

07

Mão na massa: Processando a resposta do jogador

Chegou o momento de verificar a resposta do usuário e deixar o nosso jogo completo! Já temos uma função para isso, mas, por enquanto, ela só incrementa a variável `leds_respondidos`. Procure no seu código a função `processaRespostaUsuario`:

```
void processaRespostaUsuario(){
    leds_respondidos++;
}
```

Vamos começar?!

1) Primeiro, é preciso alterar a função e incrementar a variável `leds_respondidos` apenas quando a resposta do jogador estiver correta. Lembrando que já temos uma outra função `checaRespostaJogador` que verifica a resposta do jogador. Chamamos essa função e adicionamos um `if` que testará se a resposta do usuário é equivalente a sequência de luzes proposta pelo jogo:

```
void processaRespostaUsuario(){
    int resposta = checaRespostaJogador();

    if(resposta == sequenciaLuzes[leds_respondidos]){
        leds_respondidos++;
    }else{
        //aqui terá um código ainda
        Serial.println("resposta errada");
    }
}
```

2) O problema ainda é que o jogador pode demorar para responder e nesse caso, a função `checaRespostaJogador` devolve `INDEFINIDO`. Então, é preciso verificar se recebemos o `INDEFINIDO`. Se isso acontecer, devemos sair da função já que não há resposta do jogador:

```
void processaRespostaUsuario(){
    int resposta = checaRespostaJogador();

    if(resposta == INDEFINIDO){ //novo if
        return; //saindo da função
    }

    if(resposta == sequenciaLuzes[leds_respondidos]){
        leds_respondidos++;
    }else{
        Serial.println("resposta errada");
    }
}
```

3) Falta retornar ao jogador um feedback sobre seus erros ou acertos em relação a sequência de luzes. Portanto, vamos aproveitar as funções `piscaSequencial` e `piscaSequencia2`. Renomeie as funções para:

```
piscaSequencia1 para jogoFinalizadoSucesso piscaSequencia2 para jogoFinalizadoFalha
```

- 4) Agora, procure a função `loop`. Nela, dentro do `case` do estado `JOGO_FINALIZADO_SUCESSO`, chame a função `jogoFinalizadoSucesso`. Repita o mesmo procedimento para o `case` do estado `JOGO_FINALIZADO_FALHA` chamando a função `jogoFinalizadoFalha`:

```
switch(estadoAtual()){
    case PRONTO_PARA_PROX_RODADA:
        Serial.println("pronto para próxima rodada");
        preparaNovaRodada();
        break;
    case USUARIO_RESPONDENDO:
        Serial.println("usuário respondendo");
        processaRespostaUsuario();
        break;
    case JOGO_FINALIZADO_SUCESSO:
        Serial.println("jogo finalizado sucesso");
        jogoFinalizadoSucesso(); //novo
        break;
    case JOGO_FINALIZADO_FALHA:
        Serial.println("jogo finalizado falha");
        jogoFinalizadoFalha(); //novo
        break;
}
delay(500);
}
```

- 5) Falta atender ainda o estado `JOGO_FINALIZADO_FALHA` na função `estadoAtual`. Procure a função `estadoAtual` e altere-a da seguinte maneira:

```
int estadoAtual(){
    if(rodada <= TAMANHO_SEQUENCIA){
        if(leds_respondidos == rodada){
            return PRONTO_PARA_PROX_RODADA;
        }else{
            return USUARIO_RESPONDENDO;
        }
    }else if(rodada == TAMANHO_SEQUENCIA + 1){ //novo if else
        return JOGO_FINALIZADO_SUCESSO;
    }else{ //novo else
        return JOGO_FINALIZADO_FALHA;
    }
}
```

- 6) Por último, é preciso modificar a função `processaRespostaUsuario`, pois estamos imprimindo uma mensagem somente quando o usuário responde errado. A versão final (!) da função `processaRespostaUsuario` fica da seguinte forma:

```
void processaRespostaUsuario(){
    int resposta = checaRespostaJogador();

    if(resposta == INDEFINIDO){
        return;
    }
```

```
}
```

```
if(resposta == sequenciaLuzes[leds_respondidos]){
    leds_respondidos++;
}else{
    rodada = TAMANHO_SEQUENCIA + 2; //novo
}
```

Compile, teste e divirta-se!