Geekobot
Imagine. Build. Innovate.

EXPLORER KIT

Arduino Nano robotics and smart home kit
Hands-on project manual

61 PROJECTS · BUILD · LEARN · INNOVATE

A Geekobot exclusive · Beginner to advanced · No soldering · geekobot.in

What is in the box

- Nano board (CH340)
- Mini-USB cable
- Nano I/O shield
- 830-point breadboard
- Jumper wires x120
- Red LEDs x10
- Green LEDs x10
- Yellow LEDs x10
- Blue LEDs x10
- RGB LEDs x10
- 220 ohm resistors x20
- 10k resistors x10
- 1k resistors x10
- 2.2k resistors x10
- Push buttons x5
- TTP223 touch modules x4
- Active buzzer
- Passive buzzer
- Sound sensor
- 1-digit 7-segment
- MAX7219 8x8 matrix
- 16x2 I2C LCD
- DHT11 sensor
- 10k potentiometer
- LDR
- Rain sensor
- Soil moisture sensor
- 1-channel relays x2
- Pump with tube
- HC-SR04 x2
- RC522 RFID, card and fob
- SG90 servo
- 4x4 keypad
- HC-05 Bluetooth
- MQ-2 gas sensor
- IR obstacle sensors x2
- IR line sensors x2
- Joystick
- 2WD robot chassis
- L298N motor driver
- DC motor
- Fan blade
- 2 x 18650 holder
- Cable ties x100
- 18650 cells x2 (with-cells option)

Ten levels

Level	Projects	What you do
Level 1 · Foundations	1-10	LEDs, buttons, touch, RGB colour, buzzers and a clap switch.
Level 2 · Displays and games	11-14	A 7-segment dice, an LED matrix and a countdown game.
Level 3 · LCD and environment	15-17	Text, temperature and humidity on a 16x2 LCD.
Level 4 · Light, colour and sensing	18-21	Mood lamp, street light, rain and soil sensing.
Level 5 · Smart automation	22-24	Plant watering, tank filling and a temperature alarm.
Level 6 · Security and access	25-33	RFID, keypad, Bluetooth and touch locks, motion, smoke and fire alarms.
Level 7 · Robotics and motion	34-43	The robot car: Bluetooth, obstacle avoiding, line following, joystick and more.
Level 8 · Wireless control	44-49	Phone control of LEDs, a fan, relays, colour and a notice board.
Level 9 · Interactive games	50-56	Memory game, piano, quiz buzzer, voting machine, roulette, whack-a-mole and snake.
Level 10 · Smart home	57-61	Smart lights, doorbell, water saver and a full smart home demo.

The upload routine (every project)

1. Install the free Arduino IDE 2 from arduino.cc.
2. Seat the Nano on the I/O shield and connect it with the mini-USB cable.
3. Build the circuit from the project's wiring table, with the USB cable unplugged.
4. Paste the code into a new sketch.
5. Tools > Board > Arduino Nano, then Tools > Port and pick the COM port that appears (install the CH340 driver if none does). If the upload times out, choose Tools > Processor > ATmega328P (Old Bootloader).
6. Click Upload and wait for "Done uploading".
7. Libraries: Tools > Manage Libraries, when the project lists one.

Safety and good practice

- Always use the stated resistor with every LED.
- Never drive a motor or the pump from a Nano pin or its 5V pin: use the L298N, the relays and the battery pack.
- The RC522 RFID reader runs on 3.3V only. 5V destroys it.
- Relays: switch only low-voltage demo loads. Never wire mains in a classroom project.
- HC-05: unplug its TXD and RXD wires while you upload, and keep the 1k/2.2k divider on its RXD pin.
- Check VCC, GND and signal before powering up; reversed power can damage a module.
- Unplug USB before rewiring.
- 18650 cells: insert them the right way round, never short the holder wires, switch the holder off after use.
- Nano I/O shield: each pin has its own S (signal), V (5V) and G (ground) header, so a module plugs into one column with female jumpers.

Every project also has its own page with the same wiring and code, plus troubleshooting: geekobot.in/projects/best/geekobot-explorer-kit-projects

CONTENTS

The 61 projects

#	Project	Level	Time
01	Blink One LED	Beginner	15 minutes
02	Alternate Two LEDs	Beginner	20 minutes
03	Traffic Signal System	Beginner	30 minutes
04	Push Button LED	Beginner	20 minutes
05	Touch Sensor Lamp	Beginner	20 minutes
06	RGB Color Lamp	Beginner	30 minutes
07	Police Light Effect	Beginner	20 minutes
08	Buzzer Beep Sound	Beginner	20 minutes
09	Musical Buzzer	Beginner	30 minutes
10	Clap Detection	Beginner	30 minutes
11	Digital Dice	Beginner	45 minutes
12	Scrolling Text Display	Beginner	30 minutes
13	Emoji Display on LED Matrix	Beginner	30 minutes
14	Countdown Timer Game	Beginner	45 minutes
15	LCD Welcome System	Beginner	30 minutes
16	LCD Temperature Display	Beginner	30 minutes
17	LCD Humidity Monitor	Beginner	30 minutes
18	RGB Mood Lamp	Beginner	30 minutes
19	Smart Street Light	Beginner	45 minutes
20	Smart Umbrella Reminder	Beginner	30 minutes
21	Soil Moisture Detector	Beginner	30 minutes
22	Automatic Plant Watering System	Intermediate	1 hour
23	Water Tank Automation	Intermediate	1 hour
24	Smart Temperature Alarm	Intermediate	45 minutes
25	RFID Door Lock	Intermediate	1 hour
26	Password Door Lock	Intermediate	1 hour
27	Bluetooth Door Lock	Intermediate	1 hour
28	Motion Detection Alarm	Intermediate	30 minutes
29	Touch Password System	Intermediate	45 minutes
30	Smoke Detection Alarm	Intermediate	45 minutes
31	Fire Warning System	Intermediate	1 hour
32	Smart Locker System	Intermediate	1 hour 30 minutes
33	Silent Security Alert	Intermediate	30 minutes
34	Bluetooth Controlled Car	Intermediate	2 hours
35	Obstacle Avoiding Robot	Intermediate	1 hour 30 minutes

36	Line Following Robot	Intermediate	1 hour 30 minutes
37	Hand-Controlled Robot	Intermediate	1 hour
38	Voice Buzzer Robot	Intermediate	1 hour
39	Maze Solver Beginner Bot	Advanced	2 hours
40	Smart Parking Robot Demo	Intermediate	1 hour
41	Mini Delivery Robot	Intermediate	1 hour 30 minutes
42	Auto Reverse Collision Robot	Intermediate	1 hour
43	Gesture-like Joystick Car	Intermediate	1 hour
44	Bluetooth LED Controller	Intermediate	45 minutes
45	Bluetooth Fan Control	Intermediate	1 hour
46	Bluetooth Home Automation	Advanced	1 hour 30 minutes
47	Wireless RGB Controller	Intermediate	45 minutes
48	Bluetooth Notice Board	Intermediate	45 minutes
49	Wireless Sensor Dashboard	Advanced	1 hour
50	Memory Sequence Game	Intermediate	1 hour
51	Mini Piano	Intermediate	45 minutes
52	Quiz Buzzer System	Intermediate	1 hour
53	Electronic Voting Machine	Intermediate	1 hour
54	Digital Roulette	Intermediate	45 minutes
55	Whack-a-Mole Style Game	Intermediate	1 hour
56	Joystick Snake Game	Advanced	1 hour
57	Smart Room Light	Intermediate	45 minutes
58	Automatic Bathroom Light	Intermediate	45 minutes
59	Smart Doorbell	Intermediate	45 minutes
60	Intelligent Water Saver	Intermediate	1 hour
61	Mini Smart Home Demo	Advanced	2 hours

PROJECT 01 · LEVEL 1 · FOUNDATIONS

Blink One LED

Goal: Make a single LED switch on and off at a steady rate, the first program every maker writes.

Two instructions carry the whole sketch. pinMode() tells the Nano that D8 is an output, a pin that pushes voltage out rather than listening for it. digitalWrite() then sets that pin to HIGH (5 volts, LED on) or LOW (0 volts, LED off). delay() simply stops everything for the number of milliseconds you give it, which is why the LED holds each state for a full second. Because loop() runs forever, the four instructions inside it repeat for as long as the board has power.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-01-blink-one-led

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- What an output pin is
- pinMode() and digitalWrite()
- How delay() controls timing
- Why loop() repeats forever

Wiring

Part	Pin	Nano	Note
LED	LED long leg (anode)	**D8 (S pin)**	Through a 220 ohm resistor
LED	LED short leg (cathode)	**D8 (G pin)**	

Code

```
int led = 8;

void setup()
{
  pinMode(led, OUTPUT);
  Serial.begin(9600);
  Serial.println("Blink started");
}

void loop()
{
  digitalWrite(led, HIGH);                        // Voltage on - LED lights
  Serial.println("ON");
  delay(1000);                                    // Wait one second

  digitalWrite(led, LOW);                   // Voltage off - LED dark
  Serial.println("OFF");
  delay(1000);
}
```

What you should see: The LED lights for one second, then goes dark for one second. The pattern repeats forever. Serial Monitor prints ON and OFF in time with the LED.

Alternate Two LEDs

Goal: Run two LEDs so that one is always on and the other always off, swapping over at a fixed rate.

Two outputs now need managing at once, and the trick is that they are always opposites. Setting one HIGH in the same breath as setting the other LOW guarantees they can never both be lit. Try adding a second delay between the two digitalWrite() calls and you will see both LEDs briefly on together, a small experiment that shows exactly how literally the board follows the order you write.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-02-alternate-two-leds

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Controlling several outputs
- Keeping two states as exact opposites
- Using a variable to hold a timing value
- How instruction order affects what you see

Wiring

Part	Pin	Nano	Note
LED	Red LED anode	**D8 (S pin)**	Through a 220 ohm resistor
LED	Red LED cathode	**D8 (G pin)**	
LED	Green LED anode	**D9 (S pin)**	Through a 220 ohm resistor
LED	Green LED cathode	**D9 (G pin)**	

Code

```
int redLed = 8;
int greenLed = 9;
int waitTime = 500;

void setup()
{
  pinMode(redLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  digitalWrite(redLed, HIGH);
  digitalWrite(greenLed, LOW);
  Serial.println("Red on, green off");
  delay(waitTime);

  digitalWrite(redLed, LOW);
  digitalWrite(greenLed, HIGH);
  Serial.println("Green on, red off");
  delay(waitTime);
}
```

What you should see: The red LED lights while the green one stays dark. After half a second they swap over. The alternation continues indefinitely. Serial Monitor names whichever LED is currently lit.

Traffic Signal System

Goal: Recreate a road traffic light with red, yellow and green LEDs following the correct real-world sequence.

A real signal is a fixed sequence of states, each held for a different length of time, and the timings are not arbitrary. Green and red are long because traffic needs time to move; yellow is short because it is only a warning. Writing each phase as its own block of code, with allOff() called first, means no phase can accidentally leave a previous lamp lit. That habit, clear everything, then set what you want, prevents a whole class of bugs later on.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-03-traffic-signal

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Real-life automation sequences
- Breaking a process into named phases
- Clearing all outputs before setting new ones
- Why different phases need different durations

Wiring

Part	Pin	Nano	Note
LED	Red LED anode	**D8 (S pin)**	Through a 220 ohm resistor
LED	Yellow LED anode	**D9 (S pin)**	Through a 220 ohm resistor
LED	Green LED anode	**D10 (S pin)**	Through a 220 ohm resistor
LED	All cathodes	**Any G pin**	

Code

```
int redLed = 8;
int yellowLed = 9;
int greenLed = 10;

void allOff()
{
  digitalWrite(redLed, LOW);
  digitalWrite(yellowLed, LOW);
  digitalWrite(greenLed, LOW);
}

void setup()
{
  pinMode(redLed, OUTPUT);
  pinMode(yellowLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  // STOP
  allOff();
  digitalWrite(redLed, HIGH);
  Serial.println("RED - stop");
  delay(5000);
  // GET READY
  allOff();
  digitalWrite(redLed, HIGH);
  digitalWrite(yellowLed, HIGH);
  Serial.println("RED + YELLOW - get ready");
  delay(2000);

  // GO
  allOff();
  digitalWrite(greenLed, HIGH);
  Serial.println("GREEN - go");
  delay(5000);

  // PREPARE TO STOP
  allOff();
  digitalWrite(yellowLed, HIGH);
  Serial.println("YELLOW - prepare to stop");
  delay(2000);
}
```

What you should see: Red holds for 5 seconds. Red and yellow light together for 2 seconds. Green holds for 5 seconds. Yellow alone for 2 seconds, then the cycle restarts.

Push Button LED

Goal: Read a button as an input and light an LED only while the button is held down.

An input pin left unconnected picks up electrical noise and reads randomly, so it needs pulling to a known voltage. INPUT_PULLUP switches on a resistor inside the Nano that holds the pin at 5 V, which is why an unpressed button reads HIGH. Pressing the button connects the pin to ground, so it reads LOW. That inversion catches everyone at first: pressed means LOW, not HIGH.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-04-push-button-led

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Reading a digital input
- Why INPUT_PULLUP is needed
- Pressed reads LOW, not HIGH
- Detecting the moment of a change rather than a held state

Wiring

Part	Pin	Nano	Note
Button	Button one leg	**D2 (S pin)**	
Button	Button other leg	**D2 (G pin)**	
LED	LED anode	**D8 (S pin)**	Through a 220 ohm resistor
LED	LED cathode	**D8 (G pin)**	

Code

```
int button = 2;
int led = 8;
int presses = 0;
bool wasPressed = false;

void setup()
{
  pinMode(button, INPUT_PULLUP);             // Internal resistor holds the pin HIGH
  pinMode(led, OUTPUT);

  Serial.begin(9600);
  Serial.println("Press the button");
}

void loop()
{
  bool pressed = (digitalRead(button) == LOW);              // LOW means pressed

  if (pressed)
  {
    digitalWrite(led, HIGH);
  }
  else
  {
    digitalWrite(led, LOW);
  }
  // Count only the moment of pressing, not the whole hold
  if (pressed && !wasPressed)
  {
    presses = presses + 1;
    Serial.print("Press number ");
    Serial.println(presses);
  }
  wasPressed = pressed;

  delay(20);
}
```

What you should see: The LED is dark while the button is untouched. Hold the button and the LED lights. Release it and the LED goes out immediately. Serial Monitor counts each press exactly once, however long you hold.

Touch Sensor Lamp

Goal: Use a capacitive touch pad as a switch, where one tap turns the light on and the next turns it off.

The TTP223 senses the tiny change in electrical capacitance your finger adds to its copper pad, no moving parts, and it works through several millimetres of plastic or wood. The important idea in the code is toggling: instead of the LED following the pad, one touch flips the state. The ! operator means "the opposite of", so lampOn = !lampOn turns true into false and back. Waiting for the finger to lift stops one touch from flipping the state dozens of times.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-05-touch-sensor-lamp

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Capacitive Touch Sensor Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Capacitive touch sensing basics
- Toggling a state with the ! operator
- Storing a state in a bool variable
- Waiting for release so one touch counts once

Wiring

Part	Pin	Nano	Note
Touch module	VCC	**D2 (V pin)**	
Touch module	GND	**D2 (G pin)**	
Touch module	SIG	**D2 (S pin)**	
LED	LED anode	**D8 (S pin)**	Through a 220 ohm resistor
LED	LED cathode	**D8 (G pin)**	

Code

```
int touchPin = 2;
int led = 8;

bool lampOn = false;

void setup()
{
  pinMode(touchPin, INPUT);
  pinMode(led, OUTPUT);

  Serial.begin(9600);
  Serial.println("Touch the pad to switch the lamp");
}

void loop()
{
  if (digitalRead(touchPin) == HIGH)
  {
    lampOn = !lampOn;                          // Flip to the opposite state
    digitalWrite(led, lampOn);

    Serial.print("Lamp is now ");
    Serial.println(lampOn ? "ON" : "OFF");
    // Wait until the finger is lifted before accepting another touch
    while (digitalRead(touchPin) == HIGH)
    {
      delay(10);
    }
    delay(50);
  }
}
```

What you should see: Touch the pad once and the LED lights. Take your finger away, the LED stays on. Touch again and it goes out. Serial Monitor reports the new state after each touch.

RGB Color Lamp

Goal: Produce many different colours from a single RGB LED by mixing three brightness levels.

An RGB LED is three separate LEDs sharing one package. digitalWrite() could only ever give eight colours, because each channel would be fully on or fully off. analogWrite() instead uses PWM, it switches the pin on and off thousands of times a second, and the fraction of time it spends on decides the apparent brightness. A value of 0 is off, 255 is full, and everything between is a dimmer level, which is how three channels produce millions of colours.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-06-rgb-colour-lamp

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 5mm RGB LED (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Colour mixing from three channels
- PWM and analogWrite() for brightness
- Why 0 to 255 is the range
- Fading by stepping a value in a loop

Wiring

Part	Pin	Nano	Note
RGB LED	Red leg	**D9 (S pin)**	Through a 220 ohm resistor
RGB LED	Green leg	**D10 (S pin)**	Through a 220 ohm resistor
RGB LED	Blue leg	**D11 (S pin)**	Through a 220 ohm resistor
RGB LED	Common cathode (longest leg)	**Any G pin**	

Code

```
int redPin = 9;
int greenPin = 10;
int bluePin = 11;

void showColour(int r, int g, int b, String name)
{
  analogWrite(redPin, r);
  analogWrite(greenPin, g);
  analogWrite(bluePin, b);

  Serial.print(name);
  Serial.print(" R");
  Serial.print(r);
  Serial.print(" G");
  Serial.print(g);
  Serial.print(" B");
  Serial.println(b);

  delay(1500);
}

void setup()
{
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  showColour(255,   0,   0, "Red");
  showColour( 0, 255,    0, "Green");
  showColour( 0,    0, 255, "Blue");
  showColour(255, 255,   0, "Yellow");
  showColour( 0, 255, 255, "Cyan");
  showColour(255,   0, 255, "Magenta");
  showColour(255, 120,   0, "Orange");
  showColour(255, 255, 255, "White");

  // Smooth fade from red to green
  for (int i = 0; i <= 255; i = i + 5)
  {
    analogWrite(redPin, 255 - i);
    analogWrite(greenPin, i);
    analogWrite(bluePin, 0);
    delay(20);
  }
}
```

What you should see: The lamp steps through eight named colours, holding each for 1.5 seconds. Serial Monitor prints the name and the three channel values. After white it fades smoothly from red to green. The whole sequence then repeats.

Police Light Effect

Goal: Create a convincing emergency-vehicle flash using two LEDs and a repeating burst pattern.

A police light is not a simple alternation, it is a rapid burst on one side, a pause, then a burst on the other. That structure is what makes it read as urgent rather than decorative. A for loop produces each burst without repeating the same three lines four times over, and changing the loop's limit changes the number of flashes per burst. Short delays inside the burst and a longer one between them create the rhythm.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-07-police-light-effect

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Patterns and timing
- for loops to repeat an action
- Writing a reusable function with parameters
- How rhythm changes the character of a light

Wiring

Part	Pin	Nano	Note
LED	Red LED anode	**D8 (S pin)**	Through a 220 ohm resistor
LED	Blue LED anode	**D9 (S pin)**	Through a 220 ohm resistor
LED	Both cathodes	**Any G pin**	

Code

```
int redLed = 8;
int blueLed = 9;

int flashesPerBurst = 4;
int flashTime = 60;
int gapBetweenBursts = 120;

void burst(int pin, int count)
{
  for (int i = 0; i < count; i++)
  {
    digitalWrite(pin, HIGH);
    delay(flashTime);
    digitalWrite(pin, LOW);
    delay(flashTime);
  }
}

void setup()
{
  pinMode(redLed, OUTPUT);
  pinMode(blueLed, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  Serial.println("Red burst");
  burst(redLed, flashesPerBurst);
  delay(gapBetweenBursts);

  Serial.println("Blue burst");
  burst(blueLed, flashesPerBurst);
  delay(gapBetweenBursts);
}
```

What you should see: The red LED flashes four times in rapid succession. A short pause follows. The blue LED flashes four times. The pattern repeats continuously with an urgent rhythm.

Buzzer Beep Sound

Goal: Drive a buzzer to make alarm sounds, and hear the difference between simple beeps and varied patterns.

An active buzzer contains its own oscillator, so digitalWrite() HIGH is enough to make it sound, it plays one fixed pitch and nothing else. What you can control is the pattern: how long each beep lasts and how long the silence between them is. Those two numbers alone carry a surprising amount of meaning, which is why a slow beep reads as a reminder and a fast one as an emergency.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-08-buzzer-beep

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Sound output from a digital pin
- How an active buzzer differs from a passive one
- Building patterns from on-time and off-time
- Using repetition rate to signal urgency

Wiring

Part	Pin	Nano	Note
Buzzer	Buzzer (+)	**D8 (S pin)**	
Buzzer	Buzzer (−)	**D8 (G pin)**	
LED	LED anode	**D9 (S pin)**	Through a 220 ohm resistor
LED	LED cathode	**D9 (G pin)**	

Code

```
int buzzer = 8;
int led = 9;

void beep(int onTime, int offTime, int count)
{
  for (int i = 0; i < count; i++)
  {
    digitalWrite(buzzer, HIGH);
    digitalWrite(led, HIGH);
    delay(onTime);

    digitalWrite(buzzer, LOW);
    digitalWrite(led, LOW);
    delay(offTime);
  }
}

void setup()
{
  pinMode(buzzer, OUTPUT);
  pinMode(led, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  Serial.println("Single long beep");
  beep(800, 400, 1);
  delay(800);
  Serial.println("Reminder - two slow beeps");
  beep(200, 300, 2);
  delay(1200);

  Serial.println("Warning - four medium beeps");
  beep(120, 120, 4);
  delay(1200);

  Serial.println("Emergency - rapid beeping");
  beep(50, 50, 10);
  delay(2000);
}
```

What you should see: One long beep, with the LED flashing in time. Two slower beeps as a gentle reminder. Four medium beeps as a warning. Ten rapid beeps that sound genuinely urgent.

Musical Buzzer

Goal: Play recognisable tunes on a passive buzzer by sending it specific frequencies.

Pitch is frequency measured in hertz, how many times per second the buzzer's disc is pushed and pulled. Middle C is 262 Hz, and doubling any frequency raises it exactly one octave, which is why 523 Hz is the C above. tone() generates that signal in the background so the sketch can carry on, and noTone() stops it. A tune is then just two arrays walked in step: one of pitches and one of how long each note lasts.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-09-musical-buzzer

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Musical notes as frequencies
- tone() and noTone() for pitch
- Parallel arrays for pitch and rhythm
- Why doubling a frequency gives the same note an octave higher

Wiring

Part	Pin	Nano	Note
Passive buzzer	Buzzer (+)	**D8 (S pin)**	
Passive buzzer	Buzzer (−)	**D8 (G pin)**	
Button	Button one leg	**D2 (S pin)**	
Button	Button other leg	**D2 (G pin)**	

Code

```
int buzzer = 8;
int button = 2;

// Note frequencies in hertz
int NOTE_C4 = 262, NOTE_D4 = 294, NOTE_E4 = 330, NOTE_F4 = 349;
int NOTE_G4 = 392, NOTE_A4 = 440, NOTE_B4 = 494, NOTE_C5 = 523;

// Twinkle Twinkle Little Star
int melody[14] =   {NOTE_C4, NOTE_C4, NOTE_G4, NOTE_G4, NOTE_A4, NOTE_A4, NOTE_G4, NOTE_F4, NOTE_F4,
NOTE_E4, NOTE_E4, NOTE_D4, NOTE_D4, NOTE_C4};
int beats[14] =    { 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2};
int beatLength = 400;

void playTune()
{
  for (int i = 0; i < 14; i++)
  {
    int duration = beats[i] * beatLength;

    tone(buzzer, melody[i], duration * 0.9);
    Serial.print("Note ");
    Serial.print(melody[i]);
    Serial.println(" Hz");

    delay(duration);               // Full beat, including the short gap
  }
  noTone(buzzer);
}

void playScale()
{
  int scale[8] = {NOTE_C4, NOTE_D4, NOTE_E4, NOTE_F4, NOTE_G4, NOTE_A4, NOTE_B4, NOTE_C5};

  for (int i = 0; i < 8; i++)
  {
    tone(buzzer, scale[i], 250);
    delay(300);
  }
  noTone(buzzer);
}

void setup()
{
  pinMode(buzzer, OUTPUT);
  pinMode(button, INPUT_PULLUP);

  Serial.begin(9600);
  Serial.println("Press the button to play");

  playScale();
}

void loop()
{
  if (digitalRead(button) == LOW)
  {
    Serial.println("Playing the tune");
    playTune();

    while (digitalRead(button) == LOW)
    {
      delay(10);
    }
    delay(300);
  }
}
```

What you should see: A rising eight-note scale plays when the board starts. Press the button and Twinkle Twinkle Little Star plays. Serial Monitor shows the frequency of each note. Longer notes are held for two beats instead of one.

Clap Detection

Goal: Make an LED respond to a clap, using a sound sensor and a guard that ignores the echo.

The module's digital output pulses LOW whenever the sound crosses the level set by its onboard potentiometer. The complication is that one clap in a hard-walled room produces several pulses as it echoes off the surfaces. Recording the time of each accepted clap and refusing anything within 300 ms of it filters those echoes out. That guard is the difference between a light that toggles once and one that flickers unpredictably.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-10-clap-detection

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Sound sensing with a digital output
- Using millis() to record when something happened
- Guard windows that reject echoes
- Combining sensing with a toggled state

Wiring

Part	Pin	Nano	Note
Sound sensor	VCC	**D2 (V pin)**	
Sound sensor	GND	**D2 (G pin)**	
Sound sensor	DO (digital out)	**D2 (S pin)**	
LED	LED anode	**D8 (S pin)**	Through a 220 ohm resistor
LED	LED cathode	**(G pin)**	
Buzzer	Buzzer (+)	**D9 (S pin)**	
Buzzer	Buzzer (−)	**(G pin)**	

Code

```
int soundPin = 2;
int led = 8;
int buzzer = 9;

bool lightOn = false;
unsigned long lastClap = 0;
unsigned long deadTime = 300;                 // Ignore echoes for this long
int clapCount = 0;

void setup()
{
  pinMode(soundPin, INPUT);
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);

  Serial.begin(9600);
  Serial.println("Clap to switch the light");
}

void loop()
{
  if (digitalRead(soundPin) == LOW)
  {
    if (millis() - lastClap > deadTime)
    {
      lastClap = millis();
      clapCount = clapCount + 1;

      lightOn = !lightOn;
      digitalWrite(led, lightOn);

      digitalWrite(buzzer, HIGH);
      delay(40);
      digitalWrite(buzzer, LOW);

      Serial.print("Clap ");
      Serial.print(clapCount);
      Serial.print(" - light is ");
      Serial.println(lightOn ? "ON" : "OFF");
    }
  }
}
```

What you should see: Clap once and the LED lights with a short chirp. Clap again and it goes out. Echoes in a hard room do not cause extra switching. Serial Monitor counts every accepted clap.

Digital Dice

Goal: Roll an electronic dice: press a button and a random number from 1 to 6 appears on a 7-segment display, complete with a rolling animation.

Each digit is drawn by lighting a particular combination of the seven segments. Those combinations are stored in a two- dimensional array, so showing a number is simply a matter of walking through one row of that array. randomSeed(analogRead(A0)) reads electrical noise from an unconnected analog pin, which gives a different starting point every time the board is powered up, without it, the dice would roll the same sequence after every reset.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-11-digital-dice

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x 7-Segment LED Display 0.56"
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Random number generation with random() and randomSeed()
- 7-segment logic, which segments make which digit
- Two-dimensional arrays

Wiring

Part	Pin	Nano	Note
7-segment	Segment a	**D2**	Through a 220 ohm resistor
7-segment	Segment b	**D3**	Through a 220 ohm resistor
7-segment	Segment c	**D4**	Through a 220 ohm resistor
7-segment	Segment d	**D5**	Through a 220 ohm resistor
7-segment	Segment e	**D6**	Through a 220 ohm resistor
7-segment	Segment f	**D7**	Through a 220 ohm resistor
7-segment	Segment g	**D8**	Through a 220 ohm resistor
7-segment	Common (both COM pins)	**GND**	
Button	One leg	**D9**	
Button	Other leg	**GND**	

Code

```
int segPins[7] = {2, 3, 4, 5, 6, 7, 8};
int buttonPin = 9;

// Segment order: a b c d e f g
byte digits[7][7] =
{
  {0, 0, 0, 0, 0, 0, 0}, // blank
  {0, 1, 1, 0, 0, 0, 0}, // 1
  {1, 1, 0, 1, 1, 0, 1}, // 2
  {1, 1, 1, 1, 0, 0, 1}, // 3
  {0, 1, 1, 0, 0, 1, 1}, // 4
  {1, 0, 1, 1, 0, 1, 1}, // 5
  {1, 0, 1, 1, 1, 1, 1}   // 6
};

void showDigit(int n)
{
  for (int i = 0; i < 7; i++)
  {
    digitalWrite(segPins[i], digits[n][i]);
  }
}

void setup()
{
  for (int i = 0; i < 7; i++)
  {
    pinMode(segPins[i], OUTPUT);
  }
  pinMode(buttonPin, INPUT_PULLUP);
  randomSeed(analogRead(A0));
  showDigit(0);
}

void loop()
{
  if (digitalRead(buttonPin) == LOW)
  {
    // Rolling animation
    for (int i = 0; i < 15; i++)
    {
      showDigit(random(1, 7));
      delay(80);
    }

    int result = random(1, 7);
    showDigit(result);
    delay(600);
  }
}
```

What you should see: Display stays blank until the button is pressed. Numbers flicker rapidly for about one second. A single number between 1 and 6 stays on screen.

Scrolling Text Display

Goal: Scroll your own message across an 8×8 LED matrix, letter by letter, the way a railway station board does.

An 8×8 matrix has 64 LEDs but only 16 pins. The MAX7219 driver lights one row at a time, extremely fast, and your eye blends the rows into a single steady image, this is called multiplexing. The library holds the message as an array of characters and shifts the pixel columns one step to the left on every animation frame, which is what produces the scroll.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-12-scrolling-text-matrix

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x MAX7219 LED Display Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Matrix multiplexing, how 64 LEDs are driven from one chip
- Arrays of characters and pixel columns
- Using a display library
- Controlling scroll speed and brightness

Wiring

Part	Pin	Nano	Note
Matrix	VCC	**5V**	
Matrix	GND	**GND**	
Matrix	DIN	**D11**	
Matrix	CS	**D10**	
Matrix	CLK	**D13**	

Code

```
#include <MD_Parola.h>
#include <MD_MAX72xx.h>
#include <SPI.h>

#define HARDWARE_TYPE MD_MAX72XX::FC16_HW
#define MAX_DEVICES 1
#define CS_PIN 10

MD_Parola display = MD_Parola(HARDWARE_TYPE, CS_PIN, MAX_DEVICES);

void setup()
{
  display.begin();
  display.setIntensity(5);
  display.displayClear();

  display.displayText("HELLO STUDENTS",
  PA_CENTER,
  80,
  0,
  PA_SCROLL_LEFT,
  PA_SCROLL_LEFT);
}

void loop()

{
  if (display.displayAnimate())
  {
    display.displayReset();
  }
}
```

What you should see: The message HELLO STUDENTS scrolls right to left. The scroll repeats continuously. Change the text inside displayText() to show your own message.

Emoji Display on LED Matrix

Goal: Design your own pixel art, a smiley and a heart, and display them on the 8×8 LED matrix.

Each row of the matrix is one byte, eight bits, one per LED. Writing B00111100 turns on the four middle LEDs of that row and leaves the rest off. Stack eight of those bytes and you have drawn a picture. This is bitmapping, and it is exactly how fonts and icons are stored on every small display.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-13-emoji-led-matrix

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x MAX7219 LED Display Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Bitmapping, turning binary numbers into pictures
- Custom characters and icons
- Passing arrays to a function
- Row-by-row control of a matrix

Wiring

Part	Pin	Nano	Note
Matrix	VCC	**5V**	
Matrix	GND	**GND**	
Matrix	DIN	**D11**	
Matrix	CS	**D10**	
Matrix	CLK	**D13**	

Code

```
#include <LedControl.h>

// LedControl(DIN, CLK, CS, number of devices)
LedControl lc = LedControl(11, 13, 10, 1);

byte smiley[8] =
{
  B00111100,
  B01000010,
  B10100101,
  B10000001,
  B10100101,
  B10011001,
  B01000010,
  B00111100
};

byte heart[8] =
{
  B00000000,
  B01100110,
  B11111111,
  B11111111,
  B11111111,
  B01111110,
  B00111100,
  B00011000
};

void drawImage(byte image[])
{
  for (int row = 0; row < 8; row++)
  {
    lc.setRow(0, row, image[row]);
  }
}

void setup()

{
  lc.shutdown(0, false);
  lc.setIntensity(0, 5);
  lc.clearDisplay(0);
}

void loop()
{
  drawImage(smiley);
  delay(2000);
  drawImage(heart);
  delay(2000);
}
```

What you should see: A smiley face appears for 2 seconds. A heart appears for 2 seconds. The two images alternate continuously.

Countdown Timer Game

Goal: Build a beat-the-clock game: an LCD counts down from ten while the buzzer beeps, and the player must press the stop button before time runs out.

The sketch cannot simply call delay(1000) between ticks, during a delay the Arduino ignores the button, so a player could press it and nothing would happen. Instead each second is split into ninety short 10 ms waits, and the button is checked between every one of them. A boolean variable remembers whether the player stopped the clock in time. Together, the counter and the flag are the state of the game.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-14-countdown-game

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Tactile Push Button Switch
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Complex timing without blocking the program
- State management with flag variables
- Splitting a long wait so inputs stay responsive
- Printing changing values to an LCD

Wiring

Part	Pin	Nano	Note
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**A4**	
LCD	SCL	**A5**	
Buzzer	Buzzer (+)	**D8**	
Buzzer	Buzzer (−)	**GND**	
Button	Button one leg	**D7**	
Button	Button other leg	**GND**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

int buzzer = 8;
int button = 7;
int timeLeft = 10;
bool stopped = false;

void setup()
{
  lcd.init();
  lcd.backlight();
  pinMode(buzzer, OUTPUT);
  pinMode(button, INPUT_PULLUP);

  lcd.setCursor(0, 0);
  lcd.print("COUNTDOWN START");
  delay(1500);
  lcd.clear();
}

void loop()
{
  if (stopped)
  {
    return;
  }
  if (timeLeft > 0)
  {
    lcd.setCursor(0, 0);
    lcd.print("Time left: ");
    lcd.print(timeLeft);
    lcd.print(" ");

    digitalWrite(buzzer, HIGH);
    delay(100);
    digitalWrite(buzzer, LOW);

    // Wait the rest of the second, watching the button
    for (int i = 0; i < 90; i++)
    {
      if (digitalRead(button) == LOW)
      {
        stopped = true;
        lcd.clear();
        lcd.setCursor(0, 0);
        lcd.print("YOU WIN!");
        return;
      }
      delay(10);
    }

    timeLeft = timeLeft - 1;
  }
  else
  {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("TIME UP!");

    digitalWrite(buzzer, HIGH);
    delay(2000);
    digitalWrite(buzzer, LOW);

    while (true)
    {
      // Game over - press reset to play again
    }
  }
}
```

What you should see: LCD shows COUNTDOWN START, then Time left: 10, 9, 8 …. The buzzer ticks once every second. Press the button in time and the LCD shows YOU WIN! Let it reach zero and the LCD shows TIME UP! with a long buzz.

LCD Welcome System

Goal: Display a personalised greeting on a 16×2 LCD, the foundation for every project in this manual that shows text.

An I2C LCD needs only two signal wires because the small board on its back converts the two-wire I2C messages into the sixteen signals the display itself expects. lcd.setCursor(column, row) chooses where the next text will start: columns are numbered 0 to 15 and rows 0 to 1.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-15-lcd-welcome

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- LCD interfacing over I2C
- String printing and the String variable type
- Positioning text with setCursor()
- Clearing and redrawing a screen

Wiring

Part	Pin	Nano	Note
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**A4**	
LCD	SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

String studentName = "STUDENT";

void setup()
{
  lcd.init();
  lcd.backlight();
}

void loop()
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("WELCOME");
  lcd.setCursor(0, 1);
  lcd.print(studentName);
  delay(3000);

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Have a great");
  lcd.setCursor(0, 1);
  lcd.print("day ahead!");
  delay(3000);
}
```

What you should see: WELCOME on the top line and the name below it. After 3 seconds the message changes to Have a great day ahead! The two screens alternate continuously.

LCD Temperature Display

Goal: Read the room temperature from a DHT11 sensor and show it live on the LCD instead of the Serial Monitor.

The DHT11 needs about two seconds between readings, so the sketch waits before asking again. Temperature comes back as a float, a number with a decimal point, and the degree symbol is written with lcd.write(223), because that character is not part of ordinary ASCII.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-16-lcd-temperature

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Sensor libraries and how they hide the hard work
- LCD formatting and special characters
- Working with float values
- Checking a reading with isnan() before trusting it

Wiring

Part	Pin	Nano	Note
DHT11	VCC	**5V**	
DHT11	GND	**GND**	
DHT11	DATA	**D2**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**A4**	
LCD	SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>

#define DHTPIN 2
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup()
{
  lcd.init();
  lcd.backlight();
  dht.begin();

  lcd.setCursor(0, 0);
  lcd.print("Room Monitor");
  delay(2000);
  lcd.clear();
}

void loop()
{
  float temp = dht.readTemperature();

  if (isnan(temp))
  {
    lcd.setCursor(0, 0);
    lcd.print("Sensor error              ");
    delay(2000);
    return;
  }

  lcd.setCursor(0, 0);
  lcd.print("Temperature:                   ");
  lcd.setCursor(0, 1);
  lcd.print(temp);
  lcd.write(223);
  lcd.print("C      ");

  delay(2000);
}
```

What you should see: Temperature: on the top line. 29.00°C on the second line. The value refreshes every 2 seconds.

LCD Humidity Monitor

Goal: Track the moisture in the air continuously, showing the current reading together with the highest and lowest values seen so far.

A single reading tells you very little. By keeping two extra variables, one for the smallest value ever seen and one for the largest, the sketch turns a series of instant readings into a record of what happened while you were not watching. This is the essence of continuous monitoring, and the same three lines of logic appear in weather stations and data loggers.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-17-lcd-humidity-monitor

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Continuous sensor monitoring
- Keeping running minimum and maximum values
- Comparing floats with if statements
- Controlling decimal places when printing

Wiring

Part	Pin	Nano	Note
DHT11	VCC	**5V**	
DHT11	GND	**GND**	
DHT11	DATA	**D2**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**A4**	
LCD	SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>

#define DHTPIN 2
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);

float minHum = 100.0;
float maxHum = 0.0;

void setup()
{
  lcd.init();
  lcd.backlight();
  dht.begin();

  lcd.setCursor(0, 0);
  lcd.print("Humidity Log");
  delay(2000);
  lcd.clear();
}

void loop()

{
  float hum = dht.readHumidity();
  if (isnan(hum))
  {
    lcd.setCursor(0, 0);
    lcd.print("Sensor error               ");
    delay(2000);
    return;
  }

  if (hum < minHum)
  {
    minHum = hum;
  }
  if (hum > maxHum)
  {
    maxHum = hum;
  }

  lcd.setCursor(0, 0);
  lcd.print("Humidity: ");
  lcd.print(hum, 0);
  lcd.print("%   ");

  lcd.setCursor(0, 1);
  lcd.print("L:");
  lcd.print(minHum, 0);
  lcd.print(" H:");
  lcd.print(maxHum, 0);
  lcd.print("    ");

  delay(2000);
}
```

What you should see: Humidity: 62% on the top line. L:58 H:67 on the second line. Both lines update every 2 seconds; breathe gently on the sensor and watch the high value climb.

RGB Mood Lamp

Goal: Turn a single knob and sweep smoothly through the whole colour spectrum on an RGB LED.

The potentiometer gives a number from 0 to 1023. That single number is stretched to a 0–764 colour wheel and then split into three stages: red fading into green, green into blue, and blue back into red. Each of the three channels is written with analogWrite(), which uses PWM, switching the LED on and off thousands of times a second so it appears dimmer or brighter rather than simply on or off.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-18-rgb-mood-lamp

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 5mm RGB LED (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 16mm Rotary Potentiometer (Linear)
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Analog input mapping to PWM output
- The map() function
- Mixing colours from three channels
- Why analogWrite() only works on PWM (~) pins

Wiring

Part	Pin	Nano	Note
RGB LED	Red	**D9**	Through a 220 ohm resistor
RGB LED	Green	**D10**	Through a 220 ohm resistor
RGB LED	Blue	**D11**	Through a 220 ohm resistor
RGB LED	Common cathode (−)	**GND**	
Potentiometer	Left pin	**GND**	
Potentiometer	Middle pin (wiper)	**A0**	
Potentiometer	Right pin	**5V**	

Code

```
int potPin = A0;
int redPin = 9;
int greenPin = 10;
int bluePin = 11;

void setup()
{
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop()
{
  int potValue = analogRead(potPin);
  int wheel = map(potValue, 0, 1023, 0, 764);

  int r = 0;
  int g = 0;
  int b = 0;

  if (wheel < 255)
  {
    r = 255 - wheel;
    g = wheel;
    b = 0;
  }
  else if (wheel < 510)
  {
    r = 0;
    g = 510 - wheel;
    b = wheel - 255;
  }
  else
  {
    r = wheel - 510;
    g = 0;
    b = 764 - wheel;
  }

  analogWrite(redPin, r);
  analogWrite(greenPin, g);
  analogWrite(bluePin, b);

  delay(20);
}
```

What you should see: Knob fully left: pure red. Turning slowly: red → orange → green → cyan → blue → magenta. Knob fully right: back to red.

Smart Street Light

Goal: Light a row of LEDs automatically when the surroundings get dark, and switch them off again at dawn, the logic behind every street lamp in your town.

An LDR is simply a resistor whose value falls when light hits it. On its own the Arduino cannot measure resistance, so the LDR is paired with a fixed 10kΩ resistor to form a voltage divider: as the light changes, the voltage at A0 changes with it. The threshold in the sketch is the decision point between day and night, read your own room's values in the Serial Monitor first and set it about halfway between the lit and the covered reading.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-19-smart-street-light

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 10k resistor
- 1 x 5 mm LDR Light Dependent Resistor
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- LDR thresholds and automation
- Voltage dividers, turning resistance into voltage
- Driving several outputs from one function
- Calibrating a sensor to its environment

Wiring

Part	Pin	Nano	Note
LDR divider	LDR leg 1	**5V**	
LDR divider	LDR leg 2	**A0 and one end of 10kΩ resistor**	
LDR divider	Other end of 10kΩ resistor	**GND**	
LED	LED 1 anode	**D2**	Through a 220 ohm resistor
LED	LED 2 anode	**D3**	Through a 220 ohm resistor
LED	LED 3 anode	**D4**	Through a 220 ohm resistor
LED	LED 4 anode	**D5**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

Code

```
int ldrPin = A0;
int ledPins[4] = {2, 3, 4, 5};
int threshold = 400;

void setup()
{
  for (int i = 0; i < 4; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
  Serial.begin(9600);
}

void setLamps(int state)
{
  for (int i = 0; i < 4; i++)
  {
    digitalWrite(ledPins[i], state);
  }
}

void loop()
{
  int lightLevel = analogRead(ldrPin);

  Serial.print("Light level: ");
  Serial.println(lightLevel);

  if (lightLevel < threshold)
  {
    setLamps(HIGH);       // Dark - street lights ON
  }
  else
  {
    setLamps(LOW);        // Bright - street lights OFF
  }

  delay(300);
}
```

What you should see: Serial Monitor prints Light level: 720 in a lit room. Cover the LDR with your hand and all four LEDs switch on together. Uncover it and they switch off again.

Smart Umbrella Reminder

Goal: Detect the first drops of rain and put a clear reminder on the LCD before anyone leaves the house.

The rain board's digital output is HIGH while the plate is dry and drops LOW the moment water bridges its tracks. The blue potentiometer on the comparator board sets how wet the plate must be before that happens, turn it until a single drop is enough. Notice that the sketch only redraws the LCD when the state actually changes; constantly clearing a display makes it flicker.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-20-umbrella-reminder

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic Raindrops Detection Sensor Module – LM393 Comparator, Analog/Digital Output
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Conditional logic driving a display
- Weather sensing with a digital module
- Remembering the previous state to avoid flicker
- Adjusting a comparator's sensitivity

Wiring

Part	Pin	Nano	Note
Rain sensor	VCC	**5V**	
Rain sensor	GND	**GND**	
Rain sensor	DO (digital out)	**D2**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**A4**	
LCD	SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int rainPin = 2;
LiquidCrystal_I2C lcd(0x27, 16, 2);

int lastState = -1;

void setup()
{
  pinMode(rainPin, INPUT);
  lcd.init();
  lcd.backlight();

  lcd.setCursor(0, 0);
  lcd.print("Weather Check");
  delay(2000);
  lcd.clear();
}

void loop()
{
  int rainState = digitalRead(rainPin);

  if (rainState != lastState)
  {
    lcd.clear();
    if (rainState == LOW)
    {
      lcd.setCursor(0, 0);
      lcd.print("RAIN DETECTED!");
      lcd.setCursor(0, 1);
      lcd.print("Take umbrella");
    }
    else
    {
      lcd.setCursor(0, 0);
      lcd.print("Weather: Clear");
      lcd.setCursor(0, 1);
      lcd.print("Have a nice day");
    }

    lastState = rainState;
  }

  delay(200);
}
```

What you should see: Dry plate: Weather: Clear / Have a nice day. Sprinkle a few drops of water: RAIN DETECTED! / Take umbrella. Dry the plate and the message returns to clear.

Soil Moisture Detector

Goal: Measure how much water is in the soil of a plant pot and report it on the LCD as a percentage with a plain- English status.

Wet soil conducts electricity better than dry soil, so the analog reading falls as the soil gets wetter. That is backwards from what a person expects, so map() is used with its input range reversed, 1023 becomes 0% and 300 becomes 100%. constrain() then clips anything outside that band so the display never shows 104%.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-21-soil-moisture-detector

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Capacitive Soil Moisture Sensor – 3.3V/5V Analog Output, PH2.54-3P Interface
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Analog sensor reading for soil
- Reversing a range with map()
- Keeping values in bounds with constrain()
- Turning a number into a readable status

Wiring

Part	Pin	Nano	Note
Soil sensor	VCC	**5V**	
Soil sensor	GND	**GND**	
Soil sensor	AO (analog out)	**A0**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**A4**	
LCD	SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int soilPin = A0;
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup()
{
  lcd.init();
  lcd.backlight();
  Serial.begin(9600);

  lcd.setCursor(0, 0);
  lcd.print("Soil Monitor");
  delay(2000);
  lcd.clear();
}

void loop()
{
  int raw = analogRead(soilPin);

  // Wet soil gives a LOW reading, so the range is reversed
  int moisture = map(raw, 1023, 300, 0, 100);
  moisture = constrain(moisture, 0, 100);

  Serial.print("Raw: ");
  Serial.print(raw);
  Serial.print(" Moisture: ");
  Serial.println(moisture);
  lcd.setCursor(0, 0);
  lcd.print("Moisture: ");
  lcd.print(moisture);
  lcd.print("%   ");

  lcd.setCursor(0, 1);
  if (moisture < 30)
  {
    lcd.print("Status: DRY                   ");
  }
  else if (moisture < 70)
  {
    lcd.print("Status: OK                    ");
  }
  else
  {
    lcd.print("Status: WET                   ");
  }

  delay(1000);
}
```

What you should see: Sensor in air: Moisture: 0% / Status: DRY. Sensor in damp soil: Moisture: 55% / Status: OK. Sensor in a glass of water: Moisture: 100% / Status: WET.

Automatic Plant Watering System

Goal: Combine the soil sensor with a relay and a pump so the plant waters itself whenever the soil dries out.

The relay is an electrically operated switch: a tiny current from D7 closes a pair of heavy contacts that carry the pump's much larger current. The Arduino and the pump therefore share no power at all, only a common ground reference. Most relay modules are active-LOW, meaning digitalWrite(relay, LOW) is what turns the pump on. Two thresholds are used rather than one: watering starts below 30% and stops above 60%, so the pump does not chatter on and off around a single boundary.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-22-plant-watering

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Capacitive Soil Moisture Sensor – 3.3V/5V Analog Output, PH2.54-3P Interface
- 1 x 5V Relay Module with Optocoupler
- 1 x Mini Submersible Water Pump 3–6V
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Relay control and threshold logic
- Hysteresis, two thresholds instead of one
- Powering a load from a separate supply
- Why a common ground is essential

Wiring

Part	Pin	Nano	Note
Soil sensor	VCC	**5V**	
Soil sensor	GND	**GND**	
Soil sensor	AO	**A0**	
Relay	VCC	**5V**	
Relay	GND	**GND**	
Relay	IN	**D7**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	5V jumper stays on: the board's regulator makes 5V
L298N	GND	**Battery (−) and Nano GND**	The common ground is required
Relay	COM	**L298N +5V**	The pump never runs from the Nano's own 5V
Relay	NO	**Pump red wire (+)**	
Pump	Black wire (−)	**L298N GND**	

The pump runs at 5V from the L298N's on-board regulator, which is fed by the 2 x 18650 pack. The relay switches that 5V; the Nano only drives the relay.

> Keep water away from the Nano and test the pump and tube over a tray first.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int soilPin = A0;
int relayPin = 7;

int dryLevel = 30;                // Start watering below this
int wetLevel = 60;                // Stop watering above this

bool pumpOn = false;
LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup()
{
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH);                    // Relay is active LOW - pump OFF

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Auto Watering");
  delay(2000);
  lcd.clear();
}

void loop()
{
  int raw = analogRead(soilPin);
  int moisture = map(raw, 1023, 300, 0, 100);
  moisture = constrain(moisture, 0, 100);

  if (moisture < dryLevel)
  {
    pumpOn = true;
  }
  if (moisture > wetLevel)
  {
    pumpOn = false;
  }

  if (pumpOn)
  {
    digitalWrite(relayPin, LOW);          // Pump ON
  }
  else
  {
    digitalWrite(relayPin, HIGH);         // Pump OFF
  }

  lcd.setCursor(0, 0);
  lcd.print("Moisture: ");
  lcd.print(moisture);
  lcd.print("%   ");

  lcd.setCursor(0, 1);
  if (pumpOn)
  {
    lcd.print("Pump: WATERING       ");
  }
  else
  {
    lcd.print("Pump: IDLE           ");
  }

  delay(1000);
}
```

What you should see: Damp soil: Moisture: 65% / Pump: IDLE. Soil dries below 30%: the relay clicks and Pump: WATERING appears. Once moisture passes 60% the relay clicks off again.

Water Tank Automation

Goal: Measure the water level in a tank with an ultrasonic sensor and switch a filling pump on and off automatically.

The sensor sends a burst of sound and measures how long the echo takes to return. Sound travels roughly 0.034 cm per microsecond, and the pulse makes the trip twice, so distance = duration × 0.034 ÷ 2. Mount the sensor facing down at the top of the tank: a large distance means the water is far away and the tank is empty, a small distance means it is nearly full.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-23-water-tank-automation

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x 5V Relay Module with Optocoupler
- 1 x Mini Submersible Water Pump 3–6V
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Ultrasonic distance measurement with pulseIn()
- Relay logic driven by a measured level
- Converting a distance into a percentage
- Writing your own function that returns a value

Wiring

Part	Pin	Nano	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**D9**	
HC-SR04	ECHO	**D10**	
Relay	VCC	**5V**	
Relay	GND	**GND**	
Relay	IN	**D7**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**A4**	
LCD	SCL	**A5**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	5V jumper stays on: the board's regulator makes 5V
L298N	GND	**Battery (−) and Nano GND**	The common ground is required
Relay	COM	**L298N +5V**	The pump never runs from the Nano's own 5V

Relay	NO	**Pump red wire (+)**	
Pump	Black wire (−)	**L298N GND**	

> The pump runs at 5V from the L298N's on-board regulator, which is fed by the 2 x 18650 pack. The relay switches that 5V; the Nano only drives the relay.

> Keep water away from the Nano and test the pump and tube over a tray first.

> Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int trigPin = 9;
int echoPin = 10;
int relayPin = 7;

int tankHeight = 30;                 // cm from sensor to the bottom

int fillBelow = 30;                // Start pump below this percent
int stopAbove = 90;                // Stop pump above this percent

bool pumpOn = false;

LiquidCrystal_I2C lcd(0x27, 16, 2);

long readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);
  long distance = duration * 0.034 / 2;
  return distance;
}

void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH);             // Pump OFF

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Tank Monitor");
  delay(2000);
  lcd.clear();
}

void loop()
{
  long distance = readDistance();

  int level = map(distance, tankHeight, 2, 0, 100);
  level = constrain(level, 0, 100);

  if (level < fillBelow)
  {
    pumpOn = true;
  }
  if (level > stopAbove)
  {
    pumpOn = false;
  }
  if (pumpOn)
  {
    digitalWrite(relayPin, LOW);
  }
  else
  {
    digitalWrite(relayPin, HIGH);
  }
  lcd.setCursor(0, 0);
  lcd.print("Level: ");
  lcd.print(level);
  lcd.print("%      ");

  lcd.setCursor(0, 1);
  if (pumpOn)
  {
```

```
    lcd.print("Pump: FILLING                ");
  }
  else
  {
    lcd.print("Pump: STOPPED                ");
  }

  delay(500);
}
```

What you should see: Level: 85% / Pump: STOPPED with the tank nearly full. Move your hand away from the sensor to simulate a falling level. Below 30% the relay clicks and Pump: FILLING appears.

Smart Temperature Alarm

Goal: Watch the room temperature and sound a loud buzzer the moment it crosses a limit you set.

The alarm is a threshold comparison: read a value, compare it with a limit, act. Rather than holding the buzzer permanently on, the sketch pulses it, which is both more attention-grabbing and easier on the ear. Change alarmTemp to a value just above your current room temperature and you can trigger the alarm by cupping the sensor in your hand.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-24-temperature-alarm

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Threshold-based environmental alarms
- Comparing float readings against a limit
- Pulsing an output instead of holding it on
- Combining a sensor, a display and two outputs

Wiring

Part	Pin	Nano	Note
DHT11	VCC	**5V**	
DHT11	GND	**GND**	
DHT11	DATA	**D2**	
Buzzer	Buzzer (+)	**D8**	
Buzzer	Buzzer (−)	**GND**	
LED	LED anode	**D6**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**A4**	
LCD	SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>

#define DHTPIN 2
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);

LiquidCrystal_I2C lcd(0x27, 16, 2);

int buzzer = 8;
int led = 6;
float alarmTemp = 32.0;

void setup()
{
  pinMode(buzzer, OUTPUT);
  pinMode(led, OUTPUT);
  dht.begin();

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Temp Alarm ON");
  delay(2000);
  lcd.clear();
}

void loop()
{
  float temp = dht.readTemperature();

  if (isnan(temp))
  {
    lcd.setCursor(0, 0);
    lcd.print("Sensor error          ");
    delay(2000);
    return;
  }

  lcd.setCursor(0, 0);
  lcd.print("Temp: ");
  lcd.print(temp, 1);
  lcd.write(223);
  lcd.print("C    ");

  if (temp > alarmTemp)
  {
    lcd.setCursor(0, 1);
    lcd.print("** TOO HOT! **         ");

    for (int i = 0; i < 4; i++)
    {
      digitalWrite(buzzer, HIGH);
      digitalWrite(led, HIGH);
      delay(150);
      digitalWrite(buzzer, LOW);
      digitalWrite(led, LOW);
      delay(150);
    }
  }
  else
  {
    lcd.setCursor(0, 1);
    lcd.print("Status: Normal ");
    digitalWrite(buzzer, LOW);
    digitalWrite(led, LOW);
    delay(1500);
  }
}
```

What you should see: Temp: 28.5°C / Status: Normal in an ordinary room. Hold the sensor in a closed hand for a few seconds. Above 32°C: ** TOO HOT! ** with a pulsing buzzer and flashing LED.

RFID Door Lock

Goal: Scan an RFID card and let a servo motor open the lock only for cards whose UID you have authorised.

The reader talks to the Arduino over SPI, a four-wire bus that shares MOSI, MISO and SCK between devices and uses a separate SS pin to pick which one is listening. Every card carries a factory-programmed UID, a unique serial number. The sketch turns that UID into text and compares it with an allowed value; matching UIDs is the whole of the security here. Run the sketch once, scan your card, read the printed UID from the Serial Monitor, then paste it into allowedUID and upload again.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-25-rfid-door-lock

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- SPI communication and shared buses
- UID matching as a form of authentication
- Driving a servo to a fixed angle
- Building a system from three modules at once silent alarms: eight ways to decide who and what gets through.

Wiring

Part	Pin	Nano	Note
RC522	3.3V	**3V3 pin**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**D9**	
RC522	SDA / SS	**D10**	
RC522	MOSI	**D11**	
RC522	MISO	**D12**	
RC522	SCK	**D13**	
Servo	Brown / black wire	**GND**	
Servo	Red wire	**5V**	
Servo	Orange / yellow wire	**D6**	
LCD	VCC	**5V**	
LCD	GND	**GND**	
LCD	SDA	**A4**	
LCD	SCL	**A5**	

> Power the RC522 from 3.3V only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#define SS_PIN 10

#define RST_PIN 9

MFRC522 rfid(SS_PIN, RST_PIN);
Servo lock;
LiquidCrystal_I2C lcd(0x27, 16, 2);

String allowedUID = "A1 B2 C3 D4";              // Replace with your own card

void setup()
{
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();

  lock.attach(6);
  lock.write(0);              // Locked position

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Scan your card");
}

void loop()
{
  if (!rfid.PICC_IsNewCardPresent())
  {
    return;
  }
  if (!rfid.PICC_ReadCardSerial())
  {
    return;
  }

  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10)
    {
      uid = uid + "0";
    }
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1)
    {
      uid = uid + " ";
    }
  }
  uid.toUpperCase();
  Serial.print("Card UID: ");
  Serial.println(uid);

  lcd.clear();

  if (uid == allowedUID)
  {
    lcd.setCursor(0, 0);
    lcd.print("ACCESS GRANTED");
    lock.write(90);       // Unlocked
    delay(3000);
    lock.write(0);        // Locked again
  }
  else
  {
    lcd.setCursor(0, 0);
    lcd.print("ACCESS DENIED");
    delay(2000);
  }
```

```
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Scan your card");

  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: LCD shows Scan your card. Serial Monitor prints Card UID: A1 B2 C3 D4 whenever a card is tapped. Authorised card: ACCESS GRANTED and the servo turns to 90°, returning after 3 seconds. Any other card: ACCESS DENIED and the servo does not move.

Password Door Lock

Goal: Enter a PIN on a 4×4 keypad and let a servo open the door only when the code is correct.

A matrix keypad has sixteen buttons but only eight wires. The library drives one row LOW at a time and reads all four columns, where a LOW appears, that button is down. This is keypad matrix scanning, the same trick the LED matrix used in reverse. Each key press is added to a String; pressing # compares that String with the stored password, and * clears it so a mistake can be retyped.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-26-password-door-lock

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x Membrane Matrix Keypad
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Keypad matrix scanning
- String comparison for authentication
- Masking an entry with * characters
- Using * and # as control keys

Wiring

Part	Pin	Nano	Note
Keypad	Pin 1 (Row 1)	**D9**	
Keypad	Pin 2 (Row 2)	**D8**	
Keypad	Pin 3 (Row 3)	**D7**	
Keypad	Pin 4 (Row 4)	**D6**	
Keypad	Pin 5 (Col 1)	**D5**	
Keypad	Pin 6 (Col 2)	**D4**	
Keypad	Pin 7 (Col 3)	**D3**	
Keypad	Pin 8 (Col 4)	**D2**	
Servo	Servo signal (orange)	**D11**	
Servo	Servo red	**5V**	
Servo	Servo brown	**GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

Code

```
#include <Keypad.h>
#include <Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

const byte ROWS = 4;
const byte COLS = 4;

char keys[ROWS][COLS] =
{
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
Servo lock;
LiquidCrystal_I2C lcd(0x27, 16, 2);

String password = "1234";
String entered = "";

void askForCode()
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Enter PIN:");
  entered = "";
}

void setup()
{
  lock.attach(11);
  lock.write(0);
  lcd.init();
  lcd.backlight();
  askForCode();
}

void loop()
{
  char key = keypad.getKey();

  if (key == NO_KEY)
  {
    return;
  }

  if (key == '*')
  {
    askForCode();
    return;
  }

  if (key == '#')
  {
    lcd.clear();
    lcd.setCursor(0, 0);

    if (entered == password)
    {
      lcd.print("ACCESS GRANTED");
      lock.write(90);
      delay(3000);
      lock.write(0);
    }
    else
    {
      lcd.print("WRONG PIN");
      delay(2000);
    }
```

```
    askForCode();
    return;
  }

  entered = entered + key;
  lcd.setCursor(0, 1);
  for (unsigned int i = 0; i < entered.length(); i++)
  {
    lcd.print("*");
  }
}
```

What you should see: LCD shows Enter PIN:. Each key press adds a * on the second line. Press # after 1234: ACCESS GRANTED and the servo turns for 3 seconds. Any other code: WRONG PIN.

Bluetooth Door Lock

Goal: Unlock the servo latch from a phone by sending a single character over a Bluetooth link.

Bluetooth here is nothing more exotic than a wireless serial cable. The HC-05 receives characters from a phone app and hands them to the Arduino over UART, the same one-byte-at-a-time protocol the Serial Monitor uses. SoftwareSerial creates a second UART on ordinary digital pins so the USB Serial Monitor stays free for debugging.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-27-bluetooth-door-lock

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x 1k and 2.2k resistors
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- UART and Serial communication
- SoftwareSerial on ordinary digital pins
- Parsing single-character commands
- Two-way messaging with a phone

Wiring

Part	Pin	Nano	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**D2 (Nano RX)**	
HC-05	RXD	**D3 (Nano TX)**	Through a divider: 1k from D3 to RXD and 2.2k from RXD to GND, because RXD is a 3.3V input
Servo	Servo signal	**D6**	
Servo	Servo red	**5V**	
Servo	Servo brown	**GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

Code

```
#include <SoftwareSerial.h>
#include <Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

SoftwareSerial bt(2, 3);      // RX, TX
Servo lock;
LiquidCrystal_I2C lcd(0x27, 16, 2);

bool locked = true;

void setup()
{
  bt.begin(9600);
  Serial.begin(9600);
  lock.attach(6);
  lock.write(0);

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Status: LOCKED");
}

void loop()
{
  if (!bt.available())
  {
    return;
  }

  char command = bt.read();
  Serial.println(command);

  if (command == 'U' || command == 'u')
  {
    lock.write(90);
    locked = false;
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Status: UNLOCKED");
    bt.println("Door unlocked");
  }
  else if (command == 'L' || command == 'l')
  {
    lock.write(0);
    locked = true;
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Status: LOCKED");
    bt.println("Door locked");
  }
}
```

What you should see: Pair with the HC-05 (default PIN 1234 or 0000) in any serial terminal app. Send U: the servo turns and the LCD reads Status: UNLOCKED. Send L: the servo returns and the LCD reads Status: LOCKED. The phone receives a confirmation message back.

Motion Detection Alarm

Goal: Use the ultrasonic sensor as a tripwire: if the distance in front of it changes suddenly, sound the alarm.

A plain distance threshold would fire at anything closer than, say, 50 cm, including a wall. Instead the sketch records a baseline when it starts and then watches for change. If a new reading differs from the baseline by more than the tolerance, something has moved into the beam. abs() is used so the alarm reacts whether the object came closer or moved away.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-28-motion-alarm

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Ultrasonic distance thresholds
- Calibrating a baseline at startup
- Detecting change rather than absolute value
- Using abs() for two-sided comparison

Wiring

Part	Pin	Nano	Note
HC-SR04	VCC	**5V**	
HC-SR04	GND	**GND**	
HC-SR04	TRIG	**D9**	
HC-SR04	ECHO	**D10**	
Buzzer	Buzzer (+)	**D8**	
Buzzer	Buzzer (−)	**GND**	
LED	LED anode	**D7**	Through a 220 ohm resistor
LED	LED cathode	**GND**	

Code

```
int trigPin = 9;
int echoPin = 10;
int buzzer = 8;
int led = 7;

long baseline = 0;
int tolerance = 15;            // cm of change that counts as movement

long readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH);
  return duration * 0.034 / 2;
}

void setup()
{
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(led, OUTPUT);
  Serial.begin(9600);

  delay(2000);               // Stand clear while it calibrates
  baseline = readDistance();

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

void loop()
{
  long distance = readDistance();
  long change = abs(distance - baseline);

  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.print(" Change: ");
  Serial.println(change);

  if (change > tolerance)
  {
    for (int i = 0; i < 5; i++)
    {
      digitalWrite(buzzer, HIGH);
      digitalWrite(led, HIGH);
      delay(120);
      digitalWrite(buzzer, LOW);
      digitalWrite(led, LOW);
      delay(120);
    }
  }

  delay(150);
}
```

What you should see: Serial Monitor prints the baseline once at startup. Distance and Change print continuously. Wave your hand across the beam: the buzzer pulses five times and the LED flashes.

Touch Password System

Goal: Authenticate with a secret sequence: touch three capacitive pads in the right order to unlock.

The three sensors are stored in an array, which lets one for loop check all of them however many there are. Each accepted touch is appended to a sequence array and compared against the secret. Because a finger rests on a pad for far longer than one loop pass, the sketch waits for the pad to be released before accepting the next touch, without that, a single tap would register dozens of times.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-29-touch-password

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 3 x Capacitive Touch Sensor Module
- 1 x PCB Mounted Active Buzzer Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Capacitive touch arrays and logic
- Comparing two arrays element by element
- Waiting for release to debounce a touch
- Storing a secret as data rather than as code

Wiring

Part	Pin	Nano	Note
Touch module	All VCC	**5V**	
Touch module	All GND	**GND**	
Touch module	Sensor 1 SIG	**D2**	
Touch module	Sensor 2 SIG	**D3**	
Touch module	Sensor 3 SIG	**D4**	
LED	Green LED anode	**D5**	Through a 220 ohm resistor
LED	Red LED anode	**D6**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Buzzer	Buzzer (+)	**D8**	
Buzzer	Buzzer (−)	**GND**	

Code

```
int touchPins[3] = {2, 3, 4};
int greenLed = 5;
int redLed = 6;
int buzzer = 8;

int secret[3] = {1, 3, 2};          // Touch sensor 1, then 3, then 2
int entered[3];
int position = 0;

void beep(int ms)
{
  digitalWrite(buzzer, HIGH);
  delay(ms);
  digitalWrite(buzzer, LOW);
}

void setup()
{
  for (int i = 0; i < 3; i++)
  {
    pinMode(touchPins[i], INPUT);
  }
  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);
  Serial.begin(9600);
  Serial.println("Enter the touch sequence");
}

void loop()
{
  for (int i = 0; i < 3; i++)
  {
    if (digitalRead(touchPins[i]) == HIGH)
    {
      entered[position] = i + 1;
      position = position + 1;

      Serial.print("Touched sensor ");
      Serial.println(i + 1);
      beep(60);

      // Wait until the finger is lifted
      while (digitalRead(touchPins[i]) == HIGH)
      {
        delay(10);
      }
    }
  }

  if (position == 3)
  {
    bool correct = true;
    for (int i = 0; i < 3; i++)
    {
      if (entered[i] != secret[i])
      {
        correct = false;
      }
    }

    if (correct)
    {
      Serial.println("ACCESS GRANTED");
      digitalWrite(greenLed, HIGH);
      beep(400);
      delay(2000);
      digitalWrite(greenLed, LOW);
    }
    else
    {
      Serial.println("ACCESS DENIED");
      digitalWrite(redLed, HIGH);
      beep(150);
      delay(150);
```

```
      beep(150);
      delay(1500);
      digitalWrite(redLed, LOW);
    }
    position = 0;
  }
}
```

What you should see: Serial Monitor prints Touched sensor 1 with a short beep on each pad. Correct order 1-3-2: ACCESS GRANTED, green LED and a long beep. Any wrong order: ACCESS DENIED, red LED and two short beeps.

Smoke Detection Alarm

Goal: Read an MQ2 gas sensor and raise an audible alarm when smoke or combustible gas appears.

Inside the MQ2 is a tin-dioxide element whose resistance falls when certain gases touch it. The module converts that into a voltage on AO, so the reading climbs as the air gets dirtier. The sensor's heater needs about twenty seconds to stabilise after power-up, which is why the sketch shows a warming-up message before it starts judging anything.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-30-smoke-alarm

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic MQ-2 Smoke LPG Butane Hydrogen Gas Sensor Module – 5V Analog/Digital Output
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Gas sensor analog reading
- Why chemical sensors need a warm-up period
- Setting an alarm level from observed values
- Displaying a live number alongside a status

Wiring

Part	Pin	Nano	Note
MQ-2	VCC	**5V**	
MQ-2	GND	**GND**	
MQ-2	AO (analog out)	**A0**	
Buzzer	Buzzer (+)	**D8**	
Buzzer	Buzzer (−)	**GND**	
LED	LED anode	**D7**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int gasPin = A0;
int buzzer = 8;
int led = 7;
int alarmLevel = 400;

LiquidCrystal_I2C lcd(0x27, 16, 2);

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

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Warming up...");

  for (int i = 20; i > 0; i--)
  {
    lcd.setCursor(0, 1);
    lcd.print(i);
    lcd.print(" s     ");
    delay(1000);
  }

  lcd.clear();
}

void loop()
{
  int gasLevel = analogRead(gasPin);

  Serial.print("Gas level: ");
  Serial.println(gasLevel);

  lcd.setCursor(0, 0);
  lcd.print("Gas: ");
  lcd.print(gasLevel);
  lcd.print("     ");

  lcd.setCursor(0, 1);

  if (gasLevel > alarmLevel)
  {
    lcd.print("!! SMOKE ALARM !!");
    digitalWrite(buzzer, HIGH);
    digitalWrite(led, HIGH);
    delay(200);
    digitalWrite(buzzer, LOW);
    digitalWrite(led, LOW);
    delay(200);
  }
  else
  {
    lcd.print("Air: Clean      ");
    digitalWrite(buzzer, LOW);
    digitalWrite(led, LOW);
    delay(500);
  }
}
```

What you should see: A 20-second countdown while the sensor heats up. Gas: 180 / Air: Clean in normal room air. Hold a blown-out match near the sensor: the value jumps and !! SMOKE ALARM !! appears with a pulsing buzzer.

Fire Warning System

Goal: Extend the smoke alarm into a full safety loop: buzzer, warning lamp and a relay that can cut power or start an exhaust fan.

Real safety systems latch: once triggered they stay triggered until a person resets them, because a fire that briefly dips below the threshold is still a fire. The alarmActive flag does exactly that. Note also that the relay stays energised for the whole alarm rather than pulsing, the buzzer flashes for attention, but the exhaust fan must run continuously.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-31-fire-warning

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x 5V Relay Module with Optocoupler
- 1 x Generic MQ-2 Smoke LPG Butane Hydrogen Gas Sensor Module – 5V Analog/Digital Output
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Multi-component safety loops
- Latching an alarm so it cannot silently clear
- Driving a relay alongside indicators
- Separating continuous outputs from flashing ones

Wiring

Part	Pin	Nano	Note
MQ-2	MQ2 VCC	**5V**	
MQ-2	MQ2 GND	**GND**	
MQ-2	MQ2 AO	**A0**	
Relay	Relay VCC	**5V**	
Relay	Relay GND	**GND**	
Relay	Relay IN	**D9**	
Buzzer	Buzzer (+)	**D8**	
LED	Red LED anode	**D7**	Through a 220 ohm resistor
LED	Green LED anode	**D6**	Through a 220 ohm resistor
Buzzer	All cathodes / buzzer (−)	**GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	
Relay	COM	**Nano 5V**	Demo load only: never mains
Lamp LED	Anode	**Relay NO**	Through a 220 ohm resistor
Lamp LED	Cathode	**GND**	

> Switch only low-voltage demo loads such as the LED here. Never wire mains to these relay modules in a classroom project.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int gasPin = A0;
int relayPin = 9;
int buzzer = 8;
int redLed = 7;
int greenLed = 6;

int alarmLevel = 400;
bool alarmActive = false;

LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup()
{
  pinMode(relayPin, OUTPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(greenLed, OUTPUT);

  digitalWrite(relayPin, HIGH);       // Active LOW relay - load OFF
  Serial.begin(9600);

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Fire System");
  lcd.setCursor(0, 1);
  lcd.print("Warming up...");
  delay(20000);
  lcd.clear();
}

void loop()
{
  int gasLevel = analogRead(gasPin);
  Serial.println(gasLevel);

  if (gasLevel > alarmLevel)
  {
    alarmActive = true;
  }

  lcd.setCursor(0, 0);
  lcd.print("Gas: ");
  lcd.print(gasLevel);
  lcd.print("     ");
  lcd.setCursor(0, 1);

  if (alarmActive)
  {
    lcd.print("FIRE! EVACUATE ");
    digitalWrite(relayPin, LOW);             // Exhaust fan ON
    digitalWrite(greenLed, LOW);

    digitalWrite(buzzer, HIGH);
    digitalWrite(redLed, HIGH);
    delay(250);
    digitalWrite(buzzer, LOW);
    digitalWrite(redLed, LOW);
    delay(250);
  }
  else
  {
    lcd.print("Status: SAFE    ");
    digitalWrite(relayPin, HIGH);
    digitalWrite(greenLed, HIGH);
    digitalWrite(redLed, LOW);
    digitalWrite(buzzer, LOW);
    delay(500);
  }
}
```

What you should see: Gas: 190 / Status: SAFE with the green LED steady. Introduce smoke: FIRE! EVACUATE, the relay clicks in, the buzzer and red LED pulse. The alarm stays latched until you press the Arduino's reset button.

Smart Locker System

Goal: Combine everything from the door lock into a locker that accepts several registered cards and reports each attempt on the LCD.

The single allowed UID from Experiment 25 becomes an array, and a for loop walks it looking for a match. Adding a fourth locker user is now a one-line change rather than a rewrite. The array of names alongside the array of UIDs is a common pattern called parallel arrays: index 0 in one lines up with index 0 in the other.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-32-smart-locker

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x RC522 RFID Reader Writer SPI Module with Card & Tag
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- RFID, LCD and servo working together
- Parallel arrays for UIDs and names
- Searching an array with a loop
- Building a system with three feedback channels

Wiring

Part	Pin	Nano	Note
RC522	3.3V	**3V3 pin**	Never 5V: the RC522 is a 3.3V board
RC522	GND	**GND**	
RC522	RST	**D9**	
RC522	SDA / SS	**D10**	
RC522	MOSI	**D11**	
RC522	MISO	**D12**	
RC522	SCK	**D13**	
Servo	Servo signal	**D6**	
LED	Green LED anode	**D5**	Through a 220 ohm resistor
LED	Red LED anode	**D4**	Through a 220 ohm resistor
Buzzer	Buzzer (+)	**D3**	
Buzzer	All cathodes / buzzer (−)	**GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

> Power the RC522 from 3.3V only. 5V on its VCC pin destroys it.

Code

```
#include <SPI.h>
#include <MFRC522.h>
#include <Servo.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#define SS_PIN 10
#define RST_PIN 9
#define USERS 3

MFRC522 rfid(SS_PIN, RST_PIN);
Servo locker;
LiquidCrystal_I2C lcd(0x27, 16, 2);

int greenLed = 5;
int redLed = 4;
int buzzer = 3;

String allowedUID[USERS] = {"A1 B2 C3 D4", "11 22 33 44", "DE AD BE EF"};
String userName[USERS]   = {"ANJALI", "RAHUL", "PRIYA"};

String readUID()
{
  String uid = "";
  for (byte i = 0; i < rfid.uid.size; i++)
  {
    if (rfid.uid.uidByte[i] < 0x10)
    {
      uid = uid + "0";
    }
    uid = uid + String(rfid.uid.uidByte[i], HEX);
    if (i < rfid.uid.size - 1)
    {
      uid = uid + " ";
    }
  }
  uid.toUpperCase();
  return uid;
}

void ready()
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Locker locked");
  lcd.setCursor(0, 1);
  lcd.print("Tap your card");
}

void setup()
{
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();

  pinMode(greenLed, OUTPUT);
  pinMode(redLed, OUTPUT);
  pinMode(buzzer, OUTPUT);

  locker.attach(6);
  locker.write(0);

  lcd.init();
  lcd.backlight();
  ready();
}

void loop()
{
  if (!rfid.PICC_IsNewCardPresent())
  {
    return;
  }
  if (!rfid.PICC_ReadCardSerial())
  {
```

```
    return;
  }

  String uid = readUID();
  Serial.print("Card UID: ");
  Serial.println(uid);

  int found = -1;
  for (int i = 0; i < USERS; i++)
  {
    if (uid == allowedUID[i])
    {
      found = i;
    }
  }

  lcd.clear();

  if (found >= 0)
  {
    lcd.setCursor(0, 0);
    lcd.print("Welcome");
    lcd.setCursor(0, 1);
    lcd.print(userName[found]);

    digitalWrite(greenLed, HIGH);
    digitalWrite(buzzer, HIGH);
    delay(120);
    digitalWrite(buzzer, LOW);

    locker.write(90);
    delay(4000);
    locker.write(0);
    digitalWrite(greenLed, LOW);
  }
  else
  {
    lcd.setCursor(0, 0);
    lcd.print("CARD NOT FOUND");
    digitalWrite(redLed, HIGH);
    for (int i = 0; i < 2; i++)
    {
      digitalWrite(buzzer, HIGH);
      delay(150);
      digitalWrite(buzzer, LOW);
      delay(150);
    }
    delay(1500);
    digitalWrite(redLed, LOW);
  }

  ready();
  rfid.PICC_HaltA();
  rfid.PCD_StopCrypto1();
}
```

What you should see: LCD shows Locker locked / Tap your card. A registered card: Welcome ANJALI, green LED, locker opens for 4 seconds. An unknown card: CARD NOT FOUND, red LED and two beeps.

Silent Security Alert

Goal: Detect someone crossing an IR beam and raise a silent visual flag that stays lit until it is acknowledged.

The IR module holds its output HIGH while the beam is clear and pulls it LOW when something reflects the infrared light back. An intruder passing through takes a fraction of a second, far too fast to notice if you only glance at the LEDs. The intruderSeen flag captures that instant and keeps it, which is why the alert is still there when you return to the room. Silence is deliberate: nothing warns the intruder that they were detected.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-33-silent-security-alert

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x Generic Infrared Obstacle Avoidance Sensor Module (Active Low) – LM393 Comparator, 2-30cm Range
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- IR digital sensing
- Silent flags that latch an event
- Counting events across the whole session
- Acknowledging an alert with a button

Wiring

Part	Pin	Nano	Note
IR sensor	VCC	**5V**	
IR sensor	GND	**GND**	
IR sensor	OUT	**D2**	
LED	Red LED anode	**D7**	Through a 220 ohm resistor
LED	Green LED anode	**D6**	Through a 220 ohm resistor
LED	Both cathodes	**GND**	
Button	Button one leg	**D5**	
Button	Button other leg	**GND**	

Code

```
int irPin = 2;
int redLed = 7;
int greenLed = 6;
int resetButton = 5;

bool intruderSeen = false;
int eventCount = 0;

void setup()
{
  pinMode(irPin, INPUT);
  pinMode(redLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(resetButton, INPUT_PULLUP);

  Serial.begin(9600);
  Serial.println("Silent monitoring started");
}

void loop()

{
  if (digitalRead(irPin) == LOW && !intruderSeen)
  {
    intruderSeen = true;
    eventCount = eventCount + 1;

    Serial.print("Intruder detected. Event number ");
    Serial.println(eventCount);
  }

  if (intruderSeen)
  {
    digitalWrite(redLed, HIGH);
    digitalWrite(greenLed, LOW);
  }
  else
  {
    digitalWrite(redLed, LOW);
    digitalWrite(greenLed, HIGH);
  }

  if (digitalRead(resetButton) == LOW)
  {
    intruderSeen = false;
    Serial.println("Alert acknowledged and cleared");
    delay(400);
  }

  delay(50);
}
```

What you should see: Green LED steady while the area is clear. Pass a hand in front of the sensor: the red LED latches on with no sound at all. Serial Monitor logs Intruder detected. Event number 1. Press the reset button to clear the flag and return to green.

Bluetooth Controlled Car

Goal: Drive a two-motor chassis from your phone, forward, back, left, right and stop, over a Bluetooth link.

The L298N is an H-bridge: two transistor pairs per motor that let current flow either way, which is what makes reverse possible. IN1/IN2 set the direction of the left motor and IN3/IN4 the right; ENA and ENB take a PWM value that sets the speed. Turning is done by driving the two sides in opposite directions, which spins the car on the spot. Every phone command is a single letter, so the whole protocol fits in a switch-like chain of ifs.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-34-bluetooth-car

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x 1k and 2.2k resistors
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Motor drivers and H-bridge direction control
- Serial commands as a control protocol
- Writing small helper functions for readability
- Setting motor speed with PWM

Wiring

Part	Pin	Nano	Note
L298N	ENA	**D9**	Pull the ENA jumper cap off first, so the Nano sets the speed
L298N	IN1	**D4**	
L298N	IN2	**D5**	
L298N	IN3	**D6**	
L298N	IN4	**D7**	
L298N	ENB	**D10**	Pull the ENB jumper cap off first, so the Nano sets the speed
L298N	+12V	**Battery (+)**	
L298N	GND	**Battery (−) and Nano GND**	
L298N	+5V	**Nano 5V (jumper on board fitted)**	
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**D2 (Nano RX)**	
HC-05	RXD	**D3 (Nano TX)**	Through a divider: 1k from D3 to RXD and 2.2k from RXD to GND, because RXD is a 3.3V input
Motors	Left motor wires	**OUT1 and OUT2**	

Motors	Right motor wires	**OUT3 and OUT4**	

Motors run from the 2 x 18650 pack (7.4V) through the L298N. The USB cable powers the Nano while you test; unplugged, the L298N's 5V output powers it.

Switch the battery holder off before you change any wiring, and lift the wheels off the table for the first test.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <SoftwareSerial.h>

SoftwareSerial bt(2, 3);                    // RX, TX

int ENA = 9;

int IN1 = 4;
int IN2 = 5;
int IN3 = 6;
int IN4 = 7;
int ENB = 10;

int speedValue = 180;

void drive(int l1, int l2, int r1, int r2)
{
  digitalWrite(IN1, l1);
  digitalWrite(IN2, l2);
  digitalWrite(IN3, r1);
  digitalWrite(IN4, r2);
  analogWrite(ENA, speedValue);
  analogWrite(ENB, speedValue);
}

void forward() { drive(HIGH, LOW, HIGH, LOW ); }
void backward() { drive(LOW, HIGH, LOW, HIGH); }
void turnLeft() { drive(LOW, HIGH, HIGH, LOW ); }
void turnRight(){ drive(HIGH, LOW, LOW, HIGH); }

void stopCar()
{
  drive(LOW, LOW, LOW, LOW);
  analogWrite(ENA, 0);
  analogWrite(ENB, 0);
}

void setup()
{
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);

  bt.begin(9600);
  Serial.begin(9600);
  stopCar();
}

void loop()
{
  if (!bt.available())
  {
    return;
  }

  char command = bt.read();
  Serial.println(command);

  if (command == 'F') forward();
  else if (command == 'B') backward();
  else if (command == 'L') turnLeft();
  else if (command == 'R') turnRight();
  else if (command == 'S') stopCar();
  else if (command == '+') speedValue = min(speedValue + 25, 255);
  else if (command == '-') speedValue = max(speedValue - 25, 80);
}
```

What you should see: Send F and the car drives forward. Send L or R and it spins on the spot. Send S and it stops. Send + or − to change speed between 80 and 255.

Obstacle Avoiding Robot

Goal: Let the car find its own way: drive forward until something blocks the path, then look both ways and turn towards the clearer side.

The servo carries the ultrasonic sensor like a head. When the robot meets an obstacle it stops, reverses a little, then turns the head fully left and fully right, taking one reading each way. Whichever side reports the greater distance is the side with more room, so that is the way it turns. This look-then-decide pattern is the simplest form of autonomous navigation.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-35-obstacle-avoiding-robot

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x TowerPro SG90 9g Micro Servo (180°)
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Autonomous navigation logic
- Mounting a sensor on a servo to scan
- Comparing two readings to choose an action
- Using a pulseIn() timeout so a missed echo does not stall the robot

Wiring

Part	Pin	Nano	Note
L298N	ENA	**D9**	Pull the ENA jumper cap off first, so the Nano sets the speed
L298N	IN1	**D4**	
L298N	IN2	**D5**	
L298N	IN3	**D6**	
L298N	IN4	**D7**	
L298N	ENB	**D10**	Pull the ENB jumper cap off first, so the Nano sets the speed
HC-SR04	HC-SR04 TRIG	**D11**	
HC-SR04	HC-SR04 ECHO	**D12**	
HC-SR04	HC-SR04 VCC / GND	**5V / GND**	
Servo	Servo signal	**D3**	
Servo	Servo red / brown	**5V / GND**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and Nano GND**	The common ground is required
L298N	+5V	**Nano 5V**	Powers the Nano when the USB cable is out (5V jumper stays on)

Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	

Motors run from the 2 x 18650 pack (7.4V) through the L298N. The USB cable powers the Nano while you test; unplugged, the L298N's 5V output powers it.

Switch the battery holder off before you change any wiring, and lift the wheels off the table for the first test.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <Servo.h>

int ENA = 9, IN1 = 4, IN2 = 5, IN3 = 6, IN4 = 7, ENB = 10;
int trigPin = 11;
int echoPin = 12;

Servo head;
int speedValue = 160;
int safeDistance = 25;                  // cm

void drive(int l1, int l2, int r1, int r2)
{
  digitalWrite(IN1, l1); digitalWrite(IN2, l2);
  digitalWrite(IN3, r1); digitalWrite(IN4, r2);
  analogWrite(ENA, speedValue);
  analogWrite(ENB, speedValue);
}

void forward() { drive(HIGH, LOW, HIGH, LOW ); }
void backward() { drive(LOW, HIGH, LOW, HIGH); }

void turnLeft() { drive(LOW, HIGH, HIGH, LOW ); }
void turnRight(){ drive(HIGH, LOW, LOW, HIGH); }
void stopCar() { drive(LOW, LOW, LOW, LOW); analogWrite(ENA, 0); analogWrite(ENB,
  0); }

long readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH, 25000);
  if (duration == 0)
  {
    return 200;        // Nothing in range
  }
  return duration * 0.034 / 2;
}

long lookAt(int angle)
{
  head.write(angle);
  delay(500);
  long d = readDistance();
  head.write(90);
  delay(300);
  return d;
}

void setup()
{
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  head.attach(3);
  head.write(90);
  Serial.begin(9600);
  delay(2000);
}

void loop()
{
  long distance = readDistance();
  Serial.println(distance);

  if (distance > safeDistance)
  {
    forward();
    delay(60);
```

```
    return;
  }

  stopCar();
  delay(200);

  backward();
  delay(400);
  stopCar();
  delay(200);

  long leftSpace = lookAt(160);
  long rightSpace = lookAt(20);

  if (leftSpace > rightSpace)
  {
    turnLeft();
  }
  else
  {
    turnRight();
  }
  delay(500);
  stopCar();
}
```

What you should see: The robot drives forward across open floor. Within 25 cm of a wall it stops and backs up. The sensor head sweeps left and right. It turns towards the open side and continues.

Line Following Robot

Goal: Follow a black line on a white floor using two IR sensors, correcting continuously as the robot drifts.

A white surface reflects infrared strongly and reads LOW on these modules; black tape absorbs it and reads HIGH. Mount the two sensors so the line runs between them. If both see white, the robot is centred and drives straight. If the left sensor finds black, the robot has drifted right, so it turns left, and vice versa. There is no memory and no planning here, just constant small corrections, which is exactly how a following robot stays on track.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-36-line-following-robot

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 2 x Generic TCRT5000 IR Tracking Sensor Module – Digital Output, 5V
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- IR thresholding on a reflective surface
- Continuous correction rather than planned paths
- Reading two sensors as a combined state
- Tuning speed so corrections are not violent

Wiring

Part	Pin	Nano	Note
L298N	ENA	**D9**	Pull the ENA jumper cap off first, so the Nano sets the speed
L298N	IN1	**D4**	
L298N	IN2	**D5**	
L298N	IN3	**D6**	
L298N	IN4	**D7**	
L298N	ENB	**D10**	Pull the ENB jumper cap off first, so the Nano sets the speed
Line sensor	Left sensor OUT	**D2**	
Line sensor	Right sensor OUT	**D3**	
L298N	Both VCC	**5V**	
L298N	Both GND	**GND**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and Nano GND**	The common ground is required
L298N	+5V	**Nano 5V**	Powers the Nano when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards

Motors	Right motor wires	**OUT3 and OUT4**	

Motors run from the 2 x 18650 pack (7.4V) through the L298N. The USB cable powers the Nano while you test; unplugged, the L298N's 5V output powers it.

Switch the battery holder off before you change any wiring, and lift the wheels off the table for the first test.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
int ENA = 9, IN1 = 4, IN2 = 5, IN3 = 6, IN4 = 7, ENB = 10;
int leftSensor = 2;
int rightSensor = 3;

int driveSpeed = 130;
int turnSpeed = 150;

void motors(int l1, int l2, int r1, int r2, int ls, int rs)
{
  digitalWrite(IN1, l1); digitalWrite(IN2, l2);
  digitalWrite(IN3, r1); digitalWrite(IN4, r2);
  analogWrite(ENA, ls);
  analogWrite(ENB, rs);
}

void setup()
{
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(leftSensor, INPUT);
  pinMode(rightSensor, INPUT);

  Serial.begin(9600);
  delay(2000);
}

void loop()
{
  int left = digitalRead(leftSensor);
  int right = digitalRead(rightSensor);

  Serial.print(left);
  Serial.print(" ");
  Serial.println(right);

  if (left == LOW && right == LOW)
  {
    // Both on white - go straight
    motors(HIGH, LOW, HIGH, LOW, driveSpeed, driveSpeed);
  }
  else if (left == HIGH && right == LOW)
  {
    // Drifted right - steer left
    motors(LOW, HIGH, HIGH, LOW, turnSpeed, turnSpeed);
  }
  else if (left == LOW && right == HIGH)
  {
    // Drifted left - steer right
    motors(HIGH, LOW, LOW, HIGH, turnSpeed, turnSpeed);
  }
  else
  {
    // Both on black - junction or end of line
    motors(LOW, LOW, LOW, LOW, 0, 0);
  }
}
```

What you should see: On a straight line the robot rolls smoothly forward. At a curve one sensor finds the tape and the robot pivots to recentre. Lift it onto a fully black patch and it stops.

Hand-Controlled Robot

Goal: Steer the chassis by hand with an analog joystick, with the stick's position setting both direction and speed.

A joystick is two potentiometers at right angles. At rest both read about 512, and pushing in either direction swings the reading towards 0 or 1023. Subtracting 512 turns each axis into a signed value: how far, and which way. A dead zone of ±80 is essential, real sticks never centre perfectly, and without it the robot would creep whenever you let go.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-37-hand-controlled-robot

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x PS2 Thumb Joystick Module (KY-023 style)
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Analog joystick mapping to motor speeds
- Centring a reading by subtracting the midpoint
- Why a dead zone is necessary
- Choosing between axes with abs() comparisons

Wiring

Part	Pin	Nano	Note
L298N	ENA	**D9**	Pull the ENA jumper cap off first, so the Nano sets the speed
L298N	IN1	**D4**	
L298N	IN2	**D5**	
L298N	IN3	**D6**	
L298N	IN4	**D7**	
L298N	ENB	**D10**	Pull the ENB jumper cap off first, so the Nano sets the speed
L298N	VCC (+5V)	**5V**	
Joystick	GND	**GND**	
Joystick	VRx	**A0**	
Joystick	VRy	**A1**	
Joystick	SW	**D2**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and Nano GND**	The common ground is required
L298N	+5V	**Nano 5V**	Powers the Nano when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	

Motors run from the 2 x 18650 pack (7.4V) through the L298N. The USB cable powers the Nano while you test; unplugged, the L298N's 5V output powers it.

Switch the battery holder off before you change any wiring, and lift the wheels off the table for the first test.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
int ENA = 9, IN1 = 4, IN2 = 5, IN3 = 6, IN4 = 7, ENB = 10;
int xPin = A0;
int yPin = A1;
int deadZone = 80;

void motors(int l1, int l2, int r1, int r2, int power)
{
  digitalWrite(IN1, l1); digitalWrite(IN2, l2);
  digitalWrite(IN3, r1); digitalWrite(IN4, r2);
  analogWrite(ENA, power);
  analogWrite(ENB, power);
}

void setup()
{
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  Serial.begin(9600);
}

void loop()

{
  int x = analogRead(xPin) - 512;
  int y = analogRead(yPin) - 512;

  Serial.print("X: ");
  Serial.print(x);
  Serial.print(" Y: ");
  Serial.println(y);

  if (abs(y) > deadZone && abs(y) >= abs(x))
  {
    int power = map(abs(y), deadZone, 512, 90, 255);
    power = constrain(power, 90, 255);

    if (y > 0)
    {
      motors(HIGH, LOW, HIGH, LOW, power);                              // Forward
    }
    else
    {
      motors(LOW, HIGH, LOW, HIGH, power);                              // Backward
    }
  }
  else if (abs(x) > deadZone)
  {
    int power = map(abs(x), deadZone, 512, 90, 220);
    power = constrain(power, 90, 220);

    if (x > 0)
    {
      motors(HIGH, LOW, LOW, HIGH, power);                              // Spin right
    }
    else
    {
      motors(LOW, HIGH, HIGH, LOW, power);                              // Spin left
    }
  }
  else
  {
    motors(LOW, LOW, LOW, LOW, 0);                                        // Centred - stop
  }
  delay(30);
}
```

What you should see: Serial Monitor shows X and Y near 0 when the stick is centred. Push forward gently and the robot creeps; push fully and it runs. Push sideways and it spins on the spot. Release the stick and it stops immediately.

Voice Buzzer Robot

Goal: Start and stop the robot by clapping, using a sound sensor as a one-bit voice command.

The sound module's digital output pulses LOW whenever the noise crosses the level set by its onboard potentiometer. One clap can produce several of those pulses as the sound echoes, so the sketch ignores anything arriving within 400 ms of the last accepted clap. That guard is the difference between a robot that toggles once and one that flickers on and off. A single boolean holds the whole state of the machine: moving, or not.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-38-voice-buzzer-robot

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Generic LM393 Sound Detection Sensor Module – Digital Output
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Sound triggers and basic movement
- Toggling state with the ! operator
- Ignoring repeat triggers with millis()
- Why echoes need a guard window

Wiring

Part	Pin	Nano	Note
L298N	ENA	**D9**	Pull the ENA jumper cap off first, so the Nano sets the speed
L298N	IN1	**D4**	
L298N	IN2	**D5**	
L298N	IN3	**D6**	
L298N	IN4	**D7**	
L298N	ENB	**D10**	Pull the ENB jumper cap off first, so the Nano sets the speed
Sound sensor	Sound sensor VCC	**5V**	
Sound sensor	Sound sensor GND	**GND**	
Sound sensor	Sound sensor DO	**D2**	
LED	LED anode	**D8**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and Nano GND**	The common ground is required

L298N	+5V	**Nano 5V**	Powers the Nano when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	

Motors run from the 2 x 18650 pack (7.4V) through the L298N. The USB cable powers the Nano while you test; unplugged, the L298N's 5V output powers it.

Switch the battery holder off before you change any wiring, and lift the wheels off the table for the first test.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
int ENA = 9, IN1 = 4, IN2 = 5, IN3 = 6, IN4 = 7, ENB = 10;
int soundPin = 2;
int led = 8;

bool moving = false;
unsigned long lastClap = 0;
int speedValue = 150;

void motors(int l1, int l2, int r1, int r2, int power)
{
  digitalWrite(IN1, l1); digitalWrite(IN2, l2);
  digitalWrite(IN3, r1); digitalWrite(IN4, r2);
  analogWrite(ENA, power);
  analogWrite(ENB, power);
}

void setup()
{
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(soundPin, INPUT);
  pinMode(led, OUTPUT);

  Serial.begin(9600);
  Serial.println("Clap to start or stop");
}

void loop()
{
  if (digitalRead(soundPin) == LOW)
  {
    if (millis() - lastClap > 400)
    {
      moving = !moving;
      lastClap = millis();

      Serial.print("Clap detected. Moving: ");
      Serial.println(moving);
    }
  }

  if (moving)
  {
    motors(HIGH, LOW, HIGH, LOW, speedValue);
    digitalWrite(led, HIGH);
  }
  else
  {
    motors(LOW, LOW, LOW, LOW, 0);
    digitalWrite(led, LOW);
  }
}
```

What you should see: Clap once: the LED lights and the robot drives forward. Clap again: it stops and the LED goes out. Serial Monitor logs each accepted clap.

Maze Solver Beginner Bot

Goal: Combine a forward ultrasonic sensor with two side IR sensors and apply the left-hand rule to work through a simple maze.

The left-hand rule says: if you can turn left, turn left; otherwise go straight; otherwise turn right; otherwise turn around. Follow it consistently and you will escape any maze without loops, with no map at all. Each rule is one branch of an if- else chain, and the order of the branches is the whole algorithm, swap two of them and the robot behaves completely differently.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-39-maze-solver-bot

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 2 x Generic Infrared Obstacle Avoidance Sensor Module (Active Low) – LM393 Comparator, 2-30cm Range
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Complex if-else sensor algorithms
- The left-hand maze rule
- Combining two sensor types in one decision
- How branch order changes behaviour

Wiring

Part	Pin	Nano	Note
L298N	ENA	**D9**	Pull the ENA jumper cap off first, so the Nano sets the speed
L298N	IN1	**D4**	
L298N	IN2	**D5**	
L298N	IN3	**D6**	
L298N	IN4	**D7**	
L298N	ENB	**D10**	Pull the ENB jumper cap off first, so the Nano sets the speed
HC-SR04	HC-SR04 TRIG (front)	**D11**	
HC-SR04	HC-SR04 ECHO (front)	**D12**	
IR sensor	Left IR sensor OUT	**D2**	
IR sensor	Right IR sensor OUT	**D3**	
L298N	All VCC	**5V**	
L298N	All GND	**GND**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire

L298N	GND	**Battery (−) and Nano GND**	The common ground is required
L298N	+5V	**Nano 5V**	Powers the Nano when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	

Motors run from the 2 x 18650 pack (7.4V) through the L298N. The USB cable powers the Nano while you test; unplugged, the L298N's 5V output powers it.

Switch the battery holder off before you change any wiring, and lift the wheels off the table for the first test.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
int ENA = 9, IN1 = 4, IN2 = 5, IN3 = 6, IN4 = 7, ENB = 10;
int trigPin = 11, echoPin = 12;
int leftIR = 2, rightIR = 3;

int speedValue = 140;
int wallDistance = 20;

void motors(int l1, int l2, int r1, int r2)
{
  digitalWrite(IN1, l1); digitalWrite(IN2, l2);
  digitalWrite(IN3, r1); digitalWrite(IN4, r2);
  analogWrite(ENA, speedValue);
  analogWrite(ENB, speedValue);
}

void forward() { motors(HIGH, LOW, HIGH, LOW ); }
void turnLeft() { motors(LOW, HIGH, HIGH, LOW ); }
void turnRight(){ motors(HIGH, LOW, LOW, HIGH); }
void stopBot() { motors(LOW, LOW, LOW, LOW); analogWrite(ENA, 0); analogWrite(ENB,
  0); }

long frontDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH, 25000);
  if (duration == 0)
  {
    return 200;
  }
  return duration * 0.034 / 2;
}

void setup()
{
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(leftIR, INPUT);
  pinMode(rightIR, INPUT);

  Serial.begin(9600);
  delay(2000);
}

void loop()
{
  long front = frontDistance();
  bool leftWall = (digitalRead(leftIR) == LOW);
  bool rightWall = (digitalRead(rightIR) == LOW);

  Serial.print("Front: ");
  Serial.print(front);
  Serial.print(" Left wall: ");
  Serial.print(leftWall);
  Serial.print(" Right wall: ");
  Serial.println(rightWall);

  // Left-hand rule
  if (!leftWall)
  {
    stopBot();
    delay(150);
    turnLeft();
    delay(450);
    forward();
    delay(300);
  }
  else if (front > wallDistance)
```

```
  {
    forward();
    delay(60);
  }
  else if (!rightWall)
  {
    stopBot();
    delay(150);
    turnRight();
    delay(450);
  }
  else
  {
    stopBot();
    delay(150);
    turnRight();
    delay(900);        // Dead end - turn around
  }
}
```

What you should see: Serial Monitor prints the front distance and both wall flags. The robot hugs the left wall through the maze. At a dead end it turns a full half circle and comes back.

Smart Parking Robot Demo

Goal: Model a two-bay car park: ultrasonic sensors watch each bay and an LCD plus coloured LEDs show which spaces are free.

Both bays run the same measurement, so the trigger and echo pins live in arrays and one loop handles them all, adding a third bay means extending the arrays, not copying the code. A car parked in a bay sits close to the sensor, so a short distance means occupied. Measuring one sensor at a time with a short pause between them stops the second sensor from hearing the first sensor's echo.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-40-smart-parking-demo

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x LCD1602 16x2 Character LCD Display
- 2 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Precision distance measurement
- Handling several identical sensors with arrays
- Avoiding cross-talk between ultrasonic sensors
- Summarising several states into one count

Wiring

Part	Pin	Nano	Note
HC-SR04	Bay 1 TRIG	**D2**	
HC-SR04	Bay 1 ECHO	**D3**	
HC-SR04	Bay 2 TRIG	**D4**	
HC-SR04	Bay 2 ECHO	**D5**	
HC-SR04	Both VCC / GND	**5V / GND**	
LED	Bay 1 green LED	**D6**	Through a 220 ohm resistor
LED	Bay 1 red LED	**D7**	Through a 220 ohm resistor
LED	Bay 2 green LED	**D8**	Through a 220 ohm resistor
LED	Bay 2 red LED	**D9**	Through a 220 ohm resistor
LED	All cathodes	**GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int trigPins[2] = {2, 4};
int echoPins[2] = {3, 5};
int greenLeds[2] = {6, 8};
int redLeds[2] = {7, 9};

int occupiedBelow = 12;                   // cm

LiquidCrystal_I2C lcd(0x27, 16, 2);

long readBay(int bay)
{
  digitalWrite(trigPins[bay], LOW);
  delayMicroseconds(2);
  digitalWrite(trigPins[bay], HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPins[bay], LOW);

  long duration = pulseIn(echoPins[bay], HIGH, 25000);
  if (duration == 0)
  {
    return 200;
  }
  return duration * 0.034 / 2;
}

void setup()
{
  for (int i = 0; i < 2; i++)
  {
    pinMode(trigPins[i], OUTPUT);
    pinMode(echoPins[i], INPUT);
    pinMode(greenLeds[i], OUTPUT);
    pinMode(redLeds[i], OUTPUT);
  }

  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Smart Parking");
  delay(2000);
  lcd.clear();
}

void loop()
{
  int freeCount = 0;

  for (int i = 0; i < 2; i++)
  {
    long distance = readBay(i);
    bool occupied = (distance < occupiedBelow);

    digitalWrite(redLeds[i], occupied);
    digitalWrite(greenLeds[i], !occupied);
    if (!occupied)
    {
      freeCount = freeCount + 1;
    }

    lcd.setCursor(0, i);
    lcd.print("Bay ");
    lcd.print(i + 1);
    lcd.print(": ");
    if (occupied)
    {
      lcd.print("FULL ");
    }
    else
    {
      lcd.print("FREE ");
    }
```

```
    Serial.print("Bay ");
    Serial.print(i + 1);
    Serial.print(" distance ");
    Serial.println(distance);

    delay(60);              // Let the echo die away
  }

  lcd.setCursor(11, 0);
  lcd.print("F:");
  lcd.print(freeCount);

  delay(400);
}
```

What you should see: Bay 1: FREE / Bay 2: FREE with both green LEDs on. Place a book within 12 cm of one sensor: that bay shows FULL and its red LED lights. The free-space count in the corner updates immediately.

Mini Delivery Robot

Goal: Send the robot to numbered destinations: each Bluetooth command runs a fixed route, then it reports that it has arrived.

Instead of the driver holding a key down, one command starts a whole pre-programmed route. Each destination is a short list of moves, forward for so long, turn, forward again, and the robot works through it, then buzzes to say the delivery is complete. Because the robot is following a stored sequence rather than reacting moment to moment, this is a state machine: it is in one named state at a time, and the command chooses which.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-41-mini-delivery-robot

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x PCB Mounted Active Buzzer Module
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x 1k and 2.2k resistors
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Bluetooth directional state machines
- Encoding a route as a sequence of timed moves
- Sending status messages back to the operator
- Why timed routes drift and need re-tuning

Wiring

Part	Pin	Nano	Note
L298N	ENA	**D9**	Pull the ENA jumper cap off first, so the Nano sets the speed
L298N	IN1	**D4**	
L298N	IN2	**D5**	
L298N	IN3	**D6**	
L298N	IN4	**D7**	
L298N	ENB	**D10**	Pull the ENB jumper cap off first, so the Nano sets the speed
HC-05	HC-05 TXD	**D2**	
HC-05	HC-05 RXD	**D3**	Through a divider: 1k from D3 to RXD and 2.2k from RXD to GND, because RXD is a 3.3V input
HC-05	HC-05 VCC / GND	**5V / GND**	
Buzzer	Buzzer (+)	**D8**	
Buzzer	Buzzer (−)	**GND**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and Nano GND**	The common ground is required

L298N	+5V	**Nano 5V**	Powers the Nano when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	

Motors run from the 2 x 18650 pack (7.4V) through the L298N. The USB cable powers the Nano while you test; unplugged, the L298N's 5V output powers it.

Switch the battery holder off before you change any wiring, and lift the wheels off the table for the first test.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <SoftwareSerial.h>

SoftwareSerial bt(2, 3);

int ENA = 9, IN1 = 4, IN2 = 5, IN3 = 6, IN4 = 7, ENB = 10;
int buzzer = 8;
int speedValue = 150;

void motors(int l1, int l2, int r1, int r2)
{
  digitalWrite(IN1, l1); digitalWrite(IN2, l2);
  digitalWrite(IN3, r1); digitalWrite(IN4, r2);
  analogWrite(ENA, speedValue);
  analogWrite(ENB, speedValue);
}

void forward(int ms) { motors(HIGH, LOW, HIGH, LOW ); delay(ms); }
void turnLeft(int ms) { motors(LOW, HIGH, HIGH, LOW ); delay(ms); }
void turnRight(int ms){ motors(HIGH, LOW, LOW, HIGH); delay(ms); }
void stopBot()         { motors(LOW, LOW, LOW, LOW); analogWrite(ENA, 0);
  analogWrite(ENB, 0); }

void arrived()
{
  stopBot();
  for (int i = 0; i < 3; i++)
  {
    digitalWrite(buzzer, HIGH);
    delay(150);
    digitalWrite(buzzer, LOW);
    delay(150);
  }
  bt.println("Delivered");
}

void setup()
{
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(buzzer, OUTPUT);

  bt.begin(9600);
  Serial.begin(9600);
  stopBot();
}

void loop()
{
  if (!bt.available())
  {
    return;
  }
  char destination = bt.read();
  Serial.println(destination);

  if (destination == '1')
  {
    bt.println("Going to desk 1");
    forward(1500);
    arrived();
  }
  else if (destination == '2')
  {
    bt.println("Going to desk 2");
    forward(1200);
    turnRight(450);
    forward(1000);
    arrived();
  }
  else if (destination == '3')
  {
    bt.println("Going to desk 3");
    forward(1200);
    turnLeft(450);
```

```
    forward(1000);
    arrived();
  }
  else if (destination == 'H')
  {
    bt.println("Returning home");
    turnRight(900);
    forward(1600);
    arrived();
  }
  else if (destination == 'S')
  {
    stopBot();
  }
}
```

What you should see: Send 1 and the robot drives straight to desk 1, then buzzes three times. Send 2 or 3 and it takes a turn on the way. The phone receives Going to desk 2 and then Delivered. Send S at any time to stop.

Auto Reverse Collision Robot

Goal: Add a safety layer that overrides everything: if an object comes too close, stop and reverse immediately.

This is a proximity safety interrupt. The distance check happens at the very top of loop(), before any driving logic, and it returns early when the path is blocked, so nothing further down can override the stop. Two distances are used: below 15 cm the robot reverses, and it will not resume until it has at least 25 cm of clearance. That gap prevents it from lurching forward and back at exactly 15 cm.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-42-auto-reverse-robot

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x PCB Mounted Active Buzzer Module
- 1 x HC-SR04 Ultrasonic Range Finder Sensor Module for Arduino
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Proximity safety interrupts
- Using return to make a check override later code
- Two thresholds to prevent oscillation
- Ordering checks by priority

Wiring

Part	Pin	Nano	Note
L298N	ENA	**D9**	Pull the ENA jumper cap off first, so the Nano sets the speed
L298N	IN1	**D4**	
L298N	IN2	**D5**	
L298N	IN3	**D6**	
L298N	IN4	**D7**	
L298N	ENB	**D10**	Pull the ENB jumper cap off first, so the Nano sets the speed
HC-SR04	HC-SR04 TRIG	**D11**	
HC-SR04	HC-SR04 ECHO	**D12**	
HC-SR04	HC-SR04 VCC / GND	**5V / GND**	
LED	Red LED anode	**D8**	Through a 220 ohm resistor
Buzzer	Buzzer (+)	**D3**	
Buzzer	LED cathode / buzzer (−)	**GND**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire

L298N	GND	**Battery (−) and Nano GND**	The common ground is required
L298N	+5V	**Nano 5V**	Powers the Nano when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	

Motors run from the 2 x 18650 pack (7.4V) through the L298N. The USB cable powers the Nano while you test; unplugged, the L298N's 5V output powers it.

Switch the battery holder off before you change any wiring, and lift the wheels off the table for the first test.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
int ENA = 9, IN1 = 4, IN2 = 5, IN3 = 6, IN4 = 7, ENB = 10;
int trigPin = 11, echoPin = 12;
int led = 8, buzzer = 3;

int dangerDistance = 15;

int clearDistance = 25;
int speedValue = 150;
bool blocked = false;

void motors(int l1, int l2, int r1, int r2)
{
  digitalWrite(IN1, l1); digitalWrite(IN2, l2);
  digitalWrite(IN3, r1); digitalWrite(IN4, r2);
  analogWrite(ENA, speedValue);
  analogWrite(ENB, speedValue);
}

void forward() { motors(HIGH, LOW, HIGH, LOW ); }
void backward() { motors(LOW, HIGH, LOW, HIGH); }
void stopBot() { motors(LOW, LOW, LOW, LOW); analogWrite(ENA, 0);
  analogWrite(ENB, 0); }

long readDistance()
{
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);

  long duration = pulseIn(echoPin, HIGH, 25000);
  if (duration == 0)
  {
    return 200;
  }
  return duration * 0.034 / 2;
}

void setup()
{
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);
  pinMode(led, OUTPUT);
  pinMode(buzzer, OUTPUT);

  Serial.begin(9600);
  delay(2000);
}

void loop()
{
  long distance = readDistance();
  Serial.println(distance);

  // Safety check runs first and overrides everything
  if (distance < dangerDistance)
  {
    blocked = true;

    stopBot();
    digitalWrite(led, HIGH);
    digitalWrite(buzzer, HIGH);
    delay(200);
    digitalWrite(buzzer, LOW);

    backward();
    delay(600);
    stopBot();
    delay(300);
    return;
  }
```

```
  if (distance > clearDistance)
  {
    blocked = false;
  }

  if (blocked)
  {
    stopBot();
    return;
  }

  digitalWrite(led, LOW);
  forward();
}
```

What you should see: The robot drives forward across clear floor. Put your hand within 15 cm: it stops, the LED and buzzer fire, and it reverses. It waits until there is 25 cm of clearance before moving forward again.

Gesture-like Joystick Car

Goal: Drive with proportional steering: the joystick's X and Y are mixed so each wheel gets its own speed and the car turns smoothly.

Experiment 37 picked one axis and ignored the other. Here both are used at once, with the classic differential mix: left = forward + turn, right = forward − turn. Push the stick diagonally and one wheel simply runs faster than the other, so the car sweeps through a curve instead of pivoting. The sign of each result decides that wheel's direction, and its magnitude becomes the PWM value, this is Cartesian mapping, turning an (x, y) point into two motor commands.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-43-joystick-car

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x PS2 Thumb Joystick Module (KY-023 style)
- 1 x 2WD Smart Robot Car Chassis Kit
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Analog Cartesian mapping
- Differential drive mixing
- Turning a signed value into direction plus PWM
- Why constrain() matters when two values are added

Wiring

Part	Pin	Nano	Note
L298N	ENA	**D9**	Pull the ENA jumper cap off first, so the Nano sets the speed
L298N	IN1	**D4**	
L298N	IN2	**D5**	
L298N	IN3	**D6**	
L298N	IN4	**D7**	
L298N	ENB	**D10**	Pull the ENB jumper cap off first, so the Nano sets the speed
L298N	VCC (+5V)	**5V**	
Joystick	GND	**GND**	
Joystick	VRx	**A0**	
Joystick	VRy	**A1**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	Switch the holder off while you wire
L298N	GND	**Battery (−) and Nano GND**	The common ground is required
L298N	+5V	**Nano 5V**	Powers the Nano when the USB cable is out (5V jumper stays on)
Motors	Left motor wires	**OUT1 and OUT2**	Swap the two wires if this wheel runs backwards
Motors	Right motor wires	**OUT3 and OUT4**	

Motors run from the 2 x 18650 pack (7.4V) through the L298N. The USB cable powers the Nano while you test; unplugged, the L298N's 5V output powers it.

Switch the battery holder off before you change any wiring, and lift the wheels off the table for the first test.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
int ENA = 9, IN1 = 4, IN2 = 5, IN3 = 6, IN4 = 7, ENB = 10;
int xPin = A0;
int yPin = A1;
int deadZone = 60;

void setLeft(int value)
{
  if (value >= 0)
  {
    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);
  }
  else
  {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, HIGH);
  }
  analogWrite(ENA, constrain(abs(value), 0, 255));
}

void setRight(int value)

{
  if (value >= 0)
  {
    digitalWrite(IN3, HIGH);
    digitalWrite(IN4, LOW);
  }
  else
  {
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, HIGH);
  }
  analogWrite(ENB, constrain(abs(value), 0, 255));
}

void setup()
{
  pinMode(ENA, OUTPUT); pinMode(ENB, OUTPUT);
  pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);
  Serial.begin(9600);
}

void loop()
{
  int x = analogRead(xPin) - 512;
  int y = analogRead(yPin) - 512;

  if (abs(x) < deadZone) x = 0;
  if (abs(y) < deadZone) y = 0;

  int drive = map(y, -512, 512, -255, 255);
  int turn = map(x, -512, 512, -180, 180);

  int leftPower = drive + turn;
  int rightPower = drive - turn;

  leftPower = constrain(leftPower, -255, 255);
  rightPower = constrain(rightPower, -255, 255);
  Serial.print("L: ");
  Serial.print(leftPower);
  Serial.print(" R: ");
  Serial.println(rightPower);

  setLeft(leftPower);
  setRight(rightPower);

  delay(30);
}
```

What you should see: Stick centred: both powers read 0 and the car sits still. Stick fully forward: L: 255 R: 255 and it runs straight. Stick forward-right: L: 255 R: 75 and it sweeps through a curve. Stick

fully right: L: 180 R: -180 and it spins on the spot.

Bluetooth LED Controller

Goal: Switch three LEDs on and off individually from a phone app, and read back the state of each one.

Uppercase turns an LED on and lowercase turns it off, a small design decision that makes the protocol readable and halves the number of commands to remember. The LED pins sit in an array so the same three lines of code handle all of them, and the state array remembers what is currently on so the board can answer a status request. Pairing is a one- time step: once your phone knows the HC-05, any serial terminal app can send these letters.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-44-bluetooth-led-controller

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x 1k and 2.2k resistors
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Smartphone serial pairing
- Designing a readable command protocol
- Keeping device state in an array
- Answering a status query over the same link

Wiring

Part	Pin	Nano	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**D2 (Nano RX)**	
HC-05	RXD	**D3 (Nano TX)**	Through a divider: 1k from D3 to RXD and 2.2k from RXD to GND, because RXD is a 3.3V input
LED	Red anode	**D5**	Through a 220 ohm resistor
LED	Yellow anode	**D6**	Through a 220 ohm resistor
LED	Green anode	**D7**	Through a 220 ohm resistor
LED	All cathodes	**GND**	

Code

```
#include <SoftwareSerial.h>

SoftwareSerial bt(2, 3);

int ledPins[3] = {5, 6, 7};
String ledNames[3] = {"RED", "YELLOW", "GREEN"};
bool ledState[3] = {false, false, false};

void applyState()

{
  for (int i = 0; i < 3; i++)
  {
    digitalWrite(ledPins[i], ledState[i]);
  }
}

void reportState()
{
  for (int i = 0; i < 3; i++)
  {
    bt.print(ledNames[i]);
    bt.print(": ");
    if (ledState[i])
    {
      bt.println("ON");
    }
    else
    {
      bt.println("OFF");
    }
  }
}

void setup()
{
  for (int i = 0; i < 3; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }

  bt.begin(9600);
  Serial.begin(9600);
  applyState();

  bt.println("Send R G Y to turn on, r g y to turn off, ? for status");
}

void loop()
{
  if (!bt.available())
  {
    return;
  }

  char command = bt.read();
  Serial.println(command);

  if (command == 'R') ledState[0] = true;
  else if (command == 'r') ledState[0] = false;
  else if (command == 'Y') ledState[1] = true;
  else if (command == 'y') ledState[1] = false;
  else if (command == 'G') ledState[2] = true;
  else if (command == 'g') ledState[2] = false;
  else if (command == 'A') { ledState[0] = true; ledState[1] = true; ledState[2] =
    true; }
  else if (command == 'a') { ledState[0] = false; ledState[1] = false; ledState[2]
    = false; }
  else if (command == '?') { reportState(); return; }
  applyState();
}
```

What you should see: The phone receives the command list when the board starts. Send R and the red LED lights; send r and it goes out. Send A to light all three, a to clear them. Send ? and the phone

receives RED: ON, YELLOW: OFF, GREEN: ON.

Bluetooth Fan Control

Goal: Run a DC motor fan from your phone with four speed settings, driven through the motor driver.

A DC motor will not start reliably at very low PWM values, there is not enough torque to overcome friction. That is why the speed levels here start at 120 rather than at 1, and why level 0 means fully off rather than very slow. The levels live in an array, so the command character is converted to a number and used directly as the index: '2' becomes 2, and speeds[2] is the value written to ENA.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-45-bluetooth-fan

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x 1k and 2.2k resistors
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Hobby DC Motor (130 / 180 Size)
- 1 x DC Motor Fan Blade (Toy Propeller)
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Remote device switching
- Why low PWM values will not start a motor
- Converting a character digit to a number with − '0'
- Using a command as an array index

Wiring

Part	Pin	Nano	Note
L298N	ENA	**D9**	Pull the ENA jumper cap off first, so the Nano sets the speed
L298N	IN1	**D4**	
L298N	IN2	**D5**	
Fan motor	Motor wires	**L298N OUT1 and OUT2**	
L298N	OUT1 / OUT2	**Motor terminals**	
L298N	+12V	**Battery (+)**	
L298N	GND	**Battery (−) and Nano GND**	
HC-05	HC-05 TXD	**D2**	
HC-05	HC-05 RXD	**D3**	Through a divider: 1k from D3 to RXD and 2.2k from RXD to GND, because RXD is a 3.3V input
HC-05	HC-05 VCC / GND	**5V / GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

Motors run from the 2 x 18650 pack (7.4V) through the L298N. The USB cable powers the Nano while

you test; unplugged, the L298N's 5V output powers it.

Switch the battery holder off before you change any wiring, and lift the wheels off the table for the first test.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <SoftwareSerial.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

SoftwareSerial bt(2, 3);
LiquidCrystal_I2C lcd(0x27, 16, 2);

int ENA = 9;
int IN1 = 4;
int IN2 = 5;

int speeds[4] = {0, 120, 190, 255};
String labels[4] = {"OFF ", "LOW ", "MED ", "HIGH"};
int level = 0;

void applyLevel()
{
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, speeds[level]);

  lcd.setCursor(0, 0);
  lcd.print("Fan: ");
  lcd.print(labels[level]);
  lcd.print("    ");

  lcd.setCursor(0, 1);
  lcd.print("PWM: ");
  lcd.print(speeds[level]);
  lcd.print("    ");

  bt.print("Fan set to ");
  bt.println(labels[level]);
}

void setup()
{
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  bt.begin(9600);
  Serial.begin(9600);

  lcd.init();
  lcd.backlight();
  applyLevel();
  bt.println("Send 0 1 2 3 to set fan speed");
}

void loop()
{
  if (!bt.available())
  {
    return;
  }

  char command = bt.read();
  Serial.println(command);
  if (command >= '0' && command <= '3')
  {
    level = command - '0';
    applyLevel();
  }
}
```

What you should see: LCD shows Fan: OFF / PWM: 0 at startup. Send 1 and the fan starts slowly: Fan: LOW / PWM: 120. Send 3 for full speed, 0 to stop. The phone receives Fan set to MED after each change.

Bluetooth Home Automation

Goal: Control two relay-switched appliances (a light and a fan) from a phone, with a master command to switch everything off at once.

Most relay modules are active LOW: writing LOW to an input energises that relay. That inversion catches everyone out at least once, so the sketch keeps a friendly true/false state array and does the flipping in one place, inside applyStates(). The LCD then draws a compact status line where each appliance is one character, which is enough to see the whole house at a glance.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-46-bluetooth-home-automation

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x LCD1602 16x2 Character LCD Display
- 2 x 5V Relay Module with Optocoupler
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x 1k and 2.2k resistors
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Relay array control over Bluetooth
- Handling active-LOW hardware cleanly
- Mapping command ranges to array indexes
- Designing a compact status display

Wiring

Part	Pin	Nano	Note
Relay	VCC	**5V**	
Relay	GND	**GND**	
HC-05	HC-05 TXD	**D2**	
HC-05	HC-05 RXD	**D3**	Through a divider: 1k from D3 to RXD and 2.2k from RXD to GND, because RXD is a 3.3V input
HC-05	HC-05 VCC / GND	**5V / GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	
Relay 1	IN	**D4**	First relay module (Light)
Relay 2	IN	**D5**	Second relay module (Fan)
Relays 1 and 2	COM	**Nano 5V**	Demo loads only: never mains
Load LEDs	Anodes	**Relay 1 NO and relay 2 NO**	Each through a 220 ohm resistor
Load LEDs	Cathodes	**GND**	

> Switch only low-voltage demo loads such as the LED here. Never wire mains to these relay modules in a classroom project.

Code

```
#include <SoftwareSerial.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

SoftwareSerial bt(2, 3);
LiquidCrystal_I2C lcd(0x27, 16, 2);

int relayPins[2] = {4, 5};      // two single relay modules
String names[2] = {"Light", "Fan"};
bool state[2] = {false, false};

void applyStates()
{
  for (int i = 0; i < 2; i++)
  {
    // Relay modules are active LOW
    if (state[i])
    {
      digitalWrite(relayPins[i], LOW);
    }
    else
    {
      digitalWrite(relayPins[i], HIGH);
    }
  }

  lcd.setCursor(0, 0);
  lcd.print("Light Fan");
  lcd.setCursor(0, 1);
  for (int i = 0; i < 2; i++)
  {
    if (state[i])
    {
      lcd.print("ON   ");
    }
    else
    {
      lcd.print("OFF  ");
    }
  }
}

void setup()
{
  for (int i = 0; i < 2; i++)
  {
    pinMode(relayPins[i], OUTPUT);
  }

  bt.begin(9600);
  Serial.begin(9600);

  lcd.init();
  lcd.backlight();
  applyStates();

  bt.println("1-2 ON, 5-6 OFF, 9 all OFF");
}

void loop()
{
  if (!bt.available())
  {
    return;
  }

  char command = bt.read();
  Serial.println(command);

  if (command >= '1' && command <= '2')
  {
    int index = command - '1';
    state[index] = true;
    bt.print(names[index]);
    bt.println(" ON");
```

```
  }
  else if (command >= '5' && command <= '6')
  {
    int index = command - '5';
    state[index] = false;
    bt.print(names[index]);
    bt.println(" OFF");
  }
  else if (command == '9')
  {
    for (int i = 0; i < 2; i++)
    {
      state[i] = false;
    }
    bt.println("All appliances OFF");
  }

  applyStates();
}
```

What you should see: LCD shows L F P M above a row of four zeros. Send 1: the first relay clicks, the phone reads Light ON and the display shows 1 0 0 0. Send 5 to switch it off again. Send 9 and every relay releases at once.

Wireless RGB Controller

Goal: Send a colour to the board as text, like 255,80,0, and have the RGB LED mix it instantly.

Earlier sketches read one character at a time. A colour needs three numbers, so this one collects characters into a String until a newline arrives, then splits that String at the commas using indexOf() and substring(). toInt() converts each piece to a number. This collect-then-parse pattern is how almost all text protocols work, and it is worth understanding properly, it reappears in the notice board and the sensor dashboard.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-47-wireless-rgb

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 5mm RGB LED (Pack of 10)
- 1 x 220 ohm resistors
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x 1k and 2.2k resistors
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- App-based colour value parsing
- Building a String until a newline arrives
- Splitting text with indexOf() and substring()
- Accepting both named and numeric commands

Wiring

Part	Pin	Nano	Note
RGB LED	Red	**D9**	Through a 220 ohm resistor
RGB LED	Green	**D10**	Through a 220 ohm resistor
RGB LED	Blue	**D11**	Through a 220 ohm resistor
RGB LED	Common cathode	**GND**	
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**D2**	
HC-05	RXD	**D3**	Through a divider: 1k from D3 to RXD and 2.2k from RXD to GND, because RXD is a 3.3V input

Code

```
#include <SoftwareSerial.h>

SoftwareSerial bt(2, 3);

int redPin = 9;
int greenPin = 10;
int bluePin = 11;

String input = "";

void showColour(int r, int g, int b)
{
  r = constrain(r, 0, 255);
  g = constrain(g, 0, 255);
  b = constrain(b, 0, 255);
  analogWrite(redPin, r);
  analogWrite(greenPin, g);
  analogWrite(bluePin, b);

  bt.print("Colour set to ");
  bt.print(r); bt.print(",");
  bt.print(g); bt.print(",");
  bt.println(b);
}

void handleLine(String line)
{
  line.trim();

  if (line == "off")
  {
    showColour(0, 0, 0);
    return;
  }
  if (line == "red")   { showColour(255, 0, 0);   return; }
  if (line == "green") { showColour(0, 255, 0);   return; }
  if (line == "blue") { showColour(0, 0, 255);    return; }
  if (line == "white") { showColour(255, 255, 255); return; }

  int firstComma = line.indexOf(',');
  int secondComma = line.indexOf(',', firstComma + 1);

  if (firstComma < 0 || secondComma < 0)
  {
    bt.println("Send R,G,B for example 255,80,0");
    return;
  }

  int r = line.substring(0, firstComma).toInt();
  int g = line.substring(firstComma + 1, secondComma).toInt();
  int b = line.substring(secondComma + 1).toInt();

  showColour(r, g, b);
}

void setup()
{
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);

  bt.begin(9600);
  Serial.begin(9600);
  showColour(0, 0, 0);
  bt.println("Send R,G,B or red green blue white off");
}

void loop()
{
  while (bt.available())
  {
    char c = bt.read();

    if (c == '\n')
    {
```

```
      handleLine(input);
      input = "";
    }
    else if (c != '\r')
    {
      input = input + c;
    }
  }
}
```

What you should see: Send 255,0,0 and the LED glows red. Send 255,80,0 for a warm orange. Send green or white to use a named colour. Send off to turn it out; the phone confirms each change.

Bluetooth Notice Board

Goal: Type a message on your phone and watch it appear on the LCD, with long messages wrapped across both lines.

The screen holds sixteen characters per line, so a longer message has to be cut into pieces. substring(0, 16) takes the first line and substring(16, 32) the second. length() tells the sketch whether there is anything left to show at all, asking for a substring beyond the end of a String is a common cause of blank screens. The buzzer chirp is a small touch that makes a new notice impossible to miss.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-48-bluetooth-notice-board

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x 1k and 2.2k resistors
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Serial string parsing to an LCD
- Splitting text with substring() and length()
- Guarding against out-of-range substrings
- Confirming an action back to the sender

Wiring

Part	Pin	Nano	Note
HC-05	VCC	**5V**	
HC-05	GND	**GND**	
HC-05	TXD	**D2**	
HC-05	RXD	**D3**	Through a divider: 1k from D3 to RXD and 2.2k from RXD to GND, because RXD is a 3.3V input
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	
LCD	LCD VCC / GND	**5V / GND**	
Buzzer	Buzzer (+)	**D8**	
Buzzer	Buzzer (−)	**GND**	

Code

```
#include <SoftwareSerial.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

SoftwareSerial bt(2, 3);
LiquidCrystal_I2C lcd(0x27, 16, 2);

int buzzer = 8;
String input = "";

void showMessage(String text)
{
  lcd.clear();
  lcd.setCursor(0, 0);
  if (text.length() > 16)
  {
    lcd.print(text.substring(0, 16));
  }
  else
  {
    lcd.print(text);
  }
  if (text.length() > 16)
  {
    lcd.setCursor(0, 1);
    if (text.length() > 32)
    {
      lcd.print(text.substring(16, 32));
    }
    else
    {
      lcd.print(text.substring(16));
    }
  }

  digitalWrite(buzzer, HIGH);
  delay(80);
  digitalWrite(buzzer, LOW);

  bt.println("Notice displayed");
}

void setup()
{
  pinMode(buzzer, OUTPUT);

  bt.begin(9600);
  Serial.begin(9600);

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Notice Board");
  lcd.setCursor(0, 1);
  lcd.print("Waiting...");

  bt.println("Send any message to display it");
}

void loop()

{
  while (bt.available())
  {
    char c = bt.read();

    if (c == '\n')
    {
      input.trim();
      if (input.length() > 0)
      {
        Serial.println(input);
        showMessage(input);
      }
      input = "";
```

```
    }
    else if (c != '\r')
    {
      input = input + c;
    }
  }
}
```

What you should see: LCD shows Notice Board / Waiting... at startup. Send Class at 10 AM and it appears immediately with a short beep. Send a longer message and it wraps onto the second line. The phone receives Notice displayed as confirmation.

Wireless Sensor Dashboard

Goal: Stream live temperature, humidity and light readings to your phone, either on demand or automatically every few seconds.

Sending variables over a serial link means turning them into text and giving each one a label, because the receiver has no idea what the numbers mean otherwise. The automatic mode uses millis() rather than delay() so the board keeps listening for commands while it waits, a delay(3000) here would make the dashboard ignore you for three seconds at a time.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-49-wireless-sensor-dashboard

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x 10k resistor
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x 5 mm LDR Light Dependent Resistor
- 1 x HC-05 6-Pin Bluetooth Serial Module with Button
- 1 x 1k and 2.2k resistors
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Sending variables over Serial with labels
- Non-blocking timing with millis()
- Staying responsive while streaming data
- Mirroring output to two displays at once

Wiring

Part	Pin	Nano	Note
DHT11	DHT11 DATA	**D4**	
DHT11	DHT11 VCC / GND	**5V / GND**	
LDR divider	LDR leg 1	**5V**	
LDR divider	LDR leg 2	**A0 and 10kΩ to GND**	
HC-05	HC-05 TXD	**D2**	
HC-05	HC-05 RXD	**D3**	Through a divider: 1k from D3 to RXD and 2.2k from RXD to GND, because RXD is a 3.3V input
HC-05	HC-05 VCC / GND	**5V / GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

Code

```
#include <SoftwareSerial.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>

#define DHTPIN 4
#define DHTTYPE DHT11

SoftwareSerial bt(2, 3);
DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);

int ldrPin = A0;
bool autoSend = false;
unsigned long lastSend = 0;
unsigned long interval = 3000;

void sendReadings()
{
  float temp = dht.readTemperature();
  float hum = dht.readHumidity();
  int light = analogRead(ldrPin);
  if (isnan(temp) || isnan(hum))
  {
    bt.println("Sensor error");
    return;
  }

  bt.print("TEMP=");
  bt.print(temp, 1);
  bt.print("C HUM=");
  bt.print(hum, 0);
  bt.print("% LIGHT=");
  bt.println(light);

  lcd.setCursor(0, 0);
  lcd.print("T:");
  lcd.print(temp, 1);
  lcd.print(" H:");
  lcd.print(hum, 0);
  lcd.print("   ");

  lcd.setCursor(0, 1);
  lcd.print("Light: ");
  lcd.print(light);
  lcd.print("    ");
}

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

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Dashboard ready");
  delay(2000);
  lcd.clear();
  bt.println("Send R to read once, A for auto, X to stop");
}

void loop()
{
  if (bt.available())
  {
    char command = bt.read();
    Serial.println(command);

    if (command == 'R' || command == 'r')
    {
      sendReadings();
    }
    else if (command == 'A' || command == 'a')
```

```
    {
      autoSend = true;
      bt.println("Auto mode ON");
    }
    else if (command == 'X' || command == 'x')
    {
      autoSend = false;
      bt.println("Auto mode OFF");
    }
  }

  if (autoSend && millis() - lastSend > interval)
  {
    lastSend = millis();
    sendReadings();
  }
}
```

What you should see: Send R and the phone receives TEMP=28.4C HUM=61% LIGHT=742. Send A and the same line arrives every 3 seconds. Send X to stop the stream. The LCD mirrors whatever was last sent.

Memory Sequence Game

Goal: Play the classic memory game: the board flashes a growing pattern of LEDs and you must repeat it exactly.

The whole game lives in one array of up to thirty steps and a variable saying how many of them are in play. Each round appends one new random step and replays the lot, then waits for that many button presses. Giving every colour its own tone means players remember the melody as well as the lights, which is why the original game was so hard to put down.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-50-memory-sequence-game

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 4 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Arrays, timing and game loops
- Growing a sequence one element per round
- Blocking input functions that wait for a press
- Giving audio and visual feedback together

Wiring

Part	Pin	Nano	Note
LED	Red anode	**D2**	Through a 220 ohm resistor
LED	Green anode	**D3**	Through a 220 ohm resistor
LED	Blue anode	**D4**	Through a 220 ohm resistor
LED	Yellow anode	**D5**	Through a 220 ohm resistor
LED	All cathodes	**GND**	
Button	Button 1	**D8 to GND**	
Button	Button 2	**D9 to GND**	
Button	Button 3	**D10 to GND**	
Button	Button 4	**D11 to GND**	
Passive buzzer	Buzzer (+)	**D12**	
Passive buzzer	Buzzer (−)	**GND**	

Code

```
int ledPins[4] = {2, 3, 4, 5};
int buttonPins[4] = {8, 9, 10, 11};
int tones[4] = {262, 330, 392, 523};
int buzzer = 12;

int sequence[30];
int level = 0;

void playStep(int index)
{
  digitalWrite(ledPins[index], HIGH);
  tone(buzzer, tones[index], 300);
  delay(400);
  digitalWrite(ledPins[index], LOW);
  delay(120);
}

int waitForPress()
{
  while (true)
  {
    for (int i = 0; i < 4; i++)
    {
      if (digitalRead(buttonPins[i]) == LOW)
      {
        playStep(i);
        while (digitalRead(buttonPins[i]) == LOW)
        {
          delay(10);
        }
        return i;
      }
    }
    delay(10);
  }
}

void gameOver()
{
  Serial.print("Game over at level ");
  Serial.println(level);

  for (int i = 0; i < 3; i++)
  {
    tone(buzzer, 120, 250);
    for (int j = 0; j < 4; j++)
    {
      digitalWrite(ledPins[j], HIGH);
    }
    delay(250);
    for (int j = 0; j < 4; j++)
    {
      digitalWrite(ledPins[j], LOW);
    }
    delay(250);
  }

  level = 0;
  delay(1500);
}

void setup()
{
  for (int i = 0; i < 4; i++)
  {
    pinMode(ledPins[i], OUTPUT);
    pinMode(buttonPins[i], INPUT_PULLUP);
  }
  pinMode(buzzer, OUTPUT);
  randomSeed(analogRead(A0));
  Serial.begin(9600);
  Serial.println("Watch the pattern, then repeat it");
}

void loop()
```

```
{
  sequence[level] = random(0, 4);
  level = level + 1;

  Serial.print("Level ");
  Serial.println(level);

  delay(700);
  for (int i = 0; i < level; i++)
  {
    playStep(sequence[i]);
  }

  for (int i = 0; i < level; i++)
  {
    int pressed = waitForPress();

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

  tone(buzzer, 700, 150);
  delay(600);
}
```

What you should see: One LED flashes with a tone, press the matching button. Each round adds one more step to remember. A wrong press flashes all four LEDs with a low buzz and restarts. Serial Monitor reports the level reached.

Mini Piano

Goal: Turn the keypad into a small instrument where each key sounds a different musical note.

Pitch is frequency: 262 Hz is middle C, and doubling a frequency raises it by exactly one octave, the 523 Hz at the end of the array is the C above. The keypad's sixteen keys map onto sixteen entries in a frequency array, so finding the right note is just a matter of locating the pressed key's position. tone() generates a square wave on the buzzer pin in the background, and noTone() silences it when the key is released.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-51-mini-piano

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Buzzer Module 3.3–5V
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Membrane Matrix Keypad
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Frequency mapping and keypad matrices
- How pitch relates to frequency and octaves
- tone() and noTone() for background sound
- Searching an array to find a key's position

Wiring

Part	Pin	Nano	Note
Keypad	Pin 1 (Row 1)	**D9**	
Keypad	Pin 2 (Row 2)	**D8**	
Keypad	Pin 3 (Row 3)	**D7**	
Keypad	Pin 4 (Row 4)	**D6**	
Keypad	Pin 5 (Col 1)	**D5**	
Keypad	Pin 6 (Col 2)	**D4**	
Keypad	Pin 7 (Col 3)	**D3**	
Keypad	Pin 8 (Col 4)	**D2**	
Passive buzzer	Piezo buzzer (+)	**D11**	
Passive buzzer	Piezo buzzer (−)	**GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

Code

```
#include <Keypad.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

const byte ROWS = 4;
const byte COLS = 4;

char keys[ROWS][COLS] =
{
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {9, 8, 7, 6};
byte colPins[COLS] = {5, 4, 3, 2};

Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
LiquidCrystal_I2C lcd(0x27, 16, 2);

int buzzer = 11;

char keyOrder[16] =
{'1','2','3','A','4','5','6','B','7','8','9','C','*','0','#','D'};
int noteFreq[16] = {262, 277, 294, 311, 330, 349, 370, 392,
  415, 440, 466, 494, 523, 554, 587, 622};
String noteName[16] = {"C4","C#4","D4","D#4","E4","F4","F#4","G4",
  "G#4","A4","A#4","B4","C5","C#5","D5","D#5"};

int findNote(char key)
{
  for (int i = 0; i < 16; i++)
  {
    if (keyOrder[i] == key)
    {
      return i;
    }
  }
  return -1;
}

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

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Mini Piano");
  lcd.setCursor(0, 1);
  lcd.print("Press any key");
}

void loop()
{
  char key = keypad.getKey();

  if (key == NO_KEY)
  {
    return;
  }

  int index = findNote(key);
  if (index < 0)
  {
    return;
  }

  tone(buzzer, noteFreq[index]);

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Note: ");
```

```
  lcd.print(noteName[index]);
  lcd.setCursor(0, 1);
  lcd.print(noteFreq[index]);
  lcd.print(" Hz");

  Serial.print(noteName[index]);
  Serial.print(" ");
  Serial.println(noteFreq[index]);

  // Hold the note while the key is down
  while (keypad.getState() == PRESSED)
  {
    keypad.getKey();
    delay(5);
  }

  noTone(buzzer);
}
```

What you should see: Press 1 for middle C: the LCD reads Note: C4 / 262 Hz. Each key up to D sounds a higher semitone. The note holds while the key is down and stops on release. Serial Monitor logs every note played.

Quiz Buzzer System

Goal: Build a three-contestant quiz buzzer that locks out everyone the instant the first player presses.

Fairness here is a matter of microseconds. The sketch does nothing at all in its polling loop except read three pins, so the gap between two presses that a human would call simultaneous is still easily distinguished. The moment a winner is found, the loop stops polling entirely, that is the lockout. Anything slow inside the polling loop, such as an LCD update, would blur the result, so the display is only touched after the decision is made.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-52-quiz-buzzer

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 4 x Tactile Push Button Switch
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Fast polling and why loop speed matters
- Locking out inputs after a decision
- Keeping slow operations out of a timing-critical loop

Wiring

Part	Pin	Nano	Note
Button	Player 1 button	**D2 to GND**	
Button	Player 2 button	**D3 to GND**	
Button	Player 3 button	**D4 to GND**	
LED	Player 1 LED	**D5**	Through a 220 ohm resistor
LED	Player 2 LED	**D6**	Through a 220 ohm resistor
LED	Player 3 LED	**D7**	Through a 220 ohm resistor
LED	All LED cathodes	**GND**	
Button	Reset button	**D8 to GND**	
Buzzer	Buzzer (+)	**D9**	
Buzzer	Buzzer (−)	**GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int buttonPins[3] = {2, 3, 4};
int ledPins[3] = {5, 6, 7};

int resetButton = 8;
int buzzer = 9;

LiquidCrystal_I2C lcd(0x27, 16, 2);

int winner = -1;
int scores[3] = {0, 0, 0};

void showReady()
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Ready... buzz!");
  lcd.setCursor(0, 1);
  lcd.print(scores[0]);
  lcd.print(" - ");
  lcd.print(scores[1]);
  lcd.print(" - ");
  lcd.print(scores[2]);

  for (int i = 0; i < 3; i++)
  {
    digitalWrite(ledPins[i], LOW);
  }
}

void setup()
{
  for (int i = 0; i < 3; i++)
  {
    pinMode(buttonPins[i], INPUT_PULLUP);
    pinMode(ledPins[i], OUTPUT);
  }
  pinMode(resetButton, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);

  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
  showReady();
}

void loop()
{
  if (winner < 0)
  {
    // Tight polling loop - nothing slow allowed in here
    for (int i = 0; i < 3; i++)
    {
      if (digitalRead(buttonPins[i]) == LOW)
      {
        winner = i;
        break;
      }
    }

    if (winner >= 0)
    {
      scores[winner] = scores[winner] + 1;
      digitalWrite(ledPins[winner], HIGH);
      digitalWrite(buzzer, HIGH);
      delay(500);
      digitalWrite(buzzer, LOW);

      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("PLAYER ");
      lcd.print(winner + 1);
      lcd.setCursor(0, 1);
```

```
      lcd.print("buzzed first!");

      Serial.print("Winner: player ");
      Serial.println(winner + 1);
    }
    return;
  }

  if (digitalRead(resetButton) == LOW)
  {
    winner = -1;
    delay(300);
    showReady();
  }
}
```

What you should see: LCD shows Ready... buzz! with the running score below. The first player to press lights their LED and the buzzer sounds. PLAYER 2 buzzed first! stays on screen and the others are locked out. The host presses reset to start the next question.

Electronic Voting Machine

Goal: Collect votes for three candidates, guard against double-counting from a single press, and show the result on demand.

A mechanical button does not close cleanly, its contacts bounce for a few milliseconds, and a fast Arduino can read that as five or six separate presses. Waiting 50 ms after the first LOW and then waiting for release is the simplest reliable debounce, and in a voting machine it is the difference between one vote and six. The tallies themselves are just integers being incremented, one per candidate.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-53-voting-machine

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 4 x Tactile Push Button Switch
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Button debouncing in software
- Incrementing counter variables
- Finding the largest value in an array
- Separating data entry from result display

Wiring

Part	Pin	Nano	Note
Button	Candidate A	**D2 to GND**	
Button	Candidate B	**D3 to GND**	
Button	Candidate C	**D4 to GND**	
Button	Result button	**D5 to GND**	
Buzzer	Buzzer (+)	**D8**	
Buzzer	Buzzer (−)	**GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int votePins[3] = {2, 3, 4};
int resultPin = 5;
int buzzer = 8;

String candidates[3] = {"CANDIDATE A", "CANDIDATE B", "CANDIDATE C"};
int votes[3] = {0, 0, 0};
int totalVotes = 0;

LiquidCrystal_I2C lcd(0x27, 16, 2);

void beep(int ms)
{
  digitalWrite(buzzer, HIGH);
  delay(ms);
  digitalWrite(buzzer, LOW);
}

void showPrompt()
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Cast your vote");
  lcd.setCursor(0, 1);
  lcd.print("Total: ");
  lcd.print(totalVotes);
}

void showResult()
{
  int winner = 0;
  for (int i = 1; i < 3; i++)
  {
    if (votes[i] > votes[winner])
    {
      winner = i;
    }
  }

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("A:");
  lcd.print(votes[0]);
  lcd.print(" B:");
  lcd.print(votes[1]);
  lcd.print(" C:");
  lcd.print(votes[2]);

  lcd.setCursor(0, 1);
  if (totalVotes == 0)
  {
    lcd.print("No votes yet");
  }
  else
  {
    lcd.print("Lead: ");
    lcd.print(candidates[winner].substring(10));
  }

  beep(300);
  delay(5000);
  showPrompt();
}

void setup()
{
  for (int i = 0; i < 3; i++)
  {
    pinMode(votePins[i], INPUT_PULLUP);
  }
  pinMode(resultPin, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);
```

```
  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
  showPrompt();
}

void loop()
{
  for (int i = 0; i < 3; i++)
  {
    if (digitalRead(votePins[i]) == LOW)
    {
      delay(50);                                          // Debounce
      if (digitalRead(votePins[i]) != LOW)
      {
        continue;
      }

      votes[i] = votes[i] + 1;
      totalVotes = totalVotes + 1;

      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Vote recorded");
      lcd.setCursor(0, 1);
      lcd.print(candidates[i]);
      beep(120);

      Serial.print("Vote for ");
      Serial.println(candidates[i]);

      while (digitalRead(votePins[i]) == LOW)             // Wait for release
      {
        delay(10);
      }
      delay(800);
      showPrompt();
    }
  }

  if (digitalRead(resultPin) == LOW)
  {
    delay(50);
    showResult();
  }
}
```

What you should see: LCD shows Cast your vote / Total: 0. Press a candidate button: Vote recorded / CANDIDATE A with a short beep. The total climbs by exactly one per press, however long you hold it. Press the result button: A:4 B:7 C:2 / Lead: B for five seconds.

Digital Roulette

Goal: Spin a ring of LEDs like a roulette wheel, slowing gradually until it settles on a random winner.

The illusion of a physical wheel losing momentum comes entirely from changing one number. The delay between LEDs starts at 30 ms and grows a little on every step; because it grows by a percentage rather than a fixed amount, the slowdown feels natural rather than mechanical. The modulo operator % wraps the position back to LED 1 after LED 8, turning a straight line of pins into a ring.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-54-digital-roulette

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Rapid delay manipulation to fake momentum
- The modulo operator for circular movement
- Randomising both the result and the run length
- Combining light and sound for feedback

Wiring

Part	Pin	Nano	Note
LED	LED 1	**D2**	Through a 220 ohm resistor
LED	LED 2	**D3**	Through a 220 ohm resistor
LED	LED 3	**D4**	Through a 220 ohm resistor
LED	LED 4	**D5**	Through a 220 ohm resistor
LED	LED 5	**D6**	Through a 220 ohm resistor
LED	LED 6	**D7**	Through a 220 ohm resistor
LED	LED 7	**D8**	Through a 220 ohm resistor
LED	LED 8	**D9**	Through a 220 ohm resistor
LED	All cathodes	**GND**	
Button	Button	**D10 to GND**	
Passive buzzer	Buzzer (+)	**D11**	
Passive buzzer	Buzzer (−)	**GND**	

Code

```
int ledPins[8] = {2, 3, 4, 5, 6, 7, 8, 9};
int button = 10;
int buzzer = 11;

void allOff()
{
  for (int i = 0; i < 8; i++)
  {
    digitalWrite(ledPins[i], LOW);
  }
}

void spin()
{
  int position = 0;
  int stepDelay = 30;
  int totalSteps = random(30, 50);

  for (int step = 0; step < totalSteps; step++)
  {
    allOff();
    digitalWrite(ledPins[position], HIGH);
    tone(buzzer, 1200, 15);

    delay(stepDelay);
    // Slow down a little more each step
    stepDelay = stepDelay + stepDelay / 8 + 2;
    position = (position + 1) % 8;
  }

  int winner = (position + 7) % 8;

  Serial.print("Winner: LED ");
  Serial.println(winner + 1);

  for (int i = 0; i < 4; i++)
  {
    digitalWrite(ledPins[winner], LOW);
    tone(buzzer, 500, 100);
    delay(150);
    digitalWrite(ledPins[winner], HIGH);
    delay(150);
  }
}

void setup()
{
  for (int i = 0; i < 8; i++)
  {
    pinMode(ledPins[i], OUTPUT);
  }
  pinMode(button, INPUT_PULLUP);
  pinMode(buzzer, OUTPUT);

  randomSeed(analogRead(A0));
  Serial.begin(9600);
  Serial.println("Press the button to spin");
  allOff();
}

void loop()
{
  if (digitalRead(button) == LOW)
  {
    while (digitalRead(button) == LOW)
    {
      delay(10);
    }

    spin();
  }
}
```

What you should see: Press the button and the light races around the ring. The rotation visibly slows over about three seconds. It stops on one LED, which then blinks four times with a tone. Serial Monitor prints Winner: LED 5.

Whack-a-Mole Style Game

Goal: Test reflexes: a random LED lights up and you must hit its matching button before the time runs out.

The waiting loop is the interesting part. It cannot use delay(), because the game must notice a button press the moment it happens, so it compares millis() against a deadline and checks all four buttons each time round. Every press is validated: hitting the wrong button costs a point, which is what makes input validation a game mechanic rather than a technicality. The time allowed shrinks each round, so the game gets harder as you get better.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-55-whack-a-mole

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 4 x Tactile Push Button Switch
- 1 x Buzzer Module 3.3–5V
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Randomisation and input validation
- Non-blocking deadlines with millis()
- Penalising wrong input as a game mechanic
- Scaling difficulty as the game progresses

Wiring

Part	Pin	Nano	Note
LED	LED 1	**D2**	Through a 220 ohm resistor
LED	LED 2	**D3**	Through a 220 ohm resistor
LED	LED 3	**D4**	Through a 220 ohm resistor
LED	LED 4	**D5**	Through a 220 ohm resistor
Button	Button 1	**D8 to GND**	
Button	Button 2	**D9 to GND**	
Button	Button 3	**D10 to GND**	
Button	Button 4	**D11 to GND**	
LED	All LED cathodes	**GND**	
Passive buzzer	Buzzer (+)	**D12**	
Passive buzzer	Buzzer (−)	**GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int ledPins[4] = {2, 3, 4, 5};

int buttonPins[4] = {8, 9, 10, 11};
int buzzer = 12;

LiquidCrystal_I2C lcd(0x27, 16, 2);

int score = 0;
int round_ = 0;
int allowed = 1500;                // ms to react

void showScore()
{
  lcd.setCursor(0, 0);
  lcd.print("Score: ");
  lcd.print(score);
  lcd.print("    ");

  lcd.setCursor(0, 1);
  lcd.print("Round: ");
  lcd.print(round_);
  lcd.print("   ");
}

void setup()
{
  for (int i = 0; i < 4; i++)
  {
    pinMode(ledPins[i], OUTPUT);
    pinMode(buttonPins[i], INPUT_PULLUP);
  }
  pinMode(buzzer, OUTPUT);

  randomSeed(analogRead(A0));
  Serial.begin(9600);

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Whack a Mole!");
  lcd.setCursor(0, 1);
  lcd.print("Get ready...");
  delay(2000);
  lcd.clear();
}

void loop()
{
  round_ = round_ + 1;

  delay(random(400, 1200));
  int mole = random(0, 4);
  digitalWrite(ledPins[mole], HIGH);

  unsigned long deadline = millis() + allowed;
  bool hit = false;

  while (millis() < deadline && !hit)
  {
    for (int i = 0; i < 4; i++)
    {
      if (digitalRead(buttonPins[i]) == LOW)
      {
        hit = true;

        if (i == mole)
        {
          score = score + 1;
          tone(buzzer, 900, 120);
          Serial.println("Hit");
        }
        else
```

```
        {
          score = score - 1;
          tone(buzzer, 200, 300);
          Serial.println("Wrong button");
        }

        while (digitalRead(buttonPins[i]) == LOW)
        {
          delay(10);
        }
      }
    }
  }

  if (!hit)
  {
    tone(buzzer, 150, 400);
    score = score - 1;
    Serial.println("Too slow");
  }

  digitalWrite(ledPins[mole], LOW);

  // Speed up, but never below 500 ms
  if (allowed > 500)
  {
    allowed = allowed - 40;
  }

  showScore();
}
```

What you should see: A random LED lights and you have 1.5 seconds to hit its button. A correct hit adds a point with a high beep. A wrong button or a miss costs a point with a low buzz. The reaction window shrinks by 40 ms each round, down to 500 ms.

Joystick Snake Game

Goal: Play a simplified Snake on the 8×8 LED matrix, steering with the joystick and growing each time you eat.

The snake is two arrays, snakeX and snakeY, holding the coordinates of each segment. Moving is done by shifting every segment into the position of the one ahead of it, then placing the new head, no segment is ever really moved, the values simply cascade backwards. Everything on the board is addressed as a column and row pair, which is 2D coordinate mapping: the same idea behind every tile-based game ever written.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-56-joystick-snake

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Buzzer Module 3.3–5V
- 1 x MAX7219 LED Display Module
- 1 x PS2 Thumb Joystick Module (KY-023 style)
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- 2D coordinate mapping on a matrix
- Moving a snake by cascading array values
- Preventing an illegal reverse turn
- Sampling input repeatedly during a game tick

Wiring

Part	Pin	Nano	Note
Matrix	VCC	**5V**	
Matrix	GND	**GND**	
Matrix	DIN	**D11**	
Matrix	CS	**D10**	
Matrix	CLK	**D13**	
Joystick	Joystick VCC / GND	**5V / GND**	
Joystick	Joystick VRx	**A0**	
Joystick	Joystick VRy	**A1**	
Passive buzzer	Buzzer (+)	**D8**	
Passive buzzer	Buzzer (−)	**GND**	

Code

```
#include <LedControl.h>

LedControl lc = LedControl(11, 13, 10, 1);

int xPin = A0;
int yPin = A1;
int buzzer = 8;

int snakeX[64];
int snakeY[64];
int length_ = 3;

int dirX = 1;
int dirY = 0;

int foodX;
int foodY;
int stepDelay = 400;

void placeFood()
{
  bool onSnake = true;
  while (onSnake)
  {
    foodX = random(0, 8);
    foodY = random(0, 8);
    onSnake = false;
    for (int i = 0; i < length_; i++)
    {
      if (snakeX[i] == foodX && snakeY[i] == foodY)
      {
        onSnake = true;
      }
    }
  }
}

void draw()
{
  lc.clearDisplay(0);
  for (int i = 0; i < length_; i++)
  {
    lc.setLed(0, snakeY[i], snakeX[i], true);
  }
  lc.setLed(0, foodY, foodX, true);
}

void readJoystick()
{
  int x = analogRead(xPin) - 512;
  int y = analogRead(yPin) - 512;

  if (x > 300 && dirX == 0)      { dirX = 1; dirY = 0; }
  else if (x < -300 && dirX == 0){ dirX = -1; dirY = 0; }
  else if (y > 300 && dirY == 0) { dirX = 0; dirY = -1; }
  else if (y < -300 && dirY == 0){ dirX = 0; dirY = 1; }
}

void gameOver()
{
  for (int i = 0; i < 3; i++)
  {
    tone(buzzer, 150, 200);
    for (int r = 0; r < 8; r++)
    {
      lc.setRow(0, r, B11111111);
    }
    delay(200);
    lc.clearDisplay(0);
    delay(200);
  }
  length_ = 3;
  dirX = 1;
  dirY = 0;
  stepDelay = 400;
```

```
  for (int i = 0; i < length_; i++)
  {
    snakeX[i] = 3 - i;
    snakeY[i] = 4;
  }
  placeFood();
  delay(800);
}

void setup()
{
  lc.shutdown(0, false);
  lc.setIntensity(0, 4);
  lc.clearDisplay(0);

  pinMode(buzzer, OUTPUT);
  randomSeed(analogRead(A2));

  for (int i = 0; i < length_; i++)
  {
    snakeX[i] = 3 - i;
    snakeY[i] = 4;
  }
  placeFood();
}

void loop()
{
  // Read the stick several times while waiting, so turns feel responsive
  for (int i = 0; i < 20; i++)
  {
    readJoystick();
    delay(stepDelay / 20);
  }

  int newX = snakeX[0] + dirX;
  int newY = snakeY[0] + dirY;

  if (newX < 0 || newX > 7 || newY < 0 || newY > 7)
  {
    gameOver();
    return;
  }
  for (int i = 0; i < length_; i++)
  {
    if (snakeX[i] == newX && snakeY[i] == newY)
    {
      gameOver();
      return;
    }
  }

  // Shift every segment back one place
  for (int i = length_; i > 0; i--)
  {
    snakeX[i] = snakeX[i - 1];
    snakeY[i] = snakeY[i - 1];
  }
  snakeX[0] = newX;
  snakeY[0] = newY;

  if (newX == foodX && newY == foodY)
  {
    length_ = length_ + 1;
    tone(buzzer, 900, 80);
    placeFood();

    if (stepDelay > 150)
    {
      stepDelay = stepDelay - 20;
    }
  }
```

```
  draw();
}
```

What you should see: A three-dot snake starts moving right with one dot of food on the board. Push the joystick to steer; the snake cannot reverse into itself. Eating the food adds a segment, beeps, and speeds the game up slightly. Hitting a wall or your own tail flashes the whole matrix and restarts.

Smart Room Light

Goal: Switch a real lamp through a relay when the room gets dark, with a delay that stops it flickering at dusk.

Experiment 19 switched LEDs the instant the reading crossed the threshold. A relay driving a real lamp cannot do that, at dusk the light level hovers around the threshold and the relay would chatter, wearing out its contacts in minutes. The fix is a confirmation delay: the new condition must hold continuously for three seconds before the relay is allowed to move. Recording the time the condition first appeared and comparing it with millis() is all that takes.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-57-smart-room-light

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 10k resistor
- 1 x LCD1602 16x2 Character LCD Display
- 1 x 5 mm LDR Light Dependent Resistor
- 1 x 5V Relay Module with Optocoupler
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Light automation with relays
- Confirmation delays to stop relay chatter
- Two thresholds plus a timer for stability
- Why real switching gear needs gentler logic than an LED

Wiring

Part	Pin	Nano	Note
LDR divider	LDR leg 1	**5V**	
LDR divider	LDR leg 2	**A0 and one end of 10kΩ**	
LDR divider	Other end of 10kΩ	**GND**	
Relay	Relay VCC / GND	**5V / GND**	
Relay	Relay IN	**D7**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	
Relay	COM	**Nano 5V**	Demo load only: never mains
Lamp LED	Anode	**Relay NO**	Through a 220 ohm resistor
Lamp LED	Cathode	**GND**	

> Switch only low-voltage demo loads such as the LED here. Never wire mains to these relay modules in a classroom project.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int ldrPin = A0;
int relayPin = 7;

int darkBelow = 350;
int brightAbove = 450;
unsigned long confirmTime = 3000;

bool lampOn = false;
bool pendingState = false;
unsigned long pendingSince = 0;

LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup()
{
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH);                          // Active LOW relay - lamp OFF
  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Smart Room Light");
  delay(2000);
  lcd.clear();
}

void loop()
{
  int light = analogRead(ldrPin);

  bool wantOn = lampOn;
  if (light < darkBelow)
  {
    wantOn = true;
  }
  if (light > brightAbove)
  {
    wantOn = false;
  }

  if (wantOn != lampOn)
  {
    if (wantOn != pendingState)
    {
      pendingState = wantOn;
      pendingSince = millis();
    }
    else if (millis() - pendingSince > confirmTime)
    {
      lampOn = wantOn;
      digitalWrite(relayPin, lampOn ? LOW : HIGH);

      Serial.print("Lamp switched ");
      Serial.println(lampOn ? "ON" : "OFF");
    }
  }
  else
  {
    pendingState = lampOn;
  }

  lcd.setCursor(0, 0);
  lcd.print("Light: ");
  lcd.print(light);
  lcd.print("    ");

  lcd.setCursor(0, 1);
  lcd.print("Lamp: ");
  lcd.print(lampOn ? "ON " : "OFF");
  lcd.print("       ");
```

```
  delay(200);
}
```

What you should see: Light: 720 / Lamp: OFF in a lit room. Cover the LDR and hold it: after 3 seconds the relay clicks and Lamp: ON appears. A brief shadow does nothing at all, the delay filters it out.

Automatic Bathroom Light

Goal: Turn the light on when someone walks in and off again a set time after they leave, resetting the timer on every movement.

The key idea is the retriggerable timer. Every time motion is seen, lastMotion is set to the current millis(), so the countdown restarts rather than continuing. The light only goes out once a full quiet period has passed with no movement at all. This is exactly how commercial occupancy sensors behave, and it is why they do not switch off while you are still standing there.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-58-automatic-bathroom-light

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x LCD1602 16x2 Character LCD Display
- 1 x 5V Relay Module with Optocoupler
- 1 x Generic Infrared Obstacle Avoidance Sensor Module (Active Low) – LM393 Comparator, 2-30cm Range
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- IR triggers and retriggerable timers
- Restarting a countdown on every event
- Displaying remaining time from a millis() difference
- Modelling real occupancy-sensor behaviour

Wiring

Part	Pin	Nano	Note
IR sensor	VCC	**5V**	
IR sensor	GND	**GND**	
IR sensor	OUT	**D2**	
Relay	Relay VCC / GND	**5V / GND**	
Relay	Relay IN	**D7**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	
Relay	COM	**Nano 5V**	Demo load only: never mains
Lamp LED	Anode	**Relay NO**	Through a 220 ohm resistor
Lamp LED	Cathode	**GND**	

> Switch only low-voltage demo loads such as the LED here. Never wire mains to these relay modules in a classroom project.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int sensorPin = 2;
int relayPin = 7;

unsigned long holdTime = 15000;            // Stay on 15 s after last motion
unsigned long lastMotion = 0;
bool lightOn = false;

LiquidCrystal_I2C lcd(0x27, 16, 2);

void setup()
{
  pinMode(sensorPin, INPUT);
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH);   // Lamp OFF

  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Auto Bathroom");
  delay(2000);
  lcd.clear();
}

void loop()
{
  bool motion = (digitalRead(sensorPin) == LOW);

  if (motion)
  {
    lastMotion = millis();                    // Retrigger the timer

    if (!lightOn)
    {
      lightOn = true;
      digitalWrite(relayPin, LOW);
      Serial.println("Motion detected - light ON");
    }
  }

  if (lightOn && millis() - lastMotion > holdTime)
  {
    lightOn = false;
    digitalWrite(relayPin, HIGH);
    Serial.println("No motion - light OFF");
  }

  lcd.setCursor(0, 0);
  if (lightOn)
  {
    lcd.print("Light: ON              ");

    long remaining = (holdTime - (millis() - lastMotion)) / 1000;
    if (remaining < 0)
    {
      remaining = 0;
    }

    lcd.setCursor(0, 1);
    lcd.print("Off in: ");
    lcd.print(remaining);
    lcd.print("s    ");
  }
  else
  {
    lcd.print("Light: OFF               ");
    lcd.setCursor(0, 1);
    lcd.print("Waiting...               ");
  }
```

```
  delay(150);
}
```

What you should see: Wave a hand in front of the sensor: the relay clicks and Light: ON appears. The second line counts down from 15 seconds. Move again and the countdown restarts from 15. Stay still for the full 15 seconds and the lamp switches off.

Smart Doorbell

Goal: Play a proper melody through a piezo buzzer when someone touches the doorbell pad, and count the visitors.

A melody is two parallel arrays: one of frequencies and one of durations. Walking through them with a for loop and calling tone() for each pair plays the tune. The gap after each note, here 30 percent longer than the note itself, is what stops the melody sounding like one continuous smear. Note that the touch pad works through several millimetres of plastic or wood, so the sensor can be hidden behind a real door panel.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-59-smart-doorbell

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x Capacitive Touch Sensor Module
- 1 x Buzzer Module 3.3–5V
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Touch sensing and piezo melodies
- Parallel arrays for pitch and duration
- Why notes need a gap between them
- Counting events with a persistent variable

Wiring

Part	Pin	Nano	Note
Touch module	VCC	**5V**	
Touch module	GND	**GND**	
Touch module	SIG	**D2**	
Passive buzzer	Piezo buzzer (+)	**D8**	
Passive buzzer	Piezo buzzer (−)	**GND**	
LED	LED anode	**D7**	Through a 220 ohm resistor
LED	LED cathode	**GND**	
LCD	LCD SDA	**A4**	
LCD	LCD SCL	**A5**	

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int touchPin = 2;
int buzzer = 8;
int led = 7;

LiquidCrystal_I2C lcd(0x27, 16, 2);

int melody[8] =    {659, 587, 370, 415, 554, 494, 294, 330};
int durations[8] = {300, 300, 600, 300, 300, 600, 300, 700};

int visitors = 0;

void playChime()
{
  for (int i = 0; i < 8; i++)
  {
    tone(buzzer, melody[i], durations[i]);
    digitalWrite(led, HIGH);
    delay(durations[i] * 0.6);
    digitalWrite(led, LOW);
    delay(durations[i] * 0.7);
  }
  noTone(buzzer);
}

void showIdle()
{
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Touch to ring");
  lcd.setCursor(0, 1);
  lcd.print("Visitors: ");
  lcd.print(visitors);
}

void setup()
{
  pinMode(touchPin, INPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(led, OUTPUT);

  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
  showIdle();
}

void loop()
{
  if (digitalRead(touchPin) == HIGH)
  {
    visitors = visitors + 1;

    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("DING DONG!");
    lcd.setCursor(0, 1);
    lcd.print("Someone is here");

    Serial.print("Doorbell pressed. Visitor ");
    Serial.println(visitors);
    playChime();

    // Ignore a finger left resting on the pad
    while (digitalRead(touchPin) == HIGH)
    {
      delay(20);
    }

    delay(1000);
    showIdle();
```

```
  }
}
```

What you should see: LCD shows Touch to ring / Visitors: 0. Touch the pad: DING DONG! appears and an eight-note chime plays. The LED pulses in time with the melody. The visitor count increases by one per ring, however long the finger rests.

Intelligent Water Saver

Goal: Water the plant only when it is genuinely dry, cap how long the pump may run, and refuse to run again too soon.

Experiment 22 watered whenever the soil was dry, which sounds sensible until you realise the sensor sits in the wet patch right where the water lands, so the reading rises long before the rest of the pot is watered, and the pump cycles repeatedly. Two timed rules fix it: the pump may never run longer than eight seconds at a stretch, and after watering it must wait a full minute before it is allowed to consider running again. That soak period lets the water spread and the reading settle.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-60-intelligent-water-saver

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Capacitive Soil Moisture Sensor – 3.3V/5V Analog Output, PH2.54-3P Interface
- 1 x 5V Relay Module with Optocoupler
- 1 x Mini Submersible Water Pump 3–6V
- 1 x L298N Dual H-Bridge Motor Driver Module
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)
- 1 x 18650 Battery Holder
- 2 x 18650 Li-ion Rechargeable Cell

What you learn

- Timed relay events
- Capping how long an actuator may run
- Cool-down periods between actions
- Why a sensor's position changes the logic you need

Wiring

Part	Pin	Nano	Note
Soil sensor	Soil sensor VCC / GND	**5V / GND**	
Soil sensor	Soil sensor AO	**A0**	
Relay	Relay VCC / GND	**5V / GND**	
Relay	Relay IN	**D7**	
LCD	LCD SDA / SCL	**A4 / A5**	
L298N	+12V	**Battery (+) from the 2 x 18650 holder**	5V jumper stays on: the board's regulator makes 5V
L298N	GND	**Battery (−) and Nano GND**	The common ground is required
Relay	COM	**L298N +5V**	The pump never runs from the Nano's own 5V
Relay	NO	**Pump red wire (+)**	
Pump	Black wire (−)	**L298N GND**	

The pump runs at 5V from the L298N's on-board regulator, which is fed by the 2 x 18650 pack. The relay switches that 5V; the Nano only drives the relay.

Keep water away from the Nano and test the pump and tube over a tray first.

Charge 18650 cells only in a proper Li-ion charger and never short the holder's wires.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

int soilPin = A0;
int relayPin = 7;

int dryLevel = 30;
unsigned long maxRunTime = 8000;                              // Never pump longer than 8 s
unsigned long soakTime = 60000;                               // Wait 60 s before pumping again

bool pumping = false;
unsigned long pumpStarted = 0;
unsigned long lastPumpEnd = 0;
int wateringCount = 0;

LiquidCrystal_I2C lcd(0x27, 16, 2);

void pumpOff()

{
  digitalWrite(relayPin, HIGH);
  pumping = false;
  lastPumpEnd = millis();
}

void setup()
{
  pinMode(relayPin, OUTPUT);
  digitalWrite(relayPin, HIGH);

  Serial.begin(9600);
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Water Saver");
  delay(2000);
  lcd.clear();

  lastPumpEnd = millis();
}

void loop()
{
  int raw = analogRead(soilPin);
  int moisture = constrain(map(raw, 1023, 300, 0, 100), 0, 100);

  if (pumping)
  {
    if (millis() - pumpStarted > maxRunTime)
    {
      pumpOff();
      Serial.println("Run time limit reached - pump stopped");
    }
  }
  else
  {
    bool soaked = (millis() - lastPumpEnd < soakTime);

    if (moisture < dryLevel && !soaked)
    {
      digitalWrite(relayPin, LOW);
      pumping = true;
      pumpStarted = millis();
      wateringCount = wateringCount + 1;

      Serial.print("Watering cycle ");
      Serial.println(wateringCount);
    }
  }

  lcd.setCursor(0, 0);
  lcd.print("Soil: ");
  lcd.print(moisture);
  lcd.print("% N:");
  lcd.print(wateringCount);
```

```
  lcd.print(" ");

  lcd.setCursor(0, 1);
  if (pumping)
  {
    long left = (maxRunTime - (millis() - pumpStarted)) / 1000;
    lcd.print("PUMPING ");
    lcd.print(left);
    lcd.print("s     ");
  }
  else if (millis() - lastPumpEnd < soakTime)
  {
    long wait = (soakTime - (millis() - lastPumpEnd)) / 1000;
    lcd.print("Soaking ");
    lcd.print(wait);
    lcd.print("s   ");
  }
  else
  {
    lcd.print("Monitoring      ");
  }

  delay(300);
}
```

What you should see: Soil: 55% N:0 / Monitoring with damp soil. Below 30% the relay clicks: PUMPING 8s counting down. After the run it shows Soaking 60s and refuses to pump again. The cycle counter N tracks how many times it has watered.

Mini Smart Home Demo

Goal: Bring it all together: lighting, climate, security and a rotating status display running as one integrated system.

Four independent jobs share one board, so nothing may block. Every task keeps its own "last run" timestamp and checks millis() to decide whether its turn has come, the sensors are read every two seconds, the display rotates every three, and the security check runs constantly. A single delay(2000) anywhere in here would freeze the alarm for two seconds. This is cooperative scheduling, and it is the step between a project that does one thing and a system that does several.

Online page with troubleshooting: geekobot.in/projects/explorer-kit-61-mini-smart-home

Parts

- 1 x Generic Nano CH340 Development Board (Soldered)
- 1 x Solderless Breadboard
- 1 x Dupont Jumper Wires (40 pcs)
- 1 x Through-Hole LED Pack (Pack of 10)
- 1 x 220 ohm resistors
- 1 x 10k resistor
- 1 x Tactile Push Button Switch
- 1 x PCB Mounted Active Buzzer Module
- 1 x LCD1602 16x2 Character LCD Display
- 1 x Generic DHT11 Digital Temperature & Humidity Sensor Module
- 1 x 5 mm LDR Light Dependent Resistor
- 2 x 5V Relay Module with Optocoupler
- 1 x Generic Infrared Obstacle Avoidance Sensor Module (Active Low) – LM393 Comparator, 2-30cm Range
- 1 x Arduino Nano IO Expansion Shield (328P Compatible)

What you learn

- Multi-system integration and scheduling
- Cooperative task scheduling with millis()
- Giving a safety task priority over everything else
- Rotating one display between several data sources

Wiring

Part	Pin	Nano	Note
DHT11	DHT11 DATA	**D4**	
LDR divider	LDR leg 1	**5V**	
LDR divider	LDR leg 2	**A0**	And one end of the 10k resistor
LDR divider	Other end of the 10k	**GND**	
IR sensor	IR sensor OUT	**D2**	
IR sensor	All VCC / GND	**5V / GND**	
Relay	Relay VCC / GND	**5V / GND**	
Buzzer	Buzzer (+)	**D9**	
LED	Red LED anode	**D6**	Through a 220 ohm resistor
Button	Arm / disarm button	**D5 to GND**	
LCD	LCD SDA / SCL	**A4 / A5**	
Buzzer	Buzzer (−)	**GND**	
LED	Red LED cathode	**GND**	

Relay 1	IN	**D7**	First relay module (Light)
Relay 2	IN	**D8**	Second relay module (fan)
Relays 1 and 2	COM	**Nano 5V**	Demo loads only: never mains
Load LEDs	Anodes	**Relay 1 NO and relay 2 NO**	Each through a 220 ohm resistor
Load LEDs	Cathodes	**GND**	

> Switch only low-voltage demo loads such as the LED here. Never wire mains to these relay modules in a classroom project.

Code

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <DHT.h>

#define DHTPIN 4
#define DHTTYPE DHT11

DHT dht(DHTPIN, DHTTYPE);
LiquidCrystal_I2C lcd(0x27, 16, 2);

int ldrPin = A0;
int irPin = 2;
int lightRelay = 7;
int fanRelay = 8;
int buzzer = 9;
int alarmLed = 6;
int armButton = 5;

float temp = 0;
float hum = 0;
int lightLevel = 0;

bool lightOn = false;
bool fanOn = false;
bool armed = false;
bool intruder = false;

int darkBelow = 350;
float fanAbove = 30.0;

unsigned long lastSensorRead = 0;
unsigned long lastScreenChange = 0;
int screen = 0;

void readSensors()
{
  float t = dht.readTemperature();
  float h = dht.readHumidity();

  if (!isnan(t)) temp = t;
  if (!isnan(h)) hum = h;

  lightLevel = analogRead(ldrPin);
}

void runLighting()
{
  lightOn = (lightLevel < darkBelow);
  digitalWrite(lightRelay, lightOn ? LOW : HIGH);
}

void runClimate()
{
  fanOn = (temp > fanAbove);
  digitalWrite(fanRelay, fanOn ? LOW : HIGH);
}

void runSecurity()
{
  if (digitalRead(armButton) == LOW)
  {
    armed = !armed;
    if (!armed)
    {
      intruder = false;
    }
    delay(300);
  }

  if (armed && digitalRead(irPin) == LOW)
  {
    intruder = true;
  }

  if (intruder)
```

```
  {
    digitalWrite(alarmLed, (millis() / 250) % 2);
    digitalWrite(buzzer, (millis() / 250) % 2);
  }
  else
  {
    digitalWrite(alarmLed, LOW);
    digitalWrite(buzzer, LOW);
  }
}

void updateDisplay()
{
  lcd.clear();

  if (screen == 0)
  {
    lcd.setCursor(0, 0);
    lcd.print("Temp: ");
    lcd.print(temp, 1);
    lcd.write(223);
    lcd.print("C");
    lcd.setCursor(0, 1);
    lcd.print("Humidity: ");
    lcd.print(hum, 0);
    lcd.print("%");
  }
  else if (screen == 1)
  {
    lcd.setCursor(0, 0);
    lcd.print("Light: ");
    lcd.print(lightOn ? "ON" : "OFF");
    lcd.setCursor(0, 1);
    lcd.print("Fan:   ");
    lcd.print(fanOn ? "ON" : "OFF");
  }
  else
  {
    lcd.setCursor(0, 0);
    lcd.print("Security: ");
    lcd.print(armed ? "ARMED" : "OFF");
    lcd.setCursor(0, 1);
    lcd.print(intruder ? "!! INTRUDER !!" : "All clear");
  }
}

void setup()
{
  pinMode(irPin, INPUT);
  pinMode(armButton, INPUT_PULLUP);
  pinMode(lightRelay, OUTPUT);
  pinMode(fanRelay, OUTPUT);
  pinMode(buzzer, OUTPUT);
  pinMode(alarmLed, OUTPUT);

  digitalWrite(lightRelay, HIGH);
  digitalWrite(fanRelay, HIGH);

  dht.begin();
  Serial.begin(9600);

  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Mini Smart Home");
  lcd.setCursor(0, 1);
  lcd.print("Starting up...");
  delay(2500);

  readSensors();
  updateDisplay();
}

void loop()
{
```

```
  // Security runs every pass - it must never be delayed
  runSecurity();

  if (millis() - lastSensorRead > 2000)
  {
    lastSensorRead = millis();
    readSensors();
    runLighting();
    runClimate();

    Serial.print("T=");
    Serial.print(temp, 1);
    Serial.print(" H=");
    Serial.print(hum, 0);
    Serial.print(" L=");
    Serial.println(lightLevel);
  }

  if (millis() - lastScreenChange > 3000)
  {
    lastScreenChange = millis();
    screen = (screen + 1) % 3;
    updateDisplay();
  }
}
```

What you should see: The LCD cycles through climate, appliances and security every 3 seconds. Cover the LDR and the light relay clicks in automatically. Warm the DHT11 past 30°C and the fan relay engages. Press the button to arm; break the IR beam and the buzzer and LED pulse until you disarm.

WHAT NEXT

Keep building

Every project in this book follows one pattern: read an input, make a decision, drive an output. The robot car, the locks and the smart home are the same idea with more parts. Now remix them:

- Put the HC-SR04 on the line-following car so it stops for obstacles.
- Show the plant watering readings on your phone over Bluetooth.
- Add the RFID reader to the smart home demo so only your card disarms it.
- Move up to the ESP32 for Wi-Fi, an OLED screen and motion sensing.

More than 1,700 free project guides with parts lists, wiring and code: **geekobot.in/projects**

Geekobot
Imagine. Build. Innovate.

Build. Learn. Innovate.