```
#include <EEPROMex.h>
#include <LEDFader.h>

const int maxBrightness = 190;
const int scaleLen = 13;
const int scaleCount = 5;
int currScale = 0;

int scale[scaleCount][scaleLen] = {
  {12,1,2,3,4,5,6,7,8,9,10,11,12}, // Chromatic
  {7,1,3,5,6,8,10,12}, // Major
  {7,1,3,4,6,8,9,11}, // DiaMinor
  {7,1,2,2,5,6,9,11}, // Indian
  {7,1,3,4,6,8,9,11}  // Minor
};

int root = 0;
const byte interruptPin = INT0;
const byte knobPin = A0;
const int piezoPin = 11;
Bounce button = Bounce();
const byte buttonPin = A1;

int menus = 5;
int currMenu = 0;
int pulseRate = 350;
int noteMin = 36;  // C2 - keyboard note minimum
int noteMax = 96;  // C7 - keyboard note maximum

LEDFader leds[] = {
  LEDFader(3), LEDFader(5), LEDFader(6), LEDFader(9),
LEDFader(10), LEDFader(11)
};

byte controlLED = 5;
int value = 0;
int prevValue = 0;

volatile unsigned long microseconds;
volatile byte index = 0;
volatile unsigned long samples[10];

float threshold = 1.7;   // change threshold multiplier
float knobMin = 1;
float knobMax = 1024;

unsigned long previousMillis = 0;
unsigned long currentMillis = 1;
unsigned long menuTimeout = 5000; // 5 seconds timeout in menu
mode

void setup() {
  pinMode(knobPin, INPUT);
```

```
  pinMode(buttonPin, INPUT_PULLUP);
  button.attach(buttonPin);
  button.interval(5);

  randomSeed(analogRead(0));
  Serial.begin(31250);

  attachInterrupt(interruptPin, sample, RISING);
}

void loop() {
  currentMillis = millis();
  checkButton();
  if (index >= 10) { analyzeSample(); }
  checkNote();
  checkLED();
  if (currMenu > 0) { checkMenu(); }
}

void playTone(int frequency, int duration) {
  tone(piezoPin, frequency, duration);
}

int valueToFrequency(int noteValue) {
  int frequency[] = {
    32, 34, 36, 38, 41, 43, 46, 49, 52, 55, 58, 62, 65, 69, 73,
78, 82, 87, 93, 98, 104, 110, 117, 123, 130, 138, 146, 155, 164,
174, 184, 195, 207, 220, 233, 246, 261, 277, 293, 311, 329, 349,
370, 392, 415, 440, 466, 493, 523, 554, 587, 622, 659, 698, 740,
784, 830, 880, 932, 987, 1046, 1109, 1175, 1245, 1319, 1397,
1480, 1568, 1661, 1760, 1865, 1976, 2093, 2217, 2349, 2489,
2637, 2794, 2960, 3136, 3322, 3520, 3720, 3946, 4186, 4435,
4698, 4978
  };
  return frequency[noteValue % 12];  // Возвращаем частоту для
соответствующей ноты
}

void setNote(int value, int velocity, long duration, int
notechannel) {
  for (int i = 0; i < 5; i++) {
    if (!noteArray[i].velocity) {
      noteArray[i] = {0, value, velocity, currentMillis +
duration, currentMillis + duration, notechannel};
      midiSerial(144, notechannel, value, velocity);
      rampUp(i, maxBrightness, duration);
      playTone(valueToFrequency(value), duration);
      break;
    }
  }
}

void midiSerial(int type, int channel, int data1, int data2) {
```

```
  byte statusbyte = (type | ((channel - 1) & 0x0F));
  Serial.write(statusbyte);
  Serial.write(data1 & 0x7F);
  Serial.write(data2 & 0x7F);
}

void rampUp(int ledPin, int value, int time) {
  leds[ledPin].fade(map(value, 0, 255, 0, maxBrightness), time);
}

void rampDown(int ledPin, int value, int time) {
  leds[ledPin].fade(value, time); // fade out
}

void checkLED() {
  for (byte i = 0; i < 6; i++) {
    leds[i].update();
  }
}

void checkButton() {
  button.update();
  if (button.fell()) {
    switch(currMenu) {
      case 0: currMenu = 1; break;
      case 1: handleMenuSelection(value); break;
      default: break;
    }
  }
}

void handleMenuSelection(int menuValue) {
  switch(menuValue) {
    case 0: thresholdMode(); break;
    case 1: scaleMode(); break;
    case 2: channelMode(); break;
    case 3: brightnessMode(); break;
    default: break;
  }
}

void checkMenu() {
  value = map(analogRead(knobPin), knobMin, knobMax, 0, menus);
  if (value != prevValue) {
    leds[prevValue].stop_fade();
    leds[prevValue].set_value(0);
    prevValue = value;
    previousMillis = currentMillis;
  }
  pulse(value, maxBrightness, pulseRate);
  if (currentMillis - previousMillis > menuTimeout) {
    currMenu = 0;
    leds[prevValue].stop_fade();
```

```
    leds[prevValue].set_value(0);
  }
}

void thresholdMode() {
  while (true) {
    threshold = map(analogRead(knobPin), knobMin, knobMax, 1.61,
3.71);
    checkLED();
    if (button.fell()) break;
    analyzeSample();
  }
  currMenu = 0;
}

void scaleMode() {
  while (true) {
    currScale = map(analogRead(knobPin), knobMin, knobMax, 0,
scaleCount);
    checkLED();
    if (button.fell()) break;
  }
  currMenu = 0;
}

void channelMode() {
  while (true) {
    int channelValue = map(analogRead(knobPin), knobMin,
knobMax, 1, 17);
    checkLED();
    if (button.fell()) break;
  }
  currMenu = 0;
}

void brightnessMode() {
  while (true) {
    maxBrightness = map(analogRead(knobPin), knobMin, knobMax,
1, 255);
    checkLED();
    if (button.fell()) break;
  }
  currMenu = 0;
}



void sample() {
  if (index < 10) {
    samples[index] = micros() - microseconds;
    microseconds = samples[index];
    index++;
  }
```

```
}

void analyzeSample() {
  unsigned long avg = 0, max = 0, min = 100000, delta = 0;
  for (byte i = 0; i < 9; i++) {
    unsigned long sample = samples[i + 1];
    max = max(max, sample);
    min = min(min, sample);
    avg += sample;
  }
  avg /= 9;
  delta = max - min;
  if (delta > threshold) {
    setNote(avg % 127, 100, 150, random(1, 5));
  }
  index = 0;
}
```