Controlo programado com Arduino

Prática 7: O Semáforo Interativo (Pedido de Passagem)

Até agora programámos sistemas que funcionavam de forma isolada: ou respondiam imediatamente a um botão, ou seguiam uma sequência fixa no tempo. Hoje uniremos ambos os mundos para desenhar um semáforo interativo. Por defeito, os veículos terão sempre via livre (luz verde contínua), mas quando um peão pressionar o botão de pressão, o Arduino interromperá o fluxo normal para deter o trânsito e permitir um cruzamento seguro.

Objetivos de Aprendizagem

Ao finalizar esta sessão, serás capaz de:

- Integrar componentes de entrada (botão de pressão) e de saída (LEDs) num mesmo sistema coordenado.
- Desenhar um algoritmo baseado em estados onde o programa "espera de forma ativa" uma ação do utilizador.
- Analisar criticamente o comportamento de um microcontrolador quando se combinam leituras de sensores com pausas de tempo de longa duração (`delay`).
- Desenvolver soluções lógicas para simular prioridades de passagem em ambientes automatizados urbanos.

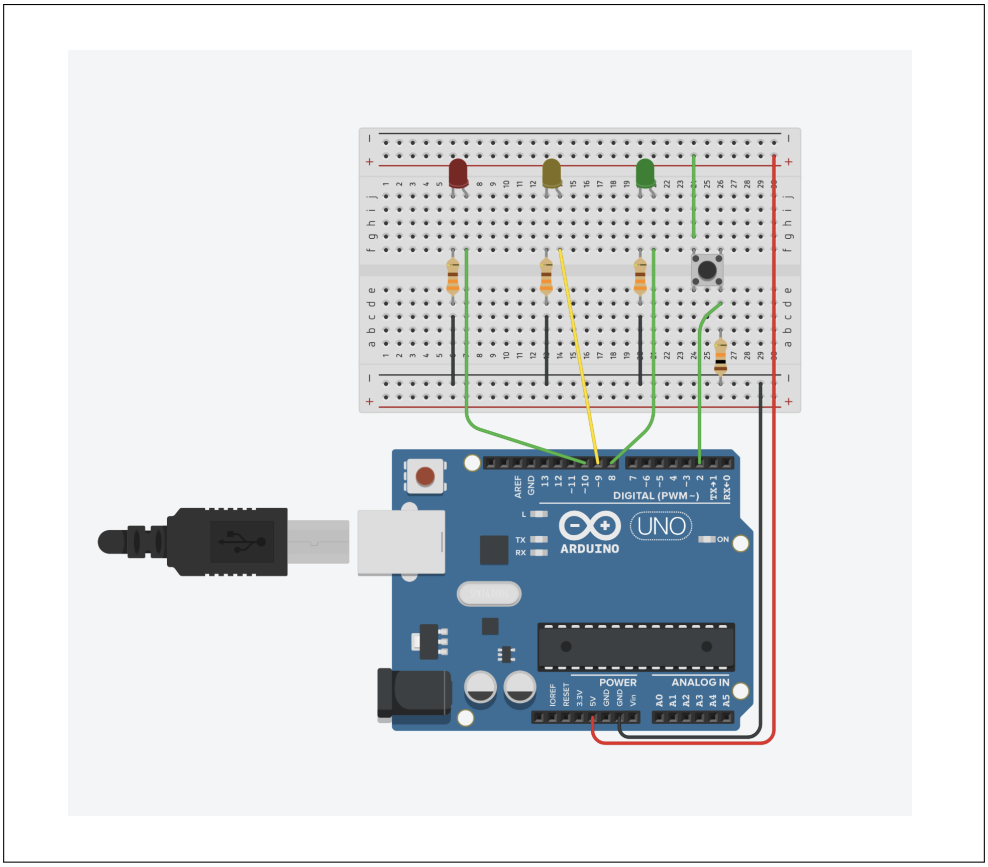
Componentes Necessários

Procura e coloca os seguintes componentes na tua mesa de trabalho do TinkerCad:

- 1 Placa Arduino Uno.
- 1 Placa de ensaio pequena (Protoboard).
- 3 Díodos LED (1 Vermelho, 1 Amarelo, 1 Verde).
- 3 Resistências de $220\,\Omega$ (para proteger os LEDs).
- 1 Botão de pressão de quatro pinos.
- 1 Resistência de $10\,\text{k}\Omega$ (para a configuração Pull-Down do botão).
- Cabos de ligação virtuais.

Esquema de Montagem na Protoboard

Este circuito é a combinação exata das montagens da Prática 3 e da Práctica 5. Garante que a cablagem seja feita com ordem para evitar cruzamentos!



Código Base: Espera Ativa do Fluxo

Copia o seguinte programa no editor do TinkerCad. Observa que na função `loop` não existe nenhum atraso de tempo inicial; o Arduino limita-se a vigiar o botão à máxima velocidade possível.

```
// Definicao de pinos de saida para os LEDs
int ledVermelho = 10;
int ledAmarelo = 9;
int ledVerde = 8;

// Definicao do pino de entrada para o botao
int botonPeaton = 2;

void setup() {
  pinMode(ledVermelho, OUTPUT);
  pinMode(ledAmarelo, OUTPUT);
  pinMode(ledVerde, OUTPUT);
  pinMode(botonPeaton, INPUT);

  // Estado inicial por defeito: via livre para os carros
  digitalWrite(ledVerde, HIGH);
  digitalWrite(ledAmarelo, LOW);
  digitalWrite(ledVermelho, LOW);
}

void loop() {
  // Lemos continuamente o estado do botao
  int peticionPaso = digitalRead(botonPeaton);

  // Se un peao pressionar o botao, iniciamos a sequencia de paragem
  if (peticionPaso == HIGH) {
    delay(1000); // Pequeno tempo de cortesia antes de reagir

    // 1. Passar de Verde para Amarelo
    digitalWrite(ledVerde, LOW);
    digitalWrite(ledAmarelo, HIGH);
    delay(2000); // 2 segundos em amarelo de aviso

    // 2. Passar de Amarelo para Vermelho (Carros detidos / Peoes cruzam)
    digitalWrite(ledAmarelo, LOW);
    digitalWrite(ledVermelho, HIGH);
    delay(5000); // 5 segundos para que os peoes cruzem em seguranca

    // 3. Voltar ao estado de via livre (Carros avancam)
    digitalWrite(ledVermelho, LOW);
    digitalWrite(ledVerde, HIGH);
  }
}
```

Listing 1: Código para um semáforo interativo com pedido de passagem.

Como funciona o código?

Neste programa, o segredo está na estrutura da função `loop` :

- **Espera sem bloqueios**: Enquanto ninguém pressionar o botão, o microcontrolador lê a linha `int digitalRead`, verifica que o resultado é `LOW`, ignora o bloco `if` e volta a começar o ciclo. Isto acontece milhares de vezes por segundo, fazendo com que o LED verde se mantenha aceso sem qualquer pausa.
- **O problema da insensibilidade temporal**: Assim que o peão pressiona o botão e entramos no bloco `if`, o fluxo sequencial toma o controlo e executam-se as instruções `delay` e `delay`. Durante esses 7 segundos totais, o Arduino fica "congelado" a executar as pausas. Se outro peão voltasse a pressionar o botão nesse intervalo, o Arduino não se aperceberia, uma vez que não consegue executar a linha `digitalRead` enquanto estiver adormecido por um `delay` .

O Desafio da Prática 7

Coloca à prova as tuas capacidades resolvendo estas duas melhorias lógicas de engenharia urbana:

1. **Tempo de Cortesia de Trânsito (Cool-Down)**: Numa cidade real, se um peão pressionar o botão, cruzar e, logo ao terminar, outro peão voltar a pressionar o botão, o trânsito ficaria caótico. Modifica o código para adicionar um tempo de espera obrigatório de **5 segundos** no final do ciclo (após acender novamente o verde) durante o qual o sistema ignore por completo qualquer pulsação do botão.
2. **Aviso acústico ou visual de fim de passagem**: Adiciona um piscar rápido no LED vermelho durante o último segundo da sua ativação para avisar de forma visual os peões de que o seu tempo de travessia está prestes a terminar.
3. *Pergunta para refletir*: Como resolverias o problema da insensibilidade do botão sem recorrer a hardware complexo? Se dividisses um atraso longo como `delay(5000)` em pequenos passos de `delay(100)`, de que forma isto poderia ajudar a verificar o estado do botão com maior frequência?