE_D4, 4, NOTE_B3, 4,
  NOTE_G3, 4, NOTE_B3, 8, NOTE_D4, -4, NOTE_B3, 4,

  NOTE_G3, 4, NOTE_G3, 8, NOTE_G3, -4, NOTE_G3, 8, NOTE_G3, 4,
  NOTE_G3, 4, NOTE_G3, 4, NOTE_G3, 8, NOTE_G3, 4,
  NOTE_C4, 4, NOTE_E4, 4, NOTE_G4, 4, NOTE_E4, 4,
  NOTE_C4, 4, NOTE_E4, 8, NOTE_G4, -4, NOTE_E4, 4,
  NOTE_A3, 4, NOTE_C4, 4, NOTE_E4, 4, NOTE_C4, 4,
  NOTE_A3, 4, NOTE_C4, 8, NOTE_E4, -4, NOTE_C4, 4,
  NOTE_G3, 4, NOTE_B3, 4, NOTE_D4, 4, NOTE_B3, 4,
  NOTE_G3, 4, NOTE_B3, 8, NOTE_D4, -4, NOTE_B3, 4,

  NOTE_G3, -1,

 };

 int notes = sizeof(melody) / sizeof(melody[0]) / 2;
 int wholenote = (60000 * 4) / tempo;

 int divider = 0, noteDuration = 0;
 for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

  divider = melody[thisNote + 1];
  if (divider > 0) {
   noteDuration = (wholenote) / divider;
  } else if (divider < 0) {
   noteDuration = (wholenote) / abs(divider);
   noteDuration *= 1.5;
  }
  tone(buzzer, melody[thisNote], noteDuration * 0.9);
  delay(noteDuration);
  noTone(buzzer);
```

```
  }
}

//--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

int JigglyPuffSong()
{
  int tempo = 85;
  int buzzer = 11;
  int melody[] = {

    NOTE_D5, -4, NOTE_A5, 8, NOTE_FS5, 8, NOTE_D5, 8,
    NOTE_E5, -4, NOTE_FS5, 8, NOTE_G5, 4,
    NOTE_FS5, -4, NOTE_E5, 8, NOTE_FS5, 4,
    NOTE_D5, -2,
    NOTE_D5, -4, NOTE_A5, 8, NOTE_FS5, 8, NOTE_D5, 8,
    NOTE_E5, -4, NOTE_FS5, 8, NOTE_G5, 4,
    NOTE_FS5, -1,
    NOTE_D5, -4, NOTE_A5, 8, NOTE_FS5, 8, NOTE_D5, 8,
    NOTE_E5, -4, NOTE_FS5, 8, NOTE_G5, 4,

    NOTE_FS5, -4, NOTE_E5, 8, NOTE_FS5, 4,
    NOTE_D5, -2,
    NOTE_D5, -4, NOTE_A5, 8, NOTE_FS5, 8, NOTE_D5, 8,
    NOTE_E5, -4, NOTE_FS5, 8, NOTE_G5, 4,
    NOTE_FS5, -1,

  };

  int notes = sizeof(melody) / sizeof(melody[0]) / 2;

  int wholenote = (60000 * 4) / tempo;

  int divider = 0, noteDuration = 0;

  for (int thisNote = 0; thisNote < notes * 2; thisNote = thisNote + 2) {

    divider = melody[thisNote + 1];
    if (divider > 0) {
      noteDuration = (wholenote) / divider;
    } else if (divider < 0) {
      noteDuration = (wholenote) / abs(divider);
      noteDuration *= 1.5;
    }

    tone(buzzer, melody[thisNote], noteDuration * 0.9);
    delay(noteDuration);
    noTone(buzzer);
  }
```

```
}

//--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------

// This function sends Arduino's uptime every second to Virtual Pin 2.
void myTimerEvent()
{
 // You can send any value at any time.
 // Please don't send more that 10 values per second.
 Blynk.virtualWrite(V2, millis() / 1000);
}

//--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------

void setup()
{
 Serial.begin(115200);
 Blynk.begin(auth, ssid, pass);
}

//--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------
//--------------------------------------------------------------------------------------------------------------------------------------
---------------------------------------------------------------------------------

void loop()
{
 Blynk.run();
 timer.run();
}
```

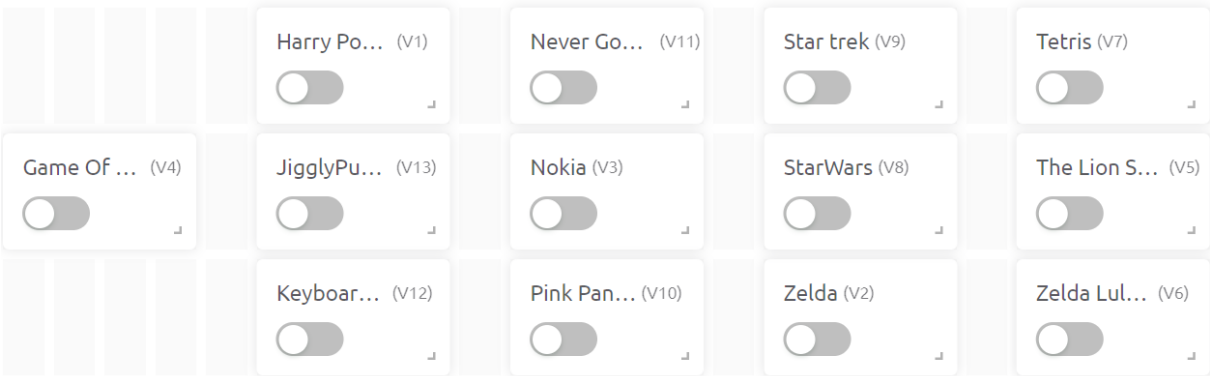
Below you will find what I have created in Blynk for this project.

13 DataStream

13 Datastreams

Id	Name	Alias	Color	Pin	Data Type	Units	Min	Max	Decimals	Default Value
2	Harry Potter	Harry Potter		V1	Integer		0	1	--	0
3	Zelda	Zelda		V2	Integer		0	1	--	0
4	Nokia	Nokia		V3	Integer		0	1	--	0
5	Game Of Thrones	Game Of Thrones		V4	Integer		0	1	--	0
6	The Lion Sleeps Tonight	The Lion Sleeps Tonight		V5	Integer		0	1	--	0
7	Zelda Lullaby	Zelda Lullaby		V6	Integer		0	1	--	0
8	Tetris	Tetris		V7	Integer		0	1	--	0
9	StarWars	StarWars		V8	Integer		0	1	--	0
10	Star trek	Star trek		V9	Integer		0	1	--	0
11	Pink Panter	Pink Panter		V10	Integer		0	1	--	0
12	Never Gonna Give You Up	Never Gonna Give You Up		V11	Integer		0	1	--	0
13	Keyboard Cat	Keyboard Cat		V12	Integer		0	1	--	0
14	JigglyPuff Song	JigglyPuff Song		V13	Integer		0	1	--	0

13 buttons (switches on web dashboard)



13 buttons op mobile

Here you find the link to the GitHub page of this project:

https://github.com/BryanAertgeerts/Jukebox

Here you find the link to the YouTube video of this project:

https://youtu.be/JJPcUCIW3f0