Programmed Control with Arduino

Activity 7: The Interactive Traffic Light (Pedestrian Request)

Up until now, we have programmed systems that operated in isolation: they either responded immediately to a button press or followed a fixed timing sequence. Today, we will merge both worlds to design an interactive traffic light. By default, vehicles will always have the right of way (continuous green light), but when a pedestrian presses the pushbutton, the Arduino will interrupt the normal flow to stop traffic and allow a safe crossing.

Learning Objectives

By the end of this session, you will be able to:

- Integrate input components (pushbutton) and output components (LEDs) into the same coordinated system.
- Design a state-based algorithm where the program "actively waits" for a user action.
- Critically analyze microcontroller behavior when combining sensor readings with long-duration time pauses (`delay`).
- Develop logical solutions to simulate right-of-way priorities in automated urban environments.

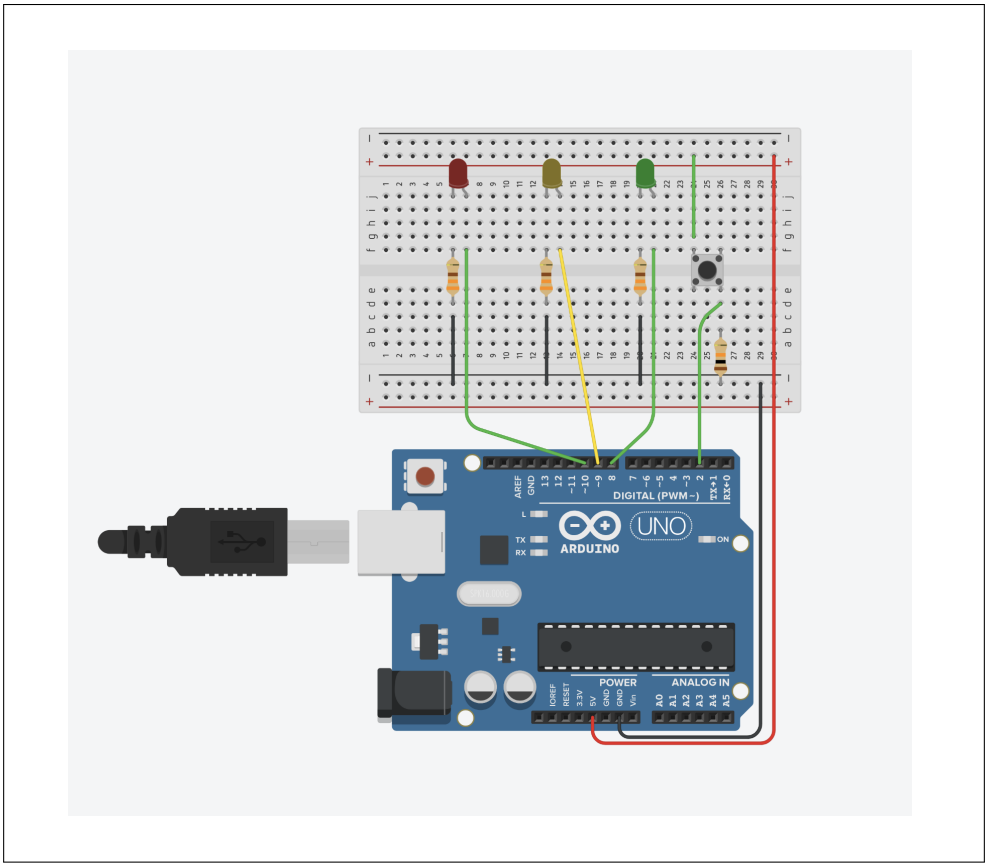
Required Components

Find and place the following components on your TinkerCad workspace:

- 1 Arduino Uno board.
- 1 Small breadboard.
- 3 LEDs (1 Red, 1 Yellow, 1 Green).
- 3 $220\,\Omega$ resistors (to protect the LEDs).
- 1 Four-pin pushbutton.
- 1 $10\,\text{k}\Omega$ resistor (for the pushbutton's Pull-Down configuration).
- Virtual jumper wires.

Breadboard Assembly Diagram

This circuit is the exact combination of the assemblies from Activity 3 and Activity 5. Make sure to wire everything neatly to avoid short circuits!



Base Code: Active Flow Waiting

Copy the following program into the TinkerCad editor. Notice that inside the `loop` function there is no initial time delay; the Arduino simply monitors the button at the highest possible speed.

```
// Output pin definitions for LEDs
int redLed = 10;
int yellowLed = 9;
int greenLed = 8;

// Input pin definition for the button
int pedestrianButton = 2;

void setup() {
  pinMode(redLed, OUTPUT);
  pinMode(yellowLed, OUTPUT);
  pinMode(greenLed, OUTPUT);
  pinMode(pedestrianButton, INPUT);

  // Default initial state: open road for cars
  digitalWrite(greenLed, HIGH);
  digitalWrite(yellowLed, LOW);
  digitalWrite(redLed, LOW);
}

void loop() {
  // Continuously read the button state
  int crossingRequest = digitalRead(pedestrianButton);

  // If a pedestrian presses the button, start the stop sequence
  if (crossingRequest == HIGH) {
    delay(1000); // Brief courtesy delay before reacting

    // 1. Transition from Green to Yellow
    digitalWrite(greenLed, LOW);
    digitalWrite(yellowLed, HIGH);
    delay(2000); // 2 seconds of warning yellow

    // 2. Transition from Yellow to Red (Cars stopped / Pedestrians cross)
    digitalWrite(yellowLed, LOW);
    digitalWrite(redLed, HIGH);
    delay(5000); // 5 seconds for pedestrians to cross safely

    // 3. Return to open road state (Cars proceed)
    digitalWrite(redLed, LOW);
    digitalWrite(greenLed, HIGH);
  }
}
```

Listing 1: Code for an interactive traffic light with a pedestrian request.

How does the code work?

The secret of this program lies within the structure of the `loop` function:

- **Non-blocking waiting**: As long as no one presses the button, the microcontroller reads the line `int digitalRead`, verifies it returns `LOW`, ignores the `if` block, and restarts the loop. This happens thousands of times per second, so the green LED remains lit without any interruption.
- **The temporal insensitivity problem**: Once a pedestrian presses the button and we enter the `if` block, sequential flow takes over, executing the commands `delay` and `delay`. During these 7 total seconds, the Arduino is "frozen" executing the pauses. If another pedestrian were to press the button during this interval, the Arduino would not notice, as it cannot execute the `digitalRead` line while asleep due to a `delay`.

Activity 7 Challenge

Put your skills to the test by solving these two logical improvements in urban engineering:

1. **Traffic Courtesy Time (Cool-Down)**: In a real city, if a pedestrian presses the button, crosses, and right after finishing another pedestrian presses it again, traffic would gridlock. Modify the code to add a mandatory waiting time of **5 seconds** at the end of the cycle (after turning the green light back on) during which the system completely ignores any button presses.
2. **Acoustic or Visual End-of-Crossing Warning**: Add a rapid blinking effect to the red LED during the last second of its activation to visually warn pedestrians that their crossing time is about to expire.
3. *Question for reflection*: How would you solve the button insensitivity problem without resorting to complex hardware? If you divided a long delay like `delay(5000)` into small steps of `delay(100)`, how could this help check the button state more frequently?