SmartElex Bharat Explorer Shield (Compatible With Arduino Uno R3, R4, & Q)

USER MANUAL & ACTIVITY GUIDE

11 Games, Projects & Activities

Complete Arduino Code · Pin Diagrams · Wiring Guide

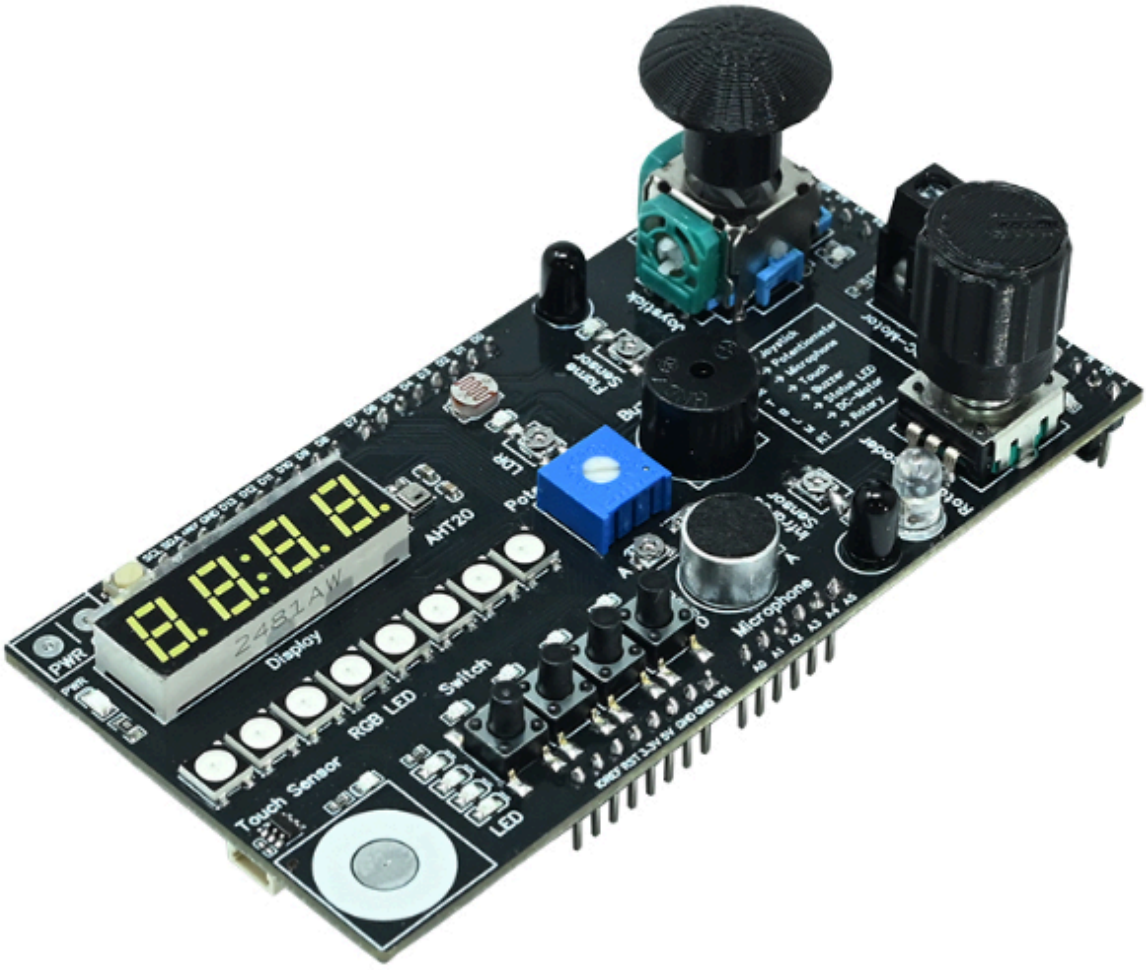

TABLE OF CONTENT

Introduction

Welcome to the SmartElex Bharat Explorer Arduino Shield! This shield packs 17 components onto a single board that plugs directly onto any Arduino Uno. Whether you are a beginner exploring electronics for the first time or an experienced maker looking for a quick prototyping platform, this shield has everything you need in one compact package.

Components on the Shield

All Built-In Components
Sensors: Joystick (X/Y), IR Sensor, Flame Sensor, Touch Sensor, Microphone, LDR (Light), AHT20 (Temp+Humidity), Rotary Encoder
Outputs: 9x RGB NeoPixel LEDs, 4-digit 7-Segment Display (I2C), Buzzer, DC Motor, Status LED
Controls: 4x Push Buttons (S1–S4), Potentiometer, Rotary Encoder with push button
Interface: I2C bus (AHT20 + 7-Segment), Analog pins A0–A3, Digital pins D2–D13

Jumper Configuration

The shield has a jumper that switches shared pins between two modes:

Jumper Positions
Round 1 (R1) — Joystick/Encoder side: A0=Joystick X, A1=Joystick Y, D4=Touch, D6=Encoder CLK
Round 2 (R2) — POT/Motor/Mic side: A0=Microphone, A1=Potentiometer, D4=Motor IN2, D6=BTN4 (S4)

Required Libraries

Install these libraries in Arduino IDE (Sketch → Include Library → Manage Libraries):

- Adafruit NeoPixel
- Adafruit GFX Library
- Adafruit LED Backpack Library
- Adafruit AHTX0

Important Instructions

7-Segment Display — Standard Coding Rules

Whenever you use the 7-segment display in any activity, always follow this standard method to write digits to the display. This ensures consistent, correct output across all projects.

```
display.writeDigitNum(0, thousands);
display.writeDigitNum(1, hundreds);
display.writeDigitNum(2, tens);
display.writeDigitNum(3, ones);
```

7- Segment Display Test Code

```
#include <Wire.h>
#include "Adafruit_LEDBackpack.h"

Adafruit_7segment display = Adafruit_7segment();

int counter = 0;

void setup() {
  display.begin(0x70);
  display.setBrightness(10);
  display.clear();
  display.writeDisplay();
}

void loop() {
  int thousands = counter / 1000;
  int hundreds  = (counter / 100) % 10;
  int tens      = (counter / 10)  % 10;
  int ones      =  counter        % 10;

  display.writeDigitNum(0, thousands);
  display.writeDigitNum(1, hundreds);
  display.writeDigitNum(2, tens);
  display.writeDigitNum(3, ones);
  display.writeDisplay();

  delay(100);  // 100ms per step → full 0-9999 cycle in ~17 minutes

  counter++;
  if (counter > 9999) counter = 0;  // loop back to 0 after 9999
}
```

ACTIVITY 01 Reaction Time Tester

Difficulty: Beginner	Est. Time: 15 minutes	Category: Interactive

Test your reflexes! The buzzer plays a random start tone, all 9 RGB LEDs light up in a random color, and you must press BTN1 as fast as possible. Printed to Serial Monitor. Challenge your friends to beat your score!

Pin & Wiring Reference

Arduino Pin	Component / Label	Direction	Notes
D13	Buzzer	OUTPUT	Plays a start beep
D3	RGB LED Strip (NeoPixel)	OUTPUT	9 LEDs signal GO!
D8	BTN1 (S1)	INPUT	Press to stop the timer
I2C (A4/A5)	7-Segment Display	OUTPUT	Shows reaction time in ms

Wiring Notes

- No external wiring needed — all components are on the shield.
- Plug the shield onto your Arduino Uno.
- Connect USB to your PC and open the Serial Monitor at 9600 baud.
- Jumper settings: No special jumper position needed for this activity.

How to Play / Use

1. Upload the code and open Serial Monitor.wss
2. Wait for the buzzer beep and LEDs to light up — this happens after a random delay (2–5 seconds).
3. Press BTN1 (S1) as quickly as possible when you see the light.
4. Your time appears on the Serial Monitor.
5. The Arduino resets automatically after 3 seconds for the next round.

Arduino Code

```
##include <Wire.h>
#include <Adafruit_NeoPixel.h>
#include "Adafruit_LEDBackpack.h"

#define LED_PIN  3
#define BUZZER   13
#define BTN1     8
#define MAX_MS   9999

Adafruit_NeoPixel strip(9, LED_PIN, NEO_GRB + NEO_KHZ800);
```

```
Adafruit_7segment display = Adafruit_7segment();

void clearDisplay() {
  display.writeDigitRaw(0, 0x00);
  display.writeDigitRaw(1, 0x00);
  display.writeDigitRaw(2, 0x00);
  display.writeDigitRaw(3, 0x00);
  display.writeDisplay();
}

void showTime(unsigned int ms) {
  ms = min(ms, (unsigned int)MAX_MS);
  int thousands = ms / 1000;
  int hundreds  = (ms % 1000) / 100;
  int tens      = (ms % 100)  / 10;
  int ones      = ms % 10;

  display.writeDigitNum(0, thousands);
  display.writeDigitNum(1, hundreds);
  display.writeDigitNum(2, tens);
  display.writeDigitNum(3, ones);
  display.writeDisplay();
}

void setup() {
  Serial.begin(9600);
  pinMode(BUZZER, OUTPUT);
  pinMode(BTN1, INPUT);
  strip.begin(); strip.show();
  Wire.begin();
  display.begin(0x70);
  display.setBrightness(10);
  randomSeed(analogRead(A5));
  Serial.println("Reaction Time Tester Ready!");
}

void loop() {
  clearDisplay();          // All 4 positions OFF before GO
  Serial.println("Get ready...");
  delay(random(2000, 5000));

  // Signal GO
  uint32_t color = strip.Color(
    random(50,255), random(50,255), random(50,255));
  for(int i = 0; i < 9; i++) strip.setPixelColor(i, color);
  strip.show();
  tone(BUZZER, 1000, 200);
  unsigned long startTime = millis();

  // Wait for button press
  while(digitalRead(BTN1) == LOW) {}
  unsigned long reaction = millis() - startTime;

  // Show result
  strip.clear(); strip.show();
  showTime((unsigned int)reaction);

  Serial.print("Reaction time: ");
  Serial.print(reaction);
  Serial.println(" ms");

  if(reaction < 200)      Serial.println("INCREDIBLE! Lightning fast!");
  else if(reaction < 350) Serial.println("Great reflexes!");
  else                     Serial.println("Keep practicing!");
```

```
  delay(3000);
}
```

ACTIVITY 02 Digital Thermometer & Humidity Monitor

Difficulty: Beginner	Est. Time: 10 minutes	Category: Interactive

Turn your board into a live weather station! The AHT20 sensor reads temperature and humidity every 2 seconds. The Serial Monitor display cycles between showing temperature and humidity, while the RGB LEDs change color based on how hot or cold it is — cool blue for cold, green for comfortable, and red for hot. A warning tone sounds if it gets too hot.

Pin & Wiring Reference

Arduino Pin	Component / Label	Direction	Notes
I2C (A4/A5)	AHT20 Temperature/Humidity	INPUT	Main sensor — no extra wiring
I2C (A4/A5)	Serial Monitor	OUTPUT	Shows temp and humidity
D3	RGB LED Strip	OUTPUT	Color = temperature range
D13	Buzzer	OUTPUT	Heat warning alarm
I2C	7 Segment Display	OUTPUT	Humidity Display

Wiring Notes

- All components are built into the shield — no external wiring required.
- The AHT20 use I2C (pins A4 and A5 on Uno).
- Make sure the shield is firmly seated on the Arduino.
- Open Serial Monitor at 9600 baud to see full readings.

How to Play / Use

6. Upload the code. The display will show 'tEMP' then cycle to the temperature.
7. Breathe on the AHT20 sensor to raise humidity — watch the serial monitor display change.
8. Touch the sensor gently to warm it slightly — LEDs shift from green to red.
9. If temperature exceeds 35°C the buzzer will sound a warning.
10. Serial Monitor shows detailed readings every 2 seconds.

Arduino Code

```
#include <Wire.h>
#include <Adafruit_NeoPixel.h>
#include "Adafruit_LEDBackpack.h"
#include "Adafruit_AHTX0.h"

#define LED_PIN  3
#define BUZZER   13
```

```
Adafruit_NeoPixel strip(9, LED_PIN, NEO_GRB + NEO_KHZ800);
Adafruit_7segment disp = Adafruit_7segment();
Adafruit_AHTX0 aht;

// ─────────────────────────────────────────
//  Show humidity on 7-segment
//  Format: XX.X H
//  Position 0 = tens digit
//  Position 1 = ones digit + decimal dot ON
//  Position 2 = tenths digit
//  Position 3 = 'H' raw segment 0x76
// ─────────────────────────────────────────
void showHumidity(float hum) {
  disp.clear();

  int whole      = (int)hum;
  int tenths     = (int)(hum * 10) % 10;
  int tens_digit = whole / 10;
  int ones_digit = whole % 10;

  // Position 0 — tens (blank if single digit)
  if (tens_digit > 0)
    disp.writeDigitNum(0, tens_digit, false);

  // Position 1 — ones with decimal dot ON
  disp.writeDigitNum(1, ones_digit, true);

  // Position 2 — tenths
  disp.writeDigitNum(2, tenths, false);

  // Position 3 — 'H' for Humidity
  disp.writeDigitRaw(3, 0x76);

  disp.writeDisplay();
}

void setup() {
  Serial.begin(9600);
  pinMode(BUZZER, OUTPUT);

  strip.begin();
  strip.setBrightness(50);
  strip.show();

  Wire.begin();
  disp.begin(0x70);
  disp.setBrightness(8);
  delay(100);

  // Blank display on boot
  disp.clear();
  disp.writeDisplay();

  if (!aht.begin()) {
    Serial.println("AHT sensor not found! Check wiring.");
    while (1) delay(100);
  }

  Serial.println("AHT sensor ready.");
}

void loop() {
  sensors_event_t hum, tmp;
  aht.getEvent(&hum, &tmp);
```

```
  float t = tmp.temperature;
  float h = hum.relative_humidity;

  // ── LED color based on temperature ──
  uint32_t c;
  if      (t < 18) c = strip.Color(0,   0,   200);  // cold   — blue
  else if (t < 26) c = strip.Color(0,   150, 0);    // normal — green
  else if (t < 35) c = strip.Color(200, 80,  0);    // warm   — orange
  else             c = strip.Color(220, 0,   0);    // hot    — red

  for (int i = 0; i < 9; i++)
    strip.setPixelColor(i, c);
  strip.show();

  // ── Buzzer if too hot ──
  if (t > 35) tone(BUZZER, 800);
  else        noTone(BUZZER);

  // ── Show humidity on 7-segment always ──
  showHumidity(h);

  // ── Serial Monitor ──
  Serial.print("Temp: ");
  Serial.print(t, 1);
  Serial.print(" C    Humidity: ");
  Serial.print(h, 1);
  Serial.println(" %");

  delay(500);
}
```

ACTIVITY 03 Simon Says Memory Game

Difficulty: Intermediate	Est. Time: 20 minutes	Category: Interactive

The classic memory game! The Arduino lights up LEDs in a random sequence and plays a corresponding tone for each. You must repeat the sequence by pressing BTN1–BTN3 (mapped to 3 LED groups). Each round adds one more step to the sequence. How many rounds can you remember? High score is stored and shown at game over.

Pin & Wiring Reference

Arduino Pin	Component / Label	Direction	Notes
D3	RGB LED Strip (NeoPixel)	OUTPUT	Three LED groups for 3 colors
D13	Buzzer	OUTPUT	Tone for each color
D8	BTN1 (S1)	INPUT	Select color 1 (Red group)
D7	BTN2 (S2)	INPUT	Select color 2 (Green group)
D12	BTN3 (S3)	INPUT	Select color 3 (Blue group)
I2C	Serial Monitor Display	OUTPUT	Shows current round / score
I2C	7 Segment Display	OUTPUT	Shows current round / score

Wiring Notes

- No external wiring — all components are on the shield.
- BTN1=S1 (pin D8), BTN2=S2 (pin D7), BTN3=S3 (pin D12).
- The 9 LEDs are split into 3 groups: LEDs 0-2 = Red, LEDs 3-5 = Green, LEDs 6-8 = Blue.
- Open Serial Monitor at 9600 baud for game instructions.

How to Play / Use

11. Upload code. Press any button to start a new game.
12. Watch the LED sequence carefully and listen to the tones.
13. Repeat the sequence using BTN1 (red), BTN2 (green), BTN3 (blue).
14. The sequence grows by 1 each round.
15. A wrong press causes game over — your score shows on the Serial Monitor display.

Arduino Code

```
#include <Wire.h>
#include <Adafruit_NeoPixel.h>
#include "Adafruit_LEDBackpack.h"
```

```
#define LED_PIN   3
#define BUZZER    13
#define BTN1      8
#define BTN2      7
#define BTN3      12
#define MAX_SEQ   20

Adafruit_NeoPixel strip(9, LED_PIN, NEO_GRB + NEO_KHZ800);
Adafruit_7segment disp = Adafruit_7segment();

int seq[MAX_SEQ];
int round_num = 0, highScore = 0;
int tones[]   = {440, 550, 660};
uint32_t colors[3];

// ─────────────────────────────────────────
//  Flash one LED group + play tone
// ─────────────────────────────────────────
void lightUp(int c, int ms) {
  int start = c * 3;
  for (int i = start; i < start + 3; i++)
    strip.setPixelColor(i, colors[c]);
  strip.show();
  tone(BUZZER, tones[c], ms);
  delay(ms + 50);
  for (int i = start; i < start + 3; i++)
    strip.setPixelColor(i, 0);
  strip.show();
  delay(150);
}

// ─────────────────────────────────────────
//  Blank display safely
// ─────────────────────────────────────────
void blankDisplay() {
  disp.clear();
  disp.writeDisplay();
}

// ─────────────────────────────────────────
//  Show score using positions 0,1,2,3
//  drawDot=false on all to kill colon dot
//  Leading zeros blanked cleanly
// ─────────────────────────────────────────
void showScore(int score) {
  disp.clear();

  int thousands =  score / 1000;
  int hundreds  = (score % 1000) / 100;
  int tens      = (score % 100)  / 10;
  int ones      =  score % 10;

  // Position 0 — thousands (blank if zero)
  if (thousands > 0)
    disp.writeDigitNum(0, thousands, false);

  // Position 1 — hundreds (blank if leading zeros)
  if (thousands > 0 || hundreds > 0)
    disp.writeDigitNum(1, hundreds, false);

  // Position 2 — tens (blank if score < 10)
  // false = dot OFF — this kills the colon half-glow
  if (thousands > 0 || hundreds > 0 || tens > 0)
    disp.writeDigitNum(2, tens, false);
```

```
  // Position 3 — ones ALWAYS shown
  disp.writeDigitNum(3, ones, false);

  disp.writeDisplay();
}

// ─────────────────────────────────────────
//  Game over
// ─────────────────────────────────────────
void gameOver() {
  if (round_num > highScore) highScore = round_num;

  // Flash all LEDs red x3
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 9; j++)
      strip.setPixelColor(j, strip.Color(200, 0, 0));
    strip.show();
    delay(200);
    strip.clear();
    strip.show();
    delay(200);
  }

  // Buzz
  tone(BUZZER, 200, 1000);
  delay(1000);

  // Serial Monitor
  Serial.print("Game Over! Score: ");
  Serial.println(round_num);
  Serial.print("High Score: ");
  Serial.println(highScore);

  // Show score on 7-segment
  showScore(round_num);

  // Hold for 3 seconds
  delay(3000);

  // Blank — stays off until next game over
  blankDisplay();

  // Reset
  round_num = 0;
}

// ─────────────────────────────────────────
//  Setup
// ─────────────────────────────────────────
void setup() {
  Serial.begin(9600);

  pinMode(BUZZER, OUTPUT);
  pinMode(BTN1,   INPUT);
  pinMode(BTN2,   INPUT);
  pinMode(BTN3,   INPUT);

  strip.begin();
  strip.show();

  Wire.begin();
  disp.begin(0x70);
  disp.setBrightness(8);
  delay(100);
```

```
  // Blank on boot — display OFF
  blankDisplay();

  colors[0] = strip.Color(200, 0, 0);
  colors[1] = strip.Color(0, 200, 0);
  colors[2] = strip.Color(0, 0, 200);

  randomSeed(analogRead(A5));

  Serial.println("Simon Says! Press any button to start.");
}

// ─────────────────────────────────────────
//  Main loop
// ─────────────────────────────────────────
void loop() {
  // Wait for any button press to start
  while (digitalRead(BTN1) == LOW &&
         digitalRead(BTN2) == LOW &&
         digitalRead(BTN3) == LOW) {}
  delay(500);

  // Add one new random step
  seq[round_num] = random(0, 3);
  round_num++;

  // Display stays BLANK during game

  // Play full sequence
  for (int i = 0; i < round_num; i++) {
    lightUp(seq[i], 600);
    delay(100);
  }

  // Read player input
  for (int i = 0; i < round_num; i++) {
    unsigned long t = millis();
    int pressed     = -1;

    while (millis() - t < 5000) {
      if (digitalRead(BTN1) == HIGH) { pressed = 0; break; }
      if (digitalRead(BTN2) == HIGH) { pressed = 1; break; }
      if (digitalRead(BTN3) == HIGH) { pressed = 2; break; }
    }

    if (pressed != seq[i]) {
      gameOver();
      return;
    }

    lightUp(pressed, 200);
    delay(100);
  }

  delay(600);
}
```

ACTIVITY 04 Joystick LED Controller

Difficulty: Beginner	Est. Time: 10 minutes	Category: Interactive

Use the joystick to paint with light! Moving the joystick left/right selects which of the 9 NeoPixel LEDs is active. Moving it up/down changes the brightness of the selected LED. Pressing the joystick button cycles through hue colors (red, orange, yellow, green, cyan, blue, violet). The potentiometer controls overall strip brightness. Great for learning analog input mapping.

Pin & Wiring Reference

Arduino Pin	Component / Label	Direction	Notes
A0	Joystick X-axis	INPUT	Select LED left/right
A1	Joystick Y-axis (or POT)	INPUT	Control brightness — Jumper R1
D3	RGB LED Strip	OUTPUT	9 NeoPixel LEDs
D6	Encoder/BTN4 (Joystick push)	INPUT	Cycle hue colors — check jumper

Wiring Notes

- Set jumper to Round 1 / Joystick side so A0=JoystickX, A1=JoystickY.
- If using POT for brightness instead, set jumper to Round 2 side.
- No external wiring — all on the shield.
- Joystick X: A0, Joystick Y: A1, LED strip: D3.

How to Play / Use

16. Upload code and move the joystick Up/Down — a cursor LED moves along the strip.
17. Open Serial Monitor at 9600 baud to see raw joystick values.

Arduino Code

```
#include <Adafruit_NeoPixel.h>

#define LED_PIN 3
#define JOY_X   A0
#define JOY_Y   A1
#define JOY_BTN 5

Adafruit_NeoPixel strip(9, LED_PIN, NEO_GRB + NEO_KHZ800);

int cursorLED = 0;
int brightness[9] = {50, 50, 50, 50, 50, 50, 50, 50, 50};
int hueIndex = 0;
```

```
uint32_t hues[] = {
  0xFF0000, 0xFF6600, 0xFFFF00, 0x00FF00,
  0x00FFFF, 0x0000FF, 0x8800FF, 0xFF00FF
};

// 🔥 Calibration values
int centerX = 512;
int centerY = 512;
int deadZone = 80;

unsigned long lastMove = 0;
unsigned long lastButton = 0;

void setup() {
  Serial.begin(9600);
  pinMode(JOY_BTN, INPUT_PULLUP);

  strip.begin();
  strip.setBrightness(80);
  strip.show();

  // 🧠 AUTO CALIBRATION (IMPORTANT)
  delay(1000); // keep joystick untouched!
  centerX = analogRead(JOY_X);
  centerY = analogRead(JOY_Y);

  Serial.print("Center X: "); Serial.println(centerX);
  Serial.print("Center Y: "); Serial.println(centerY);
}

void loop() {
  int jx = analogRead(JOY_X);
  int jy = analogRead(JOY_Y);

  // 🎯 Apply calibrated dead zone
  bool left  = (jx < centerX - deadZone);
  bool right = (jx > centerX + deadZone);
  bool up    = (jy < centerY - deadZone);
  bool down  = (jy > centerY + deadZone);

 // ⏱ Smooth movement
  if (millis() - lastMove > 150) {
    if (left && cursorLED > 0) {
      cursorLED--;
      lastMove = millis();
    }
    if (right && cursorLED < 8) {
      cursorLED++;
      lastMove = millis();
    }

    if (up) {
      brightness[cursorLED] = min(255, brightness[cursorLED] + 10);
      lastMove = millis();
    }
    if (down) {
      brightness[cursorLED] = max(0, brightness[cursorLED] - 10);
      lastMove = millis();
    }
  }
```

```
  // 🔘 Button debounce
  if (digitalRead(JOY_BTN) == LOW && millis() - lastButton > 300) {
    hueIndex = (hueIndex + 1) % 8;
    lastButton = millis();
  }

  // 🌈 Render LEDs
  for (int i = 0; i < 9; i++) {
    if (i == cursorLED) {
      strip.setPixelColor(i, strip.Color(120, 120, 120));
    } else {
      uint32_t h = hues[hueIndex];
      uint8_t R = ((h >> 16) & 0xFF) * brightness[i] / 255;
      uint8_t G = ((h >> 8) & 0xFF) * brightness[i] / 255;
      uint8_t B = (h & 0xFF) * brightness[i] / 255;
      strip.setPixelColor(i, strip.Color(R, G, B));
    }
  }

  strip.show();

  // Debug
  Serial.print("X:"); Serial.print(jx);
  Serial.print(" Y:"); Serial.print(jy);
  Serial.print(" Cursor:"); Serial.println(cursorLED);
}
```

ACTIVITY 05 Intruder Alarm with LDR Sensor

LDR Detection

Difficulty: Beginner	Est. Time: 15 minutes	Category: Interactive

Build a security alarm using the LDR Sensor ! When an light occurs at the LDR sensor the buzzer sounds an alarm pattern, all LEDs flash red, and an alert is printed to Serial Monitor with a timestamp (seconds since boot). Press BTN1 to silence the alarm. The LDR adds a light-level check — alarm only triggers in High Light Mode

Pin & Wiring Reference

Arduino Pin	Component / Label	Direction	Notes
A3	LDR (Light Sensor)	INPUT	Light Detection
D13	Buzzer	OUTPUT	Alarm sound
D3	RGB LED Strip	OUTPUT	Red flash on alarm
D8	BTN1 (S1)	INPUT	Silence / reset alarm

Wiring Notes

- All sensors and outputs are on the shield — no external wiring.
- IR sensor is on pin A2, LDR on pin A3.
- Point the IR sensor toward the area you want to monitor.
- Adjust the LDR threshold in code if needed (default: < 300 = dark).

How to Play / Use

18. Upload the code. Serial Monitor shows 'Alarm Armed' — system is watching.
19. Put some Light on the LDR , Alarm starts to buzz, INTRUDER ALERT shows on Serial Monitor.
20. Press BTN1 (S1) to silence and re-arm the alarm.

Arduino Code

```
#include <Adafruit_NeoPixel.h>

#define IR_PIN  A2
#define LDR     A3
#define BUZZER  13
#define LED_PIN 3
#define BTN1    8

Adafruit_NeoPixel strip(9, LED_PIN, NEO_GRB + NEO_KHZ800);
bool alarming = false;
```

```
void setup() {
  Serial.begin(9600);
  pinMode(IR_PIN, INPUT);
  pinMode(BUZZER, OUTPUT);
  pinMode(BTN1, INPUT);
  strip.begin(); strip.show();
  Serial.println("=== INTRUDER ALARM ARMED ===");
}

void triggerAlarm() {
  alarming = true;
  Serial.print("!! INTRUDER DETECTED at T+");
  Serial.print(millis()/1000); Serial.println("s !!");
}

void loop() {
  int light = analogRead(LDR);
  bool nightMode = (light < 300);
  bool intruder  = (digitalRead(IR_PIN) == LOW);

  if(nightMode && intruder && !alarming) triggerAlarm();

  if(alarming) {
    // Flash red LEDs
    for(int i=0;i<9;i++) strip.setPixelColor(i, strip.Color(255,0,0));
    strip.show(); tone(BUZZER,800); delay(200);
    strip.clear(); strip.show(); noTone(BUZZER); delay(200);
    // Silence button
    if(digitalRead(BTN1)==HIGH) {
      alarming = false;
      noTone(BUZZER);
      strip.clear(); strip.show();
      Serial.println("Alarm silenced. Re-arming...");
      delay(2000);
    }
  } else {
    // Heartbeat — single green LED pulse
    strip.setPixelColor(0, strip.Color(0,20,0));
    strip.show(); delay(100);
    strip.clear(); strip.show();
    Serial.print("Armed | Light: "); Serial.print(light);
    Serial.print(" | Night mode: "); Serial.println(nightMode?"ON":"OFF");
    delay(900);
  }
}
```

ACTIVITY 06 Rotary Encoder Volume Knob & LED VU Meter

Difficulty: Intermediate	Est. Time: 20 minutes	Category: Interactive

Spin the rotary encoder to control a virtual volume level from 0 to 9. The NeoPixel strip acts as a colorful VU meter bar — green at low levels, transitioning through yellow to red at maximum. Press the encoder button to mute/unmute. The buzzer plays a click sound on each encoder tick.

Pin & Wiring Reference

Arduino Pin	Component / Label	Direction	Notes
D6	Encoder CLK	INPUT	Rotation clock signal
D10	Encoder DT	INPUT	Rotation direction signal
D5	Encoder SW	INPUT	Push button — mute toggle
D3	RGB LED Strip	OUTPUT	VU meter bar display
D13	Buzzer	OUTPUT	Click on each tick

Wiring Notes

- Set jumper to Round 1 (Joystick/Encoder) side for correct CLK and DT routing.
- Encoder CLK = D6, Encoder DT = D10, Encoder button = D5.
- All other components use shield connections — no extra wiring.

How to Play / Use

21. Upload and rotate the encoder clockwise to increase volume (0–9).
22. The LED bar fills up from left to right — green, yellow, then red.
23. Press the encoder button to mute — all LEDs go dark
24. Press again to unmute and restore the previous level.

Arduino Code

```
#include <Wire.h>
#include <Adafruit_NeoPixel.h>
#include "Adafruit_LEDBackpack.h"

#define ENC_CLK 6
#define ENC_DT  10
#define ENC_SW  5
#define LED_PIN 3
#define BUZZER  13

Adafruit_NeoPixel strip(9, LED_PIN, NEO_GRB + NEO_KHZ800);
Adafruit_7segment disp = Adafruit_7segment();
```

```
int volume = 5, lastClk = HIGH;
bool muted = false;

uint32_t vuColor(int i, int vol) {
  if(i >= vol) return strip.Color(0,0,0);
  if(i < 4) return strip.Color(0,200,0);
  if(i < 7) return strip.Color(200,200,0);
  return strip.Color(255,0,0);
}

void updateDisplay() {
  int v = muted ? 0 : volume;
  disp.clear(); disp.print(v); disp.writeDisplay();
  for(int i=0;i<9;i++) strip.setPixelColor(i, muted ? 0 : vuColor(i,v));
  strip.show();
}

void setup() {
  Serial.begin(9600);
  pinMode(ENC_CLK, INPUT);
  pinMode(ENC_DT,  INPUT);
  pinMode(ENC_SW,  INPUT_PULLUP);
  pinMode(BUZZER,  OUTPUT);
  strip.begin(); strip.show();
  Wire.begin(); disp.begin(0x70); disp.setBrightness(8);
  updateDisplay();
  Serial.println("Volume Knob Active. Turn encoder.");
}

void loop() {
  int clk = digitalRead(ENC_CLK);
  if(clk != lastClk && clk == LOW) {
    if(digitalRead(ENC_DT) != clk) {
      if(volume < 9) volume++;
    } else {
      if(volume > 0) volume--;
    }
    tone(BUZZER, 1200, 30);
    updateDisplay();
    Serial.print("Volume: "); Serial.println(muted?"MUTED":String(volume));
  }
  lastClk = clk;

  if(digitalRead(ENC_SW)==LOW) {
    muted = !muted;
    tone(BUZZER, muted?400:800, 150);
    updateDisplay();
    Serial.println(muted ? "MUTED" : "UNMUTED");
    delay(400);
  }
}
```

ACTIVITY 07 Sound-Reactive Light Show

Difficulty: Intermediate	Est. Time: 15 minutes	Category: Interactive

The microphone listens to your music, claps, or voice, and the RGB LEDs dance in real time! Louder sounds produce more energetic, brighter LED patterns. The display shows the current sound level (0–100). The potentiometer controls the sensitivity. A quiet hum produces slow gentle waves; loud music triggers rapid rainbow bursts.

Pin & Wiring Reference

Arduino Pin	Component / Label	Direction	Notes
A0	Microphone	INPUT	Sound level input — Jumper R2
A1	Potentiometer	INPUT	Sensitivity control — Jumper R2
D3	RGB LED Strip	OUTPUT	Sound-reactive light patterns

Wiring Notes

- Set jumper to Round 2 (POT/Motor/Mic) side — A0 = Mic, A1 = POT.
- Place the Arduino near your speaker or in a room with music.
- No extra wiring needed.
- Open Serial Monitor at 9600 baud to see live dB levels.

How to Play / Use

25. Upload the code and play music near the board.
26. LEDs will pulse and change color with the beat.
27. Turn the potentiometer clockwise to increase sensitivity.
28. Clap your hands or snap fingers — watch the LEDs react instantly.

Arduino Code

```
#include <Wire.h>
#include <Adafruit_NeoPixel.h>
#include "Adafruit_LEDBackpack.h"

#define MIC_PIN A0
#define POT_PIN A1
#define LED_PIN 3

Adafruit_NeoPixel strip(9, LED_PIN, NEO_GRB + NEO_KHZ800);
Adafruit_7segment disp = Adafruit_7segment();

int baseline = 512;
```

```
uint8_t hue = 0;

int smoothedLevel = 0;

// 🎤 Sample microphone input level
int sampleSound(int samples) {
  long total = 0;
  for (int i = 0; i < samples; i++) {
    int v = analogRead(MIC_PIN) - baseline;
    total += abs(v);
    delayMicroseconds(150);
  }
  return total / samples;
}

void setup() {
  Serial.begin(9600);

  // 🌈 Initialize NeoPixel LED strip
  strip.begin();
  strip.setBrightness(80);
  strip.show();

  // 🔴 Force display OFF for performance
  Wire.begin();
  disp.begin(0x70);
  disp.setBrightness(0);
  disp.clear();
  disp.writeDisplay();

  // 🔧 Calibrate microphone baseline
  long sum = 0;
  for (int i = 0; i < 300; i++) {
    sum += analogRead(MIC_PIN);
    delay(3);
  }
  baseline = sum / 300;

  Serial.print("Baseline: ");
  Serial.println(baseline);
}

void loop() {

  // 🔴 Force display OFF for accurate sampling
  disp.clear();
  disp.writeDisplay();

  // 🎚️ Adjust sensitivity with potentiometer
  int sens = map(analogRead(POT_PIN), 0, 1023, 1, 10);

// 🎤 Capture current sound intensity
  int rawLevel = sampleSound(40) * sens;

  int mapped = constrain(map(rawLevel, 0, 400, 0, 100), 0, 100);

  // 🔥 Smooth output for stable visuals
  smoothedLevel = (smoothedLevel * 7 + mapped) / 8;

  // 🌈 Draw the LED bar graph
  int numLit = map(smoothedLevel, 0, 100, 0, 9);
```

```
  for (int i = 0; i < 9; i++) {
    if (i < numLit) {
      strip.setPixelColor(i, strip.ColorHSV(
        (hue + i * 3000) % 65536,
        255,
        constrain(smoothedLevel * 2 + 50, 50, 255)
      ));
    } else {
      strip.setPixelColor(i, 0);
    }
  }

  strip.show();

// 🌈 Rotate through color hues
  hue += smoothedLevel / 4 + 1;

  // 📊 Output data to Serial Monitor
  Serial.print("Level: ");
  Serial.print(smoothedLevel);
  Serial.print("  Sens: ");
  Serial.println(sens);

  delay(10);
}
```

ACTIVITY 08 Countdown Timer with Buzzer Alert

Difficulty: Beginner	Est. Time: 15 minutes	Category: Interactive

A fully functional countdown timer! Use BTN1 to add minutes, BTN2 to add seconds, and BTN3 to start/pause the timer. The 7-segment display shows the remaining time in MM:SS format (using the colon). As time runs low (under 10 seconds) the LEDs flash red and the buzzer beeps faster. At zero a continuous alarm sounds. The encoder knob can also adjust the time.

Pin & Wiring Reference

Arduino Pin	Component / Label	Direction	Notes
D8	BTN1 (S1)	INPUT	Add 1 minute to timer
D7	BTN2 (S2)	INPUT	Add 10 seconds to timer
D12	BTN3 (S3)	INPUT	Start / Pause / Reset
D13	Buzzer	OUTPUT	Countdown beeps + alarm
D3	RGB LED Strip	OUTPUT	Urgency indicator
I2C	7-Segment Display	OUTPUT	MM:SS countdown display

Wiring Notes

- All components are on the shield — no extra wiring required.
- Buttons: BTN1=D8, BTN2=D7, BTN3=D12.
- Jumper position does not matter for this activity.
- Connect USB for power and Serial Monitor (9600 baud).

How to Play / Use

29. Upload the code. Display shows 00:00.
30. Press BTN1 to add minutes (each press = +1 min). Press BTN2 for +10 seconds.
31. Press BTN3 to START the countdown.
32. Press BTN3 again to PAUSE. Press once more to RESUME.
33. Hold BTN3 for 2 seconds to RESET the timer back to 00:00.

Arduino Code

```
#include <Wire.h>
#include <Adafruit_NeoPixel.h>
#include "Adafruit_LEDBackpack.h"

#define BTN1 8
#define BTN2 7
#define BTN3 12
#define BUZZER 13
#define LED_PIN 3

Adafruit_NeoPixel strip(9, LED_PIN, NEO_GRB + NEO_KHZ800);
Adafruit_7segment disp = Adafruit_7segment();

int totalSecs = 0;
bool running = false;
unsigned long lastTick = 0;

void showTime(int s) {
  int m = s / 60; s = s % 60;
  disp.clear();
  disp.writeDigitNum(0, m/10);
  disp.writeDigitNum(1, m%10);
  disp.drawColon(true);
  disp.writeDigitNum(3, s/10);
  disp.writeDigitNum(4, s%10);
  disp.writeDisplay();
}

void setup() {
  Serial.begin(9600);
  pinMode(BTN1,INPUT); pinMode(BTN2,INPUT); pinMode(BTN3,INPUT);
  pinMode(BUZZER,OUTPUT);
  strip.begin(); strip.show();
  Wire.begin(); disp.begin(0x70); disp.setBrightness(10);
  showTime(0);
  Serial.println("Countdown Timer Ready");
}

void loop() {
  // BTN1 = +1 min
  if(!running && digitalRead(BTN1)==HIGH) { totalSecs+=60; showTime(totalSecs);
delay(300); }
  // BTN2 = +10 sec
  if(!running && digitalRead(BTN2)==HIGH) { totalSecs+=10; showTime(totalSecs);
delay(300); }
  // BTN3 = Start/Pause/Reset
  if(digitalRead(BTN3)==HIGH) {
    unsigned long held = millis();
    while(digitalRead(BTN3)==HIGH) {}
    if(millis()-held > 2000) { totalSecs=0; running=false; showTime(0); }
    else { running=!running; }
    delay(200);
  }

  if(running && millis()-lastTick >= 1000) {
    lastTick = millis();
    if(totalSecs > 0) {
      totalSecs--;
      showTime(totalSecs);
      if(totalSecs <= 10) {
        tone(BUZZER, 1000+totalSecs*50, 100);
        for(int i=0;i<9;i++) strip.setPixelColor(i, strip.Color(200,0,0));
      } else {
        for(int i=0;i<9;i++) strip.setPixelColor(i, strip.Color(0,100,0));
      }
```

```
      strip.show();
      Serial.println(totalSecs);
    } else {
      running = false;
      Serial.println("TIME UP!");
      for(int r=0;r<5;r++){
        tone(BUZZER,800,400);
        for(int i=0;i<9;i++) strip.setPixelColor(i, strip.Color(255,0,0));
        strip.show(); delay(400);
        strip.clear(); strip.show(); delay(200);
      }
    }
  }
}
```

ACTIVITY 09 Fire Detection & Smart Fan Controller

Difficulty: Intermediate	Est. Time: 20 minutes	Category: Interactive

A real-world safety application combining multiple sensors. The flame sensor watches for fire. The AHT20 monitors temperature. If either detects danger (flame detected OR temperature > 35C), the DC motor spins as a fan to cool the area, LEDs flash orange as a warning, and the buzzer sounds an alert.. Normal conditions show a steady green status.

Pin & Wiring Reference

Arduino Pin	Component / Label	Direction	Notes
D2	Flame Sensor	INPUT	Detects open flame
I2C	AHT20 Temp Sensor	INPUT	Monitors temperature
D11	Motor IN1	OUTPUT	DC fan motor — Jumper R2
D4	Motor IN2 (TOUCH/MOTOR)	OUTPUT	DC fan motor — Jumper R2
D13	Buzzer	OUTPUT	Fire alarm
D3	RGB LED Strip	OUTPUT	Status and warning LEDs

Wiring Notes

- Set jumper to Round 2 (Motor/POT side) so D4 = Motor IN2.
- Connect a small DC fan or motor to the Motor terminals on the shield.
- Flame sensor is built in on D2 — no extra wiring.
- AHT20 use I2C (built-in on shield).

How to Play / Use

34. Upload the code. Green LEDs and steady temp reading = all clear.
35. Hold a lighter near the flame sensor (carefully!) — motor starts, alarm sounds.
36. Breathe warm air on the AHT20 — if temperature exceeds 35C, motor activates.
37. Remove the heat/flame source — system automatically disarms after 5 seconds.
38. Serial Monitor logs all events with timestamps.

Arduino Code

```
#include <Wire.h>
#include <Adafruit_NeoPixel.h>
#include "Adafruit_LEDBackpack.h"
#include "Adafruit_AHTX0.h"

#define FLAME     2
```

```
#define MOTOR_IN1 11
#define MOTOR_IN2 4
#define BUZZER    13
#define LED_PIN   3

Adafruit_NeoPixel strip(9, LED_PIN, NEO_GRB + NEO_KHZ800);
Adafruit_7segment disp = Adafruit_7segment();
Adafruit_AHTX0    aht;

bool alertActive = false;
unsigned long alertClear = 0;

void setup() {
  Serial.begin(9600);
  pinMode(FLAME, INPUT);
  pinMode(MOTOR_IN1, OUTPUT); pinMode(MOTOR_IN2, OUTPUT);
  pinMode(BUZZER, OUTPUT);
  strip.begin(); strip.show();
  Wire.begin(); disp.begin(0x70); disp.setBrightness(8);
  aht.begin();
  Serial.println("Fire Detection System Active");
}

void runFan(bool on) {
  digitalWrite(MOTOR_IN1, on ? HIGH : LOW);
  digitalWrite(MOTOR_IN2, LOW);
}

void loop() {
  sensors_event_t hum, tmp;
  aht.getEvent(&hum, &tmp);
  float temp = tmp.temperature;
  bool flame = (digitalRead(FLAME) == LOW);
  bool hotTemp = (temp > 35);

  disp.clear(); disp.print((int)temp); disp.writeDisplay();

  if(flame || hotTemp) {
    alertActive = true; alertClear = millis() + 5000;
    Serial.print("ALERT! Flame="); Serial.print(flame);
    Serial.print(" Temp="); Serial.println(temp);
  }
  if(alertActive && millis() > alertClear) alertActive = false;

  if(alertActive) {
    runFan(true);
    tone(BUZZER, 800);
    for(int i=0;i<9;i++) strip.setPixelColor(i,
      (millis()/200)%2 ? strip.Color(255,80,0) : 0);
    strip.show();
  } else {
    runFan(false); noTone(BUZZER);
    for(int i=0;i<9;i++) strip.setPixelColor(i, strip.Color(0,60,0));
    strip.show();
  }
  delay(200);
}
```

ACTIVITY 10 Digital Piano with Recorder

Difficulty: Advanced	Est. Time: 30 minutes	Category: Interactive

Play a 4-octave digital piano using the 4 buttons and the rotary encoder! BTN1–BTN4 play 4 notes in the current octave. The encoder selects the octave (1–4). The buzzer plays each note. LEDs light up to show which note is pressed in a colorful pattern. Press and hold BTN3+BTN4 together to START recording — up to 32 notes. Play a sequence, then release both buttons to play it back automatically. The 7-segment shows octave and playback status.

Pin & Wiring Reference

Arduino Pin	Component / Label	Direction	Notes
D8	BTN1 (S1)	INPUT	Play note 1 (C)
D7	BTN2 (S2)	INPUT	Play note 2 (E)
D12	BTN3 (S3)	INPUT	Play note 3 (G)
D6	BTN4/Enc CLK (S4)	INPUT	Play note 4 (B) — Jumper R2
D10	Encoder DT	INPUT	Octave selector turn
D5	Encoder SW	INPUT	Octave selector push
D13	Buzzer	OUTPUT	Piano note output
D3	RGB LED Strip	OUTPUT	Note color visualizer
I2C	7-Segment Display	OUTPUT	Octave + status display

Wiring Notes

- Set jumper to Round 2 side so BTN4 is accessible on D6.
- Or leave jumper on R1 side and use the Encoder CLK pin (D6) as BTN4.
- All other connections are standard shield pins — no external wiring.
- Connect headphones or small speaker in series with the buzzer for better sound.

How to Play / Use

39. Upload code. Display shows 'O 1' (Octave 1).
40. Press BTN1–BTN4 to play C, E, G, B in octave 1.
41. Rotate encoder to change octave (1–4). Higher octave = higher pitch.
42. Press encoder button to hear a scale demo.
43. Hold BTN3+BTN4 together to record. Play notes. Release both to play back.

Arduino Code

```
#include <Wire.h>
#include <Adafruit_NeoPixel.h>
#include "Adafruit_LEDBackpack.h"

#define BTN1 8
#define BTN2 7
#define BTN3 12
#define BTN4 6
#define ENC_DT 10
#define ENC_SW 5
#define BUZZER 13
#define LED_PIN 3

Adafruit_NeoPixel strip(9, LED_PIN, NEO_GRB + NEO_KHZ800);
Adafruit_7segment disp = Adafruit_7segment();

// Notes: C E G B in 4 octaves
int notes[4][4] = {
  {262,330,392,494},  // Octave 1
  {523,659,784,988},  // Octave 2
  {1047,1319,1568,1976}, // Octave 3
  {2093,2637,3136,3951}  // Octave 4
};
uint32_t noteColors[] = {
  0xFF0000, 0x00FF00, 0x0000FF, 0xFFFF00
};

int octave = 0;
int lastClk = HIGH;
int recBuf[32];
int recLen = 0;

void playNote(int btn) {
  int freq = notes[octave][btn];
  int ledsOn = (btn+1)*2;
  for(int i=0;i<9;i++)
    strip.setPixelColor(i, i<ledsOn ? noteColors[btn] : 0);
  strip.show();
  tone(BUZZER, freq, 200);
  delay(220);
  strip.clear(); strip.show();
}

void showOctave() {
  disp.clear();
  disp.writeDigitRaw(0, 0x3F); // O shape
  disp.writeDigitNum(1, octave+1);
  disp.writeDisplay();
}

void setup() {
  Serial.begin(9600);
  pinMode(BTN1,INPUT); pinMode(BTN2,INPUT);
  pinMode(BTN3,INPUT); pinMode(BTN4,INPUT);
  pinMode(ENC_DT,INPUT); pinMode(ENC_SW,INPUT_PULLUP);
  pinMode(BUZZER,OUTPUT);
  strip.begin(); strip.show();
  Wire.begin(); disp.begin(0x70); disp.setBrightness(8);
  showOctave();
  Serial.println("Digital Piano Ready! BTN1-4 = C E G B");
}

void loop() {
  // Encoder = change octave
  int clk = digitalRead(BTN4); // BTN4 pin = ENC_CLK on R1
```

```
  if(clk != lastClk && clk == LOW) {
    if(digitalRead(ENC_DT) != clk) octave = min(3, octave+1);
    else octave = max(0, octave-1);
    showOctave();
    Serial.print("Octave: "); Serial.println(octave+1);
  }
  lastClk = clk;

  // Record mode: BTN3+BTN4 held
  if(digitalRead(BTN3)==HIGH && digitalRead(BTN4)==HIGH) {
    recLen = 0;
    disp.print(8888); disp.writeDisplay(); // blink to show recording
    delay(500);
    while(digitalRead(BTN3)==HIGH || digitalRead(BTN4)==HIGH) {
      if(digitalRead(BTN1)==HIGH && recLen<32){recBuf[recLen++]=0; playNote(0);}
      if(digitalRead(BTN2)==HIGH && recLen<32){recBuf[recLen++]=1; playNote(1);}
    }
    // Playback
    Serial.print("Playing back "); Serial.print(recLen); Serial.println(" notes");
    for(int i=0;i<recLen;i++) { playNote(recBuf[i]); delay(100); }
    showOctave();
    return;
  }

  // Normal play
  if(digitalRead(BTN1)==HIGH) playNote(0);
  if(digitalRead(BTN2)==HIGH) playNote(1);
  if(digitalRead(BTN3)==HIGH) playNote(2);

  // Encoder push = play scale demo
  if(digitalRead(ENC_SW)==LOW) {
    for(int i=0;i<4;i++) { playNote(i); delay(100); }
    delay(300);
  }
}
```

ACTIVITY 11 Stopwatch with Start/Pause/Reset

Difficulty: Beginner	Est. Time: 15 minutes	Category: Interactive

A precision stopwatch built with two buttons! BTN1 (S1) starts and pauses the timer, while BTN2 (S2) resets it to zero. The 7-segment display shows elapsed time in centiseconds (0.01 s units), updating every 100 ms. Time is tracked using millis() for accuracy up to 99.99 seconds. Great for timing experiments, reaction tests, or learning how millis()-based timers work in Arduino.

Pin & Wiring Reference

Arduino Pin	Component / Label	Direction	Notes
D8	BTN1 (S1)	INPUT	Start / Pause toggle
D7	BTN2 (S2)	INPUT	Reset — stops and clears timer
I2C (A4/A5)	7-Segment Display	OUTPUT	Shows elapsed time (centiseconds)

Wiring Notes

- No external wiring needed — all components are on the shield.
- Plug the shield onto your Arduino Uno and connect USB to your PC.
- Jumper settings: No special jumper position required for this activity.
- Display shows centiseconds (0.01 s units) up to a maximum of 99.99 seconds.

How to Play / Use

44. Upload the code. The display will show 0000 (all zeros).
45. Press BTN1 (S1) to START the stopwatch — the display begins counting up.
46. Press BTN1 again to PAUSE — the timer freezes at the current value.
47. Press BTN1 once more to RESUME from where you paused.
48. Press BTN2 (S2) at any time to RESET — timer stops and display clears to 0000.

Arduino Code

```
#include <Wire.h>
#include "Adafruit_LEDBackpack.h"
#define BTN1 8   // Start / Pause
#define BTN2 7   // Reset
Adafruit_7segment display = Adafruit_7segment();
bool running = false;
unsigned long startTime = 0;
unsigned long elapsed = 0;
unsigned long lastDisplayUpdate = 0;

void setup() {
  pinMode(BTN1, INPUT);
```

```
  pinMode(BTN2, INPUT);
  Wire.begin();
  display.begin(0x70);
  display.setBrightness(5);
}

void loop() {
  // START / PAUSE
  if (digitalRead(BTN1) == HIGH) {
    delay(200);
    running = !running;
    if (running) {
      startTime = millis() - elapsed;
    } else {
      elapsed = millis() - startTime;
    }
  }
  // RESET
  if (digitalRead(BTN2) == HIGH) {
    delay(200);
    running = false;
    elapsed = 0;
    display.clear();
    display.writeDisplay();
  }
  // TIME UPDATE
  if (running) {
    elapsed = millis() - startTime;
  }
  // DISPLAY UPDATE (every 100 ms)
  if (millis() - lastDisplayUpdate > 100) {
    lastDisplayUpdate = millis();
    int value = elapsed / 10;
    if (value > 9999) value = 9999;
    int thousands = (value / 1000) % 10;
    int hundreds  = (value / 100) % 10;
    int tens      = (value / 10) % 10;
    int ones      = value % 10;
    display.writeDigitNum(0, thousands);
    display.writeDigitNum(1, hundreds);
    display.writeDigitNum(2, tens);
    display.writeDigitNum(3, ones);
    display.writeDisplay();
  }
  // Other sensors safe here
}
```

Primary Code — Automated Test Suite

This is the complete automated test suite sketch for the SmartElex Multi-Function Shield. Upload it to your Arduino Uno, open the Serial Monitor at 115200 baud, and follow the on-screen prompts to verify every component on the board.

Full Sketch

```
/*
 * ============================================================
 *  SmartElex Multi-Function Shield — AUTOMATED TEST SUITE
 * ============================================================
 *
 *  COMMANDS (type in Serial Monitor at 115200 baud, press Enter):
 *
 *    R1      → Start Round 1  (Joystick / Encoder / Buzzer side)
 *    R2      → Start Round 2  (POT / Motor / Mic side)
 *    STOP    → Stop current test
 *    STATUS  → Show PASS/FAIL status of every test so far
 *    LIST    → Show all test numbers for quick reference
 *    HELP    → Reprint this command list
 *
 *  RETRY any single test by typing its number + letter:
 *    1A … 13A  → Retry Round-1 test by number
 *    1B …  5B  → Retry Round-2 test by number
 *
 *  Round 1 numbers:
 *    1A  Joystick X/Y     2A  IR Sensor      3A  Buzzer
 *    4A  Rotary Encoder   5A  BTN1 (S1)      6A  BTN2 (S2)
 *    7A  BTN3 (S3)        8A  Flame Sensor   9A  Touch Sensor
 *   10A  AHT20           11A  LDR           12A  7-Segment
 *   13A  RGB LED
 *
 *  Round 2 numbers:
 *    1B  Potentiometer    2B  BTN4 (S4)      3B  Microphone
 *    4B  DC Motor         5B  Status LED
 * ============================================================
 */

#include <Wire.h>
#include <Adafruit_NeoPixel.h>
#include <Adafruit_GFX.h>
#include "Adafruit_LEDBackpack.h"
#include "Adafruit_AHTX0.h"

// ─── Pin Definitions ──────────────────────────────────────
#define JOY_X      A0
#define JOY_Y      A1
#define IR_PIN     A2
#define BUZZER     13
#define ENC_CLK    6
#define ENC_DT     10
#define ENC_SW     5
#define BTN1       8
#define BTN2       7
#define BTN3       12
#define BTN4       6     // shared with ENC_CLK  — Round 2 only
#define FLAME      2
#define TOUCH      4     // shared with MOTOR_IN2
#define MIC        A0    // shared with JOY_X    — Round 2 only
#define MOTOR_IN1  11
```

```
#define MOTOR_IN2  4     // shared with TOUCH    — Round 2 only
#define LED_PIN    3
#define POT        A1    // shared with JOY_Y    — Round 2 only
#define LDR        A3
#define STATUS_LED 9

// ─── Objects ──────────────────────────────────────────────
Adafruit_NeoPixel strip(9, LED_PIN, NEO_GRB + NEO_KHZ800);
Adafruit_7segment seg = Adafruit_7segment();
Adafruit_AHTX0    aht;

// ─── Round / state machine ────────────────────────────────
// IDLE    = nothing running
// ROUND1  = full Round-1 sequence running
// ROUND2  = full Round-2 sequence running
// SINGLE  = one retry test running (uses retryRound to know which array)
enum RoundMode { IDLE, ROUND1, ROUND2, SINGLE };

RoundMode activeRound = IDLE;
RoundMode retryRound  = IDLE;  // set when activeRound == SINGLE

int           testIndex   = 0;
bool          testRunning = false;
unsigned long slotStart   = 0;

// encoder state
int  lastClkState = HIGH;
int  encCount     = 0;

// motor state
bool          motorFwd      = true;
unsigned long motorToggleAt = 0;

// ─── Test slot structure ──────────────────────────────────
struct TestSlot {
  const char*   name;
  unsigned long durationMs;
  bool          passed;
  bool          checked;
};

// ── Round 1 slots ──
TestSlot r1[] = {
  { "Joystick X/Y",    6000,  false, false },  // index 0  → cmd 1A
  { "IR Sensor",       5000,  false, false },  // index 1  → cmd 2A
  { "Buzzer",          4000,  false, false },  // index 2  → cmd 3A
  { "Rotary Encoder",  8000,  false, false },  // index 3  → cmd 4A
  { "BTN1 (S1)",       5000,  false, false },  // index 4  → cmd 5A
  { "BTN2 (S2)",       5000,  false, false },  // index 5  → cmd 6A
  { "BTN3 (S3)",       5000,  false, false },  // index 6  → cmd 7A
  { "Flame Sensor",   15000,  false, false },  // index 7  → cmd 8A
  { "Touch Sensor",    6000,  false, false },  // index 8  → cmd 9A
  { "AHT20 Temp/Hum",  9000,  false, false },  // index 9  → cmd 10A
  { "LDR",             5000,  false, false },  // index 10 → cmd 11A
  { "7-Segment",       5000,  false, false },  // index 11 → cmd 12A
  { "RGB LED",         8000,  false, false },  // index 12 → cmd 13A
};
const int R1_COUNT = sizeof(r1) / sizeof(r1[0]);   // 13

// ── Round 2 slots ──
TestSlot r2[] = {
  { "Potentiometer",   6000,  false, false },  // index 0 → cmd 1B
  { "BTN4 (S4)",       5000,  false, false },  // index 1 → cmd 2B
  { "Microphone",      5000,  false, false },  // index 2 → cmd 3B
  { "DC Motor",       12000,  false, false },  // index 3 → cmd 4B
```

```
  { "Status LED",      4000,  false, false },  // index 4 → cmd 5B
};
const int R2_COUNT = sizeof(r2) / sizeof(r2[0]);   // 5

// ─── Forward declarations ──────────────────────────────
// (Arduino's auto-prototype only works for simple signatures;
//  overloads and RoundMode params need explicit declarations.)
void beginSlot   (int idx, RoundMode r);
void runSlotTick (int idx, RoundMode r);
void finalizeSlot(int idx, RoundMode r);
void printRoundSummary(RoundMode r);
void runR1Tick(int idx);
void runR2Tick(int idx);
void printStatus();
void printList();
void printCommandMenu();
void stopRound();
void stopAllOutputs();

// ─── Helpers ───────────────────────────────────────────
void printSeparator() {
  Serial.println(F("------------------------------------------"));
}

void printHeader(const char* title) {
  printSeparator();
  Serial.print(F(">>> "));
  Serial.println(title);
  printSeparator();
}

void verdictLine(const char* name, bool pass) {
  Serial.print(F("  ["));
  Serial.print(pass ? F("PASS") : F("FAIL"));
  Serial.print(F("] "));
  Serial.println(name);
}

void stopAllOutputs() {
  digitalWrite(BUZZER, LOW);
  digitalWrite(MOTOR_IN1, LOW);
  pinMode(MOTOR_IN2, OUTPUT);
  digitalWrite(MOTOR_IN2, LOW);
  strip.clear();
  strip.show();
  seg.clear();
  seg.writeDisplay();
}

// ─── Setup ─────────────────────────────────────────────
void setup() {
  Serial.begin(115200);
  while (!Serial) {}

  pinMode(IR_PIN,     INPUT);
  pinMode(BUZZER,     OUTPUT);
  pinMode(ENC_CLK,    INPUT);
  pinMode(ENC_DT,     INPUT);
  pinMode(ENC_SW,     INPUT_PULLUP);
  pinMode(BTN1,       INPUT);
  pinMode(BTN2,       INPUT);
  pinMode(BTN3,       INPUT);
  pinMode(FLAME,      INPUT);
  pinMode(MOTOR_IN1,  OUTPUT);
  pinMode(STATUS_LED, OUTPUT);
```

```
  strip.begin();
  strip.show();

  Wire.begin();
  seg.begin(0x70);
  seg.setBrightness(12);

  if (!aht.begin()) {
    Serial.println(F("WARNING: AHT20 not found — check I2C wiring!"));
  }

  printHeader("SmartElex Shield — Automated Test Suite");
  printCommandMenu();
}

// ─── Command menu ──────────────────────────────────────────
void printCommandMenu() {
  Serial.println(F("COMMANDS:"));
  Serial.println(F("  R1      → Start Round 1"));
  Serial.println(F("  R2      → Start Round 2"));
  Serial.println(F("  STOP    → Stop current test"));
  Serial.println(F("  STATUS  → Show all pass/fail results"));
  Serial.println(F("  LIST    → Show test numbers"));
  Serial.println(F("  HELP    → Show this menu"));
  Serial.println(F("  RETRY   → Type number+letter e.g. 8A or 3B"));
  printSeparator();
}

// ─── Main Loop ─────────────────────────────────────────────
void loop() {

  // ── 1. Read serial command ──────────────────────────
  if (Serial.available()) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();
    cmd.toUpperCase();

    if (cmd.length() == 0) {
      // ignore blank lines
    }
    else if (cmd == "R1") {
      startRound(ROUND1);
    }
    else if (cmd == "R2") {
      startRound(ROUND2);
    }
    else if (cmd == "STOP") {
      stopRound();
    }
    else if (cmd == "STATUS") {
      printStatus();
    }
    else if (cmd == "LIST") {
      printList();
    }
    else if (cmd == "HELP") {
      printCommandMenu();
    }
    else {
      // Try to parse as a retry command e.g. "8A" or "3B"
      parseRetry(cmd);
    }
  }
```

```
  // ── 2. Execute running test ──────────────────────────
  if (!testRunning) return;
  if (activeRound == IDLE) return;

  // Resolve which round's slot array we are using
  RoundMode runRound = (activeRound == SINGLE) ? retryRound : activeRound;
  int       count    = (runRound   == ROUND1)  ? R1_COUNT   : R2_COUNT;
  TestSlot* slots    = (runRound   == ROUND1)  ? r1         : r2;

  unsigned long elapsed = millis() - slotStart;

  if (elapsed < slots[testIndex].durationMs) {
    // Still within time window — run the tick
    runSlotTick(testIndex, runRound);
  }
  else {
    // Time window expired — record result and advance
    finalizeSlot(testIndex, runRound);
    stopAllOutputs();

    if (activeRound == SINGLE) {
      // Retry finished — return to IDLE
      Serial.println(F(""));
      Serial.println(F("[RETRY DONE] Type another command or test number."));
      printSeparator();
      activeRound = IDLE;
      testRunning = false;
    }
    else {
      // Full round — move to next test
      testIndex++;
      if (testIndex < count) {
        beginSlot(testIndex, runRound);
      }
      else {
        printRoundSummary(runRound);
        stopRound();
      }
    }
  }
}

// ─── parseRetry ──────────────────────────────────────────
// Accepts "8A", "13A", "3B", "5B" etc.
// Format: one or two digits followed by A or B.
void parseRetry(String cmd) {

  int len = cmd.length();

  // Must be at least 2 characters (e.g. "1A")
  if (len < 2) {
    Serial.print(F("Unknown command: "));
    Serial.println(cmd);
    Serial.println(F("Type HELP to see all commands."));
    return;
  }

  // Last character must be A or B
  char letter = cmd.charAt(len - 1);
  if (letter != 'A' && letter != 'B') {
    Serial.print(F("Unknown command: "));
    Serial.println(cmd);
    Serial.println(F("Type HELP to see all commands."));
    return;
  }
```

```
  // All characters before the letter must be digits
  String numPart = cmd.substring(0, len - 1);
  for (int i = 0; i < (int)numPart.length(); i++) {
    if (numPart.charAt(i) < '0' || numPart.charAt(i) > '9') {
      Serial.print(F("Unknown command: "));
      Serial.println(cmd);
      Serial.println(F("Type HELP to see all commands."));
      return;
    }
  }

  int num = numPart.toInt();   // 1-based test number

  // Determine target round and bounds
  RoundMode targetRound = (letter == 'A') ? ROUND1 : ROUND2;
  int       maxTests    = (targetRound == ROUND1) ? R1_COUNT : R2_COUNT;
  TestSlot* slots       = (targetRound == ROUND1) ? r1 : r2;

  if (num < 1 || num > maxTests) {
    Serial.print(F("ERROR: "));
    Serial.print(letter);
    Serial.print(F(" tests go from 1 to "));
    Serial.println(maxTests);
    return;
  }

  int slotIdx = num - 1;   // convert to 0-based index

  // If something is already running, stop it first
  if (testRunning) {
    stopAllOutputs();
    Serial.println(F("  (Previous test stopped to run retry)"));
  }

  // Reset this slot so it runs fresh
  slots[slotIdx].passed  = false;
  slots[slotIdx].checked = false;

  // Extra motor setup
  if (targetRound == ROUND2 && slotIdx == 3) {
    motorFwd      = true;
    motorToggleAt = millis() + 6000;
    pinMode(MOTOR_IN2, OUTPUT);
  }

  // Announce retry
  Serial.println(F(""));
  printSeparator();
  Serial.print(F(">>> RETRY ["));
  Serial.print(num);
  Serial.print(letter);
  Serial.print(F("]  "));
  Serial.println(slots[slotIdx].name);
  printSeparator();

  // Set state for SINGLE mode
  activeRound = SINGLE;
  retryRound  = targetRound;
  testIndex   = slotIdx;
  testRunning = true;

  // Start the slot
  beginSlot(slotIdx, targetRound);
}
```

```
// ─── Round control ──────────────────────────────────────────
void startRound(RoundMode r) {
  // Reset all slots in the chosen round
  int       count = (r == ROUND1) ? R1_COUNT : R2_COUNT;
  TestSlot* slots = (r == ROUND1) ? r1       : r2;
  for (int i = 0; i < count; i++) {
    slots[i].passed  = false;
    slots[i].checked = false;
  }

  activeRound = r;
  testIndex   = 0;
  testRunning = true;

  if (r == ROUND1) {
    printHeader("ROUND 1 STARTED — Jumper: Joystick/Encoder side");
  } else {
    printHeader("ROUND 2 STARTED — Jumper: POT/Motor/Mic side");
  }
  Serial.println(F("Interact with each component when prompted!"));
  Serial.println(F("Tip: type STOP anytime, then NxA/NxB to retry."));

  beginSlot(0, r);
}

void stopRound() {
  stopAllOutputs();
  activeRound = IDLE;
  testRunning = false;
  testIndex   = 0;
  Serial.println(F(""));
  Serial.println(F("[STOPPED] Type R1, R2, or a test number to retry."));
  printSeparator();
}

// ─── beginSlot ──────────────────────────────────────────────
void beginSlot(int idx, RoundMode r) {
  TestSlot* slots = (r == ROUND1) ? r1 : r2;
  slotStart    = millis();
  encCount     = 0;
  lastClkState = HIGH;

  Serial.println(F(""));
  Serial.print(F("=== Testing: "));
  Serial.print(slots[idx].name);
  Serial.print(F("  ("));
  Serial.print(slots[idx].durationMs / 1000);
  Serial.println(F("s) ==="));

  if (r == ROUND1) {
    switch (idx) {
      case 0:  Serial.println(F("  >> Move joystick in all directions...")); break;
      case 1:  Serial.println(F("  >> Wave hand in front of IR sensor...")); break;
      case 2:  Serial.println(F("  >> Listen for buzzer beeps...")); break;
      case 3:  Serial.println(F("  >> Rotate encoder CW/CCW and press its
button...")); break;
      case 4:  Serial.println(F("  >> Press BTN1 (S1)...")); break;
      case 5:  Serial.println(F("  >> Press BTN2 (S2)...")); break;
      case 6:  Serial.println(F("  >> Press BTN3 (S3)...")); break;
      case 7:  Serial.println(F("  >> Hold flame/lighter near sensor
(carefully!)...")); break;
      case 8:  Serial.println(F("  >> Touch the touch pad...")); break;
      case 9:  Serial.println(F("  >> AHT20 reading automatically...")); break;
      case 10: Serial.println(F("  >> Cover then uncover the LDR...")); break;
```

```
      case 11: Serial.println(F("  >> Watch 7-segment count up...")); break;
      case 12: Serial.println(F("  >> Watch RGB LEDs cycle colors...")); break;
    }
  } else {
    switch (idx) {
      case 0: Serial.println(F("  >> Turn the potentiometer knob...")); break;
      case 1: Serial.println(F("  >> Press BTN4 (S4)...")); break;
      case 2: Serial.println(F("  >> Make noise near the microphone...")); break;
      case 3:
        Serial.println(F("  >> Motor: 6s forward then 6s reverse..."));
        motorFwd      = true;
        motorToggleAt = millis() + 6000;
        pinMode(MOTOR_IN2, OUTPUT);
        break;
      case 4: Serial.println(F("  >> Watch Status LED blink...")); break;
    }
  }
}

// ─── runSlotTick ──────────────────────────────────────────
void runSlotTick(int idx, RoundMode r) {
  if (r == ROUND1) runR1Tick(idx);
  else             runR2Tick(idx);
}

// ─── Round 1 ticks ────────────────────────────────────────
void runR1Tick(int idx) {
  switch (idx) {

    case 0: {  // Joystick
      int x = analogRead(JOY_X);
      int y = analogRead(JOY_Y);
      Serial.print(F("  X=")); Serial.print(x);
      Serial.print(F("  Y=")); Serial.println(y);
      if (x < 400 || x > 600 || y < 400 || y > 600) r1[idx].passed = true;
      r1[idx].checked = true;
      delay(300);
      break;
    }

    case 1: {  // IR
      bool det = (digitalRead(IR_PIN) == LOW);
      Serial.println(det ? F("  IR: Object detected!") : F("  IR: No object"));
      if (det) r1[idx].passed = true;
      r1[idx].checked = true;
      delay(300);
      break;
    }

    case 2: {  // Buzzer
      digitalWrite(BUZZER, HIGH); delay(200);
      digitalWrite(BUZZER, LOW);  delay(200);
      r1[idx].passed  = true;
      r1[idx].checked = true;
      break;
    }

    case 3: {  // Encoder
      int clk = digitalRead(ENC_CLK);
      if (clk != lastClkState) {
        if (clk == LOW) {
          if (digitalRead(ENC_DT) != clk) {
            encCount++;
            Serial.println(F("  Encoder: CW"));
          } else {
```

```
            encCount--;
            Serial.println(F("  Encoder: CCW"));
          }
          r1[idx].passed = true;
        }
        lastClkState = clk;
      }
      if (digitalRead(ENC_SW) == LOW) {
        Serial.println(F("  Encoder button pressed"));
        r1[idx].passed = true;
      }
      r1[idx].checked = true;
      break;
    }

    case 4: {  // BTN1
      if (digitalRead(BTN1) == HIGH) {
        Serial.println(F("  BTN1 pressed!"));
        r1[idx].passed = true;
      }
      r1[idx].checked = true;
      delay(100);
      break;
    }

    case 5: {  // BTN2
      if (digitalRead(BTN2) == HIGH) {
        Serial.println(F("  BTN2 pressed!"));
        r1[idx].passed = true;
      }
      r1[idx].checked = true;
      delay(100);
      break;
    }

    case 6: {  // BTN3
      if (digitalRead(BTN3) == HIGH) {
        Serial.println(F("  BTN3 pressed!"));
        r1[idx].passed = true;
      }
      r1[idx].checked = true;
      delay(100);
      break;
    }

    case 7: {  // Flame
      bool fl = (digitalRead(FLAME) == LOW);
      Serial.println(fl ? F("  FLAME DETECTED!") : F("  No flame"));
      if (fl) r1[idx].passed = true;
      r1[idx].checked = true;
      delay(300);
      break;
    }

    case 8: {  // Touch
      pinMode(TOUCH, INPUT);
      bool t = (digitalRead(TOUCH) == HIGH);
      Serial.println(t ? F("  Touch detected!") : F("  No touch"));
      if (t) r1[idx].passed = true;
      r1[idx].checked = true;
      delay(200);
      break;
    }

    case 9: {  // AHT20
```

```
      sensors_event_t hum, tmp;
      aht.getEvent(&hum, &tmp);
      float temp = tmp.temperature;
      float humd = hum.relative_humidity;
      Serial.print(F("  Temp: "));   Serial.print(temp);
      Serial.print(F(" C  |  Hum: ")); Serial.print(humd);
      Serial.println(F(" %"));
      if (temp > 0 && temp < 50 && humd > 10 && humd < 100) r1[idx].passed = true;
      r1[idx].checked = true;
      delay(2000);
      break;
    }

    case 10: {  // LDR
      int v = analogRead(LDR);
      Serial.print(F("  LDR=")); Serial.println(v);
      if (v > 10 && v < 1010) r1[idx].passed = true;
      r1[idx].checked = true;
      delay(300);
      break;
    }

    case 11: {  // 7-Segment
      static int segNum = 0;
      seg.clear();
      seg.writeDigitNum(0, (segNum / 1000) % 10);
      seg.writeDigitNum(1, (segNum / 100)  % 10);
      seg.writeDigitNum(2, (segNum / 10)   % 10);
      seg.writeDigitNum(3,  segNum % 10);
      seg.writeDisplay();
      segNum = (segNum + 7) % 10000;
      r1[idx].passed  = true;
      r1[idx].checked = true;
      delay(5);
      break;
    }

    case 12: {  // RGB LED
      static uint8_t hue    = 0;
      static int     ledIdx = 0;
      strip.clear();
      strip.setPixelColor(ledIdx, Wheel(hue));
      strip.show();
      hue    = (hue + 3) % 256;
      ledIdx = (ledIdx + 1) % strip.numPixels();
      r1[idx].passed  = true;
      r1[idx].checked = true;
      delay(50);
      break;
    }
  }
}

// ─── Round 2 ticks ──────────────────────────────────────────
void runR2Tick(int idx) {
  switch (idx) {

    case 0: {  // POT
      int v = analogRead(POT);
      Serial.print(F("  POT=")); Serial.println(v);
      if (v > 20 && v < 1000) r2[idx].passed = true;
      r2[idx].checked = true;
      delay(300);
      break;
    }
```

```
    case 1: {  // BTN4
      pinMode(BTN4, INPUT);
      if (digitalRead(BTN4) == HIGH) {
        Serial.println(F("  BTN4 pressed!"));
        r2[idx].passed = true;
      }
      r2[idx].checked = true;
      delay(100);
      break;
    }

    case 2: {  // Microphone
      static int micMin = 1024;
      static int micMax = 0;
      int v = analogRead(MIC);
      Serial.print(F("  MIC=")); Serial.println(v);
      if (v < micMin) micMin = v;
      if (v > micMax) micMax = v;
      if ((micMax - micMin) > 30) r2[idx].passed = true;
      r2[idx].checked = true;
      delay(50);
      break;
    }

    case 3: {  // DC Motor
      pinMode(MOTOR_IN2, OUTPUT);
      unsigned long now = millis();
      if (now >= motorToggleAt) {
        motorFwd      = !motorFwd;
        motorToggleAt = now + 6000;
        Serial.println(motorFwd ? F("  Motor: FORWARD") : F("  Motor: REVERSE"));
      }
      if (motorFwd) {
        digitalWrite(MOTOR_IN1, HIGH);
        digitalWrite(MOTOR_IN2, LOW);
      } else {
        digitalWrite(MOTOR_IN1, LOW);
        digitalWrite(MOTOR_IN2, HIGH);
      }
      r2[idx].passed  = true;
      r2[idx].checked = true;
      delay(200);
      break;
    }

    case 4: {  // Status LED
      static bool ledState = false;
      ledState = !ledState;
      digitalWrite(STATUS_LED, ledState ? HIGH : LOW);
      Serial.println(ledState ? F("  Status LED: ON") : F("  Status LED: OFF"));
      r2[idx].passed  = true;
      r2[idx].checked = true;
      delay(400);
      break;
    }
  }
}

// ─── finalizeSlot ──────────────────────────────────────────
void finalizeSlot(int idx, RoundMode r) {
  TestSlot* slots = (r == ROUND1) ? r1 : r2;
  Serial.print(F("  -> Result: "));
  Serial.println(slots[idx].passed
    ? F("PASS")
```

```
    : F("FAIL (no activity detected)"));
}

// ─── printRoundSummary ──────────────────────────────────────
void printRoundSummary(RoundMode r) {
  int       count  = (r == ROUND1) ? R1_COUNT : R2_COUNT;
  TestSlot* slots  = (r == ROUND1) ? r1       : r2;
  char      letter = (r == ROUND1) ? 'A'      : 'B';
  int passed = 0, failed = 0;

  printHeader((r == ROUND1) ? "ROUND 1 SUMMARY" : "ROUND 2 SUMMARY");

  for (int i = 0; i < count; i++) {
    Serial.print(F("  "));
    if (i + 1 < 10) Serial.print(' ');
    Serial.print(i + 1);
    Serial.print(letter);
    Serial.print(F("  "));
    verdictLine(slots[i].name, slots[i].passed);
    if (slots[i].passed) passed++; else failed++;
  }

  printSeparator();
  Serial.print(F("PASSED: ")); Serial.print(passed);
  Serial.print(F("  /  FAILED: ")); Serial.println(failed);

  if (failed == 0) {
    Serial.println(F("ALL TESTS PASSED — Board is healthy!"));
  } else {
    Serial.println(F("To retry a failed test, type its number+letter."));
    Serial.println(F("Example:  8A  retries Flame Sensor"));
  }
  printSeparator();
}

// ─── printStatus ────────────────────────────────────────────
void printStatus() {
  printSeparator();
  Serial.println(F(">>> CURRENT STATUS  ([----] = not yet tested)"));

  Serial.println(F("-- Round 1 --"));
  for (int i = 0; i < R1_COUNT; i++) {
    Serial.print(F("  "));
    if (i + 1 < 10) Serial.print(' ');
    Serial.print(i + 1);
    Serial.print('A');
    Serial.print(F("  "));
    if (!r1[i].checked) Serial.print(F("[----]  "));
    else Serial.print(r1[i].passed ? F("[PASS]  ") : F("[FAIL]  "));
    Serial.println(r1[i].name);
  }

  Serial.println(F("-- Round 2 --"));
  for (int i = 0; i < R2_COUNT; i++) {
    Serial.print(F("  "));
    if (i + 1 < 10) Serial.print(' ');
    Serial.print(i + 1);
    Serial.print('B');
    Serial.print(F("  "));
    if (!r2[i].checked) Serial.print(F("[----]  "));
    else Serial.print(r2[i].passed ? F("[PASS]  ") : F("[FAIL]  "));
    Serial.println(r2[i].name);
  }

  printSeparator();
```

```
  Serial.println(F("Type  NxA  or  NxB  to retry any test."));
  printSeparator();
}

// ─── printList ──────────────────────────────────────────
void printList() {
  printSeparator();
  Serial.println(F("Round 1 tests:"));
  for (int i = 0; i < R1_COUNT; i++) {
    Serial.print(F("  "));
    if (i + 1 < 10) Serial.print(' ');
    Serial.print(i + 1);
    Serial.print(F("A  "));
    Serial.println(r1[i].name);
  }
  Serial.println(F("Round 2 tests:"));
  for (int i = 0; i < R2_COUNT; i++) {
    Serial.print(F("  "));
    Serial.print(i + 1);
    Serial.print(F("B  "));
    Serial.println(r2[i].name);
  }
  printSeparator();
}

// ─── Rainbow Wheel ──────────────────────────────────────
uint32_t Wheel(byte pos) {
  pos = 255 - pos;
  if (pos < 85)  return strip.Color(255 - pos * 3, 0, pos * 3);
  if (pos < 170) { pos -= 85; return strip.Color(0, pos * 3, 255 - pos * 3); }
  pos -= 170;
  return strip.Color(pos * 3, 255 - pos * 3, 0);
}
```

Quick Reference — All Commands

Test Suite Commands

Serial Monitor Commands (115200 baud)	
`R1`	Start Round 1 automated test (Joystick/Encoder side)
`R2`	Start Round 2 automated test (POT/Motor/Mic side)
`STOP`	Abort current test
`STATUS`	Show pass/fail status for all tests
`LIST`	List all test numbers for current round
`8A`	Retry Round-1 test #8 (Flame Sensor)
`3B`	Retry Round-2 test #3 (Microphone)
`HELP`	Show full command menu

Round 1 Test Numbers (suffix A)

Round 1 — Joystick/Encoder Jumper Side		
`1A` Joystick X/Y	`2A` IR Sensor	`3A` Buzzer
`4A` Rotary Encoder	`5A` BTN1 (S1)	`6A` BTN2 (S2)
`7A` BTN3 (S3)	`8A` Flame Sensor (15s)	`9A` Touch Sensor
`10A` AHT20 Temp/Hum	`11A` LDR	`12A` 7-Segment
`13A` RGB LED Strip		

Round 2 Test Numbers (suffix B)

Round 2 — POT/Motor/Mic Jumper Side		
`1B` Potentiometer	`2B` BTN4 (S4)	`3B` Microphone
`4B` DC Motor (12s: 6s fwd + 6s rev)	`5B` Status LED	

Troubleshooting

Common Issues & Fixes
Display not showing: Check I2C address is 0x70. Ensure Wire.begin() is called in setup().
NeoPixels not lighting: Confirm LED_PIN=3 and strip.begin() called. Check USB power supply.
AHT20 not found: Check I2C wiring. Both AHT20 and 7-seg share I2C — address conflict unlikely but verify.
Buzzer silent: BUZZER pin is D13 — shared with onboard LED. Use tone() not analogWrite().
Motor not running: Set jumper to R2 side. MOTOR_IN2=D4 only works when D4 is not in Touch mode.
Retry command ignored: Make sure Serial Monitor line ending is set to 'Newline' not 'No line ending'.

50