01

Lendo a resposta do jogador

Transcrição

No capítulo anterior aprendemos a piscar os LEDs de diversas formas. Neste capítulo, veremos como **ler** a sequência luminosa clicada pelo jogador, ou seja, veremos como entender qual foi a *resposta* dele. Para isso, nós utilizaremos os botões.

Lembra-se como procedemos com o LED? Nós primeiro fizemos um teste e apenas observamos para que depois pudéssemos reproduzir!

Nós vamos repetir essa estratégia, primeiro, analisaremos a lógica de um botão para depois expandir a conclusão para cada LED restante, ou seja, para os demais*botões*. Assim, enviamos um programa vazio para o **Arduino** e dessa maneira poderemos retirar o circuito sem preocupações. Fazendo isso nós podemos deixar **apenas** o circuito vermelho. Vamos no `Fritzing` , nele poderemos verificar como está o circuito.

Podemos realizar uma busca no `Fritzing` por `button` e selecionar o `push button` que é a versão de um botão simples com 4 pernas. Mas, não precisamos nos preocupar com ela, pois nós optaremos por utilizar uma versão que possui apenas 2 pernas, ou seja, uma de *entrada* e outra de *saída*.

Precisamos pensar em como comunicar para o **Arduino** que o botão foi pressionado. Primeiro, você precisa saber que o botão *corta* o circuito e quando pressionado ele se **fecha** permitindo, assim, a corrente passar.

Identificando o clique no botão

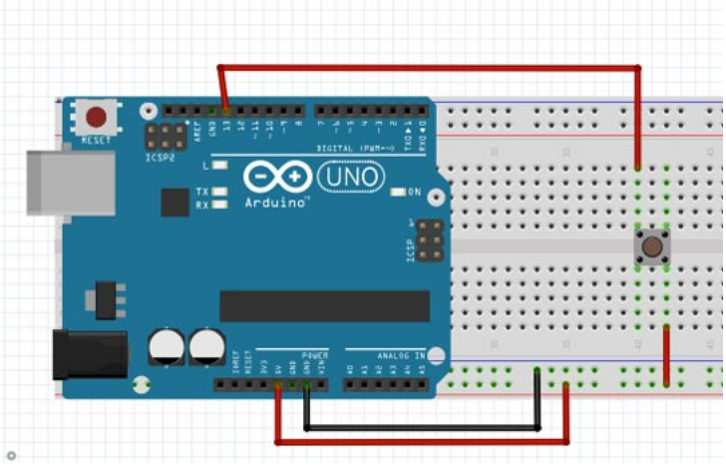
Como identificar que o botão foi pressionado? Podemos seguir o seguinte caminho:

Vamos conectar uma entrada de `5V` no botão;

No seu pino de saída, nós ligamos um fio até uma das portas do **Arduino**;

Caso não chegue nenhuma corrente na porta, quer dizer que o botão está parado, sem ninguém mexendo nele. Porém, quando alguém pressioná-lo, o circuito irá se fechar e uma corrente de `5V` chegará na porta escolhida, o que indicará que o botão foi pressionado.

E isso funciona! Observe como fica o circuito no `Fritzing` :



Agora, precisamos **ler** essa entrada `5V`. Para fazer isso vamos iniciar um novo arquivo para testar nossas ideias! Quando elas estiverem melhor estabelecidas nós podemos passar os códigos para o programa principal e, assim, evitamos que fique confuso. Neste circuito, temos o `LED_VERMELHO` conectado à porta **4** do arduino, e o `BOTAO_VERMELHO` conectado à porta **10**. Configurando as portas teremos:

```
#define LED_VERMELHO 4
#define BOTAO_VERMELHO 10


void setup(){
    pinMode(LED_VERMELHO, OUTPUT);


    pinMode(BOTAO_VERMELHO, INPUT);


    digitalWrite(LED_VERMELHO, HIGH);


    delay(1000);


    digitalWrite(LED_VERMELHO, LOW);
}
```

Podemos configurar o LED_VERMELHO normalmente e, ademais, adicionamos o `BOTAO_VERMELHO`, que diferentemente dos nossos LEDs, é uma porta de **entrada** de sinal. Piscaremos a luz vermelha apenas para ter a certeza de que o programa está rodando corretamente. Tendo feito isso, salve o novo arquivo e envie para o **Arduino**. Agora, tudo deverá ocorrer corretamente!

Nesse momento, vamos utilizar o `Serial` para descobrir se o botão foi pressionado ou não. Por isso, no `setup()`, adicionamos a linha que configura o `serial`: `Serial.begin(9600);`.

No `loop()` tentaremos **ler** a entrada do botão. A dica já foi dada, da mesma maneira que enviamos um sinal utilizando o `digitalWrite()` vamos **ler** o sinal aplicando a função `digitalRead()`. Ao inserir isso no `loop()` poderemos verificar constantemente qual o sinal que chega na porta do botão. No caso, o **Arduino** identifica os sinais através do **1** ou **0**; o **0** indica "nenhum" e o número **1** mostra quando o sinal é receptado. Entretanto, para inspecionar isso não basta colarmos o valor do sinal no Serial:

```
void loop(){
    int estadoBotao = digitalRead(BOTAO_VERMELHO);
    Serial.println(estadoBotao);
    delay(500);
}
```

Entretanto, compilando isso e enviando para o **Arduino** vemos que, infelizmente, não funciona! Ao inspecionarmos o `Serial` percebemos que nossa porta recebe constantemente o `0`, e mesmo sem apertarmos o botão, recebe o sinal `1`. Por que isso acontece?