```
/*
  Control a ROYGBIV LED with an analog joystick and play 8 different melodies
depending on the position of the
  joystick when depressed.

  circuit:
  - 8 ohm speaker on digital pin 8
  - 9 blue LED
  - 10 green LED
  - 11 red LED
  - 2 joystick switch
  - A0 joystick X-Axis
  - A1 joystick Y-Axis
  - 5V power joystick

  Project 1
  class: IDEA 310L @ CSU
  date: 12 Sept 2021
  by: Eric Martin

 Songs by https://github.com/robsoncouto/arduino-songs
  Joystick code influenced by https://create.arduino.
cc/projecthub/Raushancpr/control-rgb-led-with-joystick-68f601

*/

#include "pitches.h"

const int JOYCLICK_PIN = 2;
const int RED_PIN = 11;
const int GREEN_PIN = 10;
const int BLUE_PIN = 9;

const int redX = 512;
const int redY = 1023;
const int greenX = 1023;
const int greenY = 0;
const int blueX = 0;
const int blueY = 0;

// ~~~~~~~~~~~~~~~~~~Final Fantasy Victory Fanfare Melody ~~~~~~~~~~~~~~~~~~~~~~~~

int melody_FF[] = {
    NOTE_D4,10, NOTE_D4,12, NOTE_D4,12, NOTE_D4,4,
    NOTE_AS3,4, NOTE_C4,4, NOTE_D4,5, REST,12,
    NOTE_C4,8, NOTE_D4,2
```

```
};

//~~~~~~~~~~~~~~~~~~~~~~~~Mariomelody~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

int melody_Mario[] = {

  NOTE_E5,8, NOTE_E5,8, REST,8, NOTE_E5,8, REST,8, NOTE_C5,8, NOTE_E5,8, //1
  NOTE_G5,4, REST,4, NOTE_G4,8, REST,4,
  NOTE_C5,-4, NOTE_G4,8, REST,4, NOTE_E4,-4, // 3
  NOTE_A4,4, NOTE_B4,4, NOTE_AS4,8, NOTE_A4,4,
  NOTE_G4,-8, NOTE_E5,-8, NOTE_G5,-8, NOTE_A5,4, NOTE_F5,8, NOTE_G5,8,
  REST,8, NOTE_E5,4,NOTE_C5,8, NOTE_D5,8, NOTE_B4,-4,
  NOTE_C5,-4, NOTE_G4,8, REST,4, NOTE_E4,-4, // repeats from 3
  NOTE_A4,4, NOTE_B4,4, NOTE_AS4,8, NOTE_A4,4,
  NOTE_G4,-8, NOTE_E5,-8, NOTE_G5,-8, NOTE_A5,4, NOTE_F5,8, NOTE_G5,8,
  REST,8, NOTE_E5,4,NOTE_C5,8, NOTE_D5,8, NOTE_B4,-4

};

// ~~~~~~~~~~~~~~~~~~~~~~~~Harry Potter Hedwigs Theme Melody ~~~~~~~~~~~~~~~~~~~~
int melody_HP[] = {
  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,
};

// ~~~~~~~~~~~~~~~~~~~ Keyboard Cat Song Melody ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

int melody_Cat[] = {

   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,
};

// ~~~~~~~~~~~~~~~~~~~  Skyrim - Greensleeves Melody ~~~~~~~~~~~~~~~~~~~~~~~~~~

int melody_Skyrim[] = {

  NOTE_G4,8,//1
  NOTE_AS4,4, NOTE_C5,8, NOTE_D5,-8, NOTE_DS5,16, NOTE_D5,8,
  NOTE_C5,4, NOTE_A4,8, NOTE_F4,-8, NOTE_G4,16, NOTE_A4,8,
  NOTE_AS4,4, NOTE_G4,8, NOTE_G4,-8, NOTE_FS4,16, NOTE_G4,8,
  NOTE_A4,4, NOTE_FS4,8, NOTE_D4,4, NOTE_G4
};

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~ GoT Theme Melody ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

int melody_GoT[] = {

  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
};


// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Star Wars Imperial Theme Melody
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

int melody_StarWars[] = {

  NOTE_A4,4, NOTE_A4,4, NOTE_A4,4, NOTE_F4,-8, NOTE_C5,16,

  NOTE_A4,4, NOTE_F4,-8, NOTE_C5,16, NOTE_A4,2,//4
  NOTE_E5,4, NOTE_E5,4, NOTE_E5,4, NOTE_F5,-8, NOTE_C5,16,
  NOTE_A4,4, NOTE_F4,-8, NOTE_C5,16, NOTE_A4,2,

  NOTE_A5,4, NOTE_A4,-8, NOTE_A4,16, NOTE_A5,4, NOTE_GS5,-8, NOTE_G5,16, //7
  NOTE_DS5,16, NOTE_D5,16, NOTE_DS5,8, REST,8, NOTE_A4,8, NOTE_DS5,4, NOTE_D5,-8,
NOTE_CS5,16,

  NOTE_C5,16, NOTE_B4,16, NOTE_C5,16, REST,8, NOTE_F4,8, NOTE_GS4,4, NOTE_F4,-8,
NOTE_A4,-16,//9
  NOTE_C5,4, NOTE_A4,-8, NOTE_C5,16, NOTE_E5,2,
};

// ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Pokemon - JigglyPuff Song Melody
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

int melody_Jiggly[] = {

  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,
};
```

```
//-------------------------------------------SETUP
----------------------------------------


void setup() {
  Serial.begin(9600);
  // Set the Joystick button as an input
  pinMode(JOYCLICK_PIN, INPUT);
  digitalWrite(JOYCLICK_PIN, HIGH);

  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);
};

//--------------------------------------LOOP
-----------------------------------------------------
void loop() {

  // set the analog outputs as analogReads
  int xAxis = analogRead(A0);
  int yAxis = analogRead(A1);

  // set both axes
  xAxis = map(xAxis, 0, 1023, 0, 1023);
  yAxis = map(yAxis, 0, 1023, 1023, 0);

  // allow the ability to smoothly change RGB colors using the joystick
  int distanceRed = sqrt(pow(abs(redX - xAxis), 2) +  pow(abs(redY - yAxis), 2));
  int distanceGreen = sqrt(pow(abs(greenX - xAxis), 2) +  pow(abs(greenY - yAxis),
2));
  int distanceBlue = sqrt(pow(abs(blueX - xAxis), 2) +  pow(abs(blueY - yAxis), 2));

  // map the analog output values from 0 to 1023 to the LED input values of 0 to 255
  // while constraining the values to 0 to 255 to prevent values outside of that
range from occuring
  int brightRed = 255 - constrain(map(distanceRed, 0, 1023, 0, 255), 0, 255);
  int brightGreen = 255 - constrain(map(distanceGreen, 0, 1023, 0, 255), 0, 255);
  int brightBlue = 255 - constrain(map(distanceBlue, 0, 1023, 0, 255), 0, 255);

  analogWrite(RED_PIN, brightRed);
  analogWrite(GREEN_PIN, brightGreen);
  analogWrite(BLUE_PIN, brightBlue);

  Serial.print("KEY: ");
```

```
 Serial.print(digitalRead(JOYCLICK_PIN));
  Serial.print(", X: ");
 Serial.print(xAxis);
  Serial.print(", Y: ");
 Serial.print(yAxis);
  Serial.print(", R: ");
 Serial.print(brightRed);
  Serial.print(", G: ");
 Serial.print(brightGreen);
  Serial.print(", B: ");
 Serial.print(brightBlue);
 Serial.println("\n");

  delay(200);

  // ~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Make Light Brighter
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  while ((digitalRead(JOYCLICK_PIN) == 0) && (xAxis >= 485 && xAxis <= 530) &&
(yAxis >= 485 && yAxis <= 530)) {
     analogWrite(RED_PIN, random(0, 255));
     analogWrite(GREEN_PIN, random(0, 255));
     analogWrite(BLUE_PIN, random(0, 255));
     Serial.print("Random Color");
      delay(2000);
  }


// ~~~~~~~~~~~~~~~~~~~~~~~~ MELODY FINAL FANTASY VICTORY FANFARE
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

  while ((digitalRead(JOYCLICK_PIN) == 0) && (xAxis >= 0 && xAxis <= 5) && (yAxis >=
485 && yAxis <= 530)) {

     Serial.print("Final Fantasy Victory Song");

      int tempo = 80;

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

     // this calculates the duration of a whole note in ms
     int wholenote = (60000 * 2) / 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_FF[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(8, melody_FF[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(8);
     }
  }

 // ~~~~~~~~~~~~~~~~~~~~~~~~ MELODY MARIO ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


  while ((digitalRead(JOYCLICK_PIN) == 0) && (xAxis >= 1015 && xAxis <= 1023)&&
(yAxis >= 485 && yAxis <= 530)) {

    Serial.print("Mario Song");

    int tempo = 200;

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

    // this calculates the duration of a whole note in ms
    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_Mario[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(8, melody_Mario[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(8);
    }
  }


 // ~~~~~~~~~~~~~~~~~~~~~~~~ MELODY HARRY POTTER  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


  while ((digitalRead(JOYCLICK_PIN) == 0) && (yAxis >= 1015 && yAxis <= 1023)&&
(xAxis >= 490 && xAxis <= 510)) {

    Serial.print("Harry Potter - Hedwigs Theme Song");

    int tempo = 140;

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

    // this calculates the duration of a whole note in ms
    int wholenote = (60000 * 2) / 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_HP[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(8, melody_HP[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(8);
    }
  }

 // ~~~~~~~~~~~~~~~~~~~~~~~~~ MELODY Keyboard Cat  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


 while ((digitalRead(JOYCLICK_PIN) == 0) && (yAxis >= 0 && yAxis <= 10) && (xAxis
>= 490 && xAxis <= 510)) {

    Serial.print("Keyboard Cat Song");

    int tempo = 160;

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

    // this calculates the duration of a whole note in ms
    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_Cat[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(8, melody_Cat[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(8);
      }
  }



 // ~~~~~~~~~~~~~~~~~~~~~~~~ MELODY Skyrim  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


  while ((digitalRead(JOYCLICK_PIN) == 0) && (yAxis >= 0 && yAxis <= 10) && (xAxis
>= 0 && xAxis <= 10)) {

    Serial.print("Skyrim - Greensleeves Song");

     int tempo = 70;

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

    // this calculates the duration of a whole note in ms
    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_Skyrim[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(8, melody_Skyrim[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(8);
      }
 }

  // ~~~~~~~~~~~~~~~~~~~~~~~~ MELODY GoT  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


 while ((digitalRead(JOYCLICK_PIN) == 0) && (yAxis >= 1015 && yAxis <= 1023) &&
(xAxis >= 0 && xAxis <= 10)) {

    Serial.print("Star Wars Imperial March Song");

    int tempo = 85;

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

    // this calculates the duration of a whole note in ms
    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_GoT[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(8, melody_GoT[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(8);
       }
  }

   // ~~~~~~~~~~~~~~~~~~~~~~~~ MELODY Star Wars  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


  while ((digitalRead(JOYCLICK_PIN) == 0) && (yAxis >= 1015 && yAxis <= 1023) &&
(xAxis >= 1015 && xAxis <= 1023)) {

    Serial.print("Star Wars Imperial March Song");

     int tempo = 120;

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

    // this calculates the duration of a whole note in ms
    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_StarWars[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(8, melody_StarWars[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(8);
       }
  }


  // ~~~~~~~~~~~~~~~~~~~~~~~~ MELODY Jiggly  ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~


 while ((digitalRead(JOYCLICK_PIN) == 0) && (yAxis >= 0 && yAxis <= 10) && (xAxis
>= 1015 && xAxis <= 1023)) {

    Serial.print("Pokemon - JigglyPuff Song");

    int tempo = 85;

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

    // this calculates the duration of a whole note in ms
    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_Jiggly[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(8, melody_Jiggly[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(8);
      }
  }
}
```