

ARDUINO UNO: CARTILHA DE PROJETOS

ESCOLA PADRÃO PARQUE DAS NAÇÕES



v0.0.3

SUMÁRIO

Apresentação.....	4
Lista de componentes (inventário).....	5
Conhecendo os componentes.....	5
Arduino UNO.....	5
Pinos: Analógicos e Digitais.....	6
Protoboard (mini).....	7
LED.....	8
Motores e rodas.....	9
• Motores com suporte.....	9
• Motores sem suporte.....	9
Módulo sensor de distância ultrassônico.....	10
Módulo sensor de obstáculo infravermelho.....	11
Módulo sensor de movimento (PIR).....	12
Potenciômetro.....	13
Resistores.....	13
Servo-motor.....	14
Ponte H.....	14
Programação com Arduino.....	16
Ferramentas.....	16
Arduino IDE.....	16
Plataformas online (simuladores).....	17
Projetos.....	18
1. Protótipo de Semáforo.....	18
Materiais.....	18
Montagem.....	18
Código.....	19
2. Girassol Solar.....	21
Materiais.....	21
Montagem.....	21
Código.....	22
3. Luminária automática.....	24
Materiais.....	24
Montagem.....	24
Código.....	25
4. Jogo da Batata Quente.....	26
Materiais.....	26
Montagem.....	27
Código.....	28

5. Jogo Genius.....	30
Materiais.....	30
Montagem.....	30
Código.....	31
6. Controle de Servo-Motor com Potenciômetro.....	35
Materiais.....	35
Montagem.....	35
Código.....	36
Agradecimentos.....	38

Apresentação

É com grande entusiasmo que compartilhamos com vocês o fruto de um projeto extensionista, oriundo da disciplina Mentoring II, na respeitada Pontifícia Universidade Católica de Minas Gerais, unidade Poços de Caldas. Como estudantes do curso de Ciência da Computação, decidimos aplicar nossos conhecimentos e paixão pela tecnologia para enriquecer o ambiente educacional em que nos encontramos.

Nossa jornada teve início com uma análise detalhada dos kits Arduino disponíveis na escola. Identificamos uma riqueza de recursos que poderiam ser explorados, percebendo o potencial para oferecer uma experiência prática e envolvente aos alunos.

Com o objetivo de estimular a aprendizagem prática, compilamos uma lista de projetos viáveis, levando em consideração os componentes disponíveis. Esses projetos não apenas abordam conceitos fundamentais de eletrônica e programação, mas também incentivam a criatividade e o pensamento crítico.

Desenvolvemos uma cartilha abrangente que serve como guia passo a passo para a montagem e execução dos projetos propostos. Nosso intuito é tornar a aprendizagem acessível, inclusiva e, acima de tudo, divertida. Acreditamos que a experiência prática com Arduino fortalece não apenas os fundamentos técnicos, mas também estimula o raciocínio lógico e o trabalho em equipe.

Convidamos todos vocês a explorarem conosco o vasto universo da tecnologia com Arduino. Cada LED aceso, cada código escrito é um passo em direção a um futuro mais brilhante e inovador.

Junte-se a nós nessa jornada emocionante, onde a educação encontra a inovação, e juntos construímos pontes para o futuro.

Lista de componentes (inventário)

- **Arduino UNO:** 34 unidades
- **Interruptor:** 43 unidades
- **Conector bateria 9V:** 35 unidades
- **Jumpers:** diversos
- **LDR (Sensor fotoresistor):** 7 unidades
- **Módulo sensor de distância ultrassônico:** 7 unidades
- **Módulo sensor de movimento PIR:** 7 unidades
- **Módulo sensor de obstáculo infravermelho:** 7 unidades
- **Módulo sensor de som:** 7 unidades
- **Motor (com suporte para roda):** 53 unidades
- **Motor (sem suporte):** 42 unidades
- **Mini-protoboards:** 71 unidades
- **Ponte H:** 35 unidades
- **Potenciômetro:** 41 unidades
- **Resistor:** diversos
- **Rodas:** 74 unidades
- **Servo-motor:** 28 unidades
- **Suporte para pilha AA:** 42 unidades
- **LEDs:**
 - **Azul:** 40 unidades
 - **Vermelho:** 41 unidades
 - **Verde:** 40 unidades
 - **Amarelo:** 40 unidades
 - **Branco:** 90 unidades
 - **Não-categorizado:** 20 unidades

Conhecendo os componentes

Arduino UNO

Um Arduino é como o cérebro de um projeto eletrônico que você pode criar. Imagine que você quer fazer um brinquedo que se move, acende luzes ou faz sons especiais. O Arduino é como o comandante desse brinquedo.

Pense no Arduino como um controlador mágico que você programa para dizer ao seu projeto o que fazer. Ele tem entradas, como olhos e ouvidos, chamadas de "pinos", onde ele pode receber informações. E tem saídas, como braços e pernas, chamadas de "pinos de saída", onde ele pode enviar comandos para fazer coisas acontecerem.

Quando você escreve um código (um conjunto de instruções) para o Arduino, é como dar um conjunto de regras para o seu projeto seguir. Por exemplo, "se alguém se aproximar, acenda as luzes". O Arduino executa essas regras e faz o seu projeto funcionar do jeito que você quer.

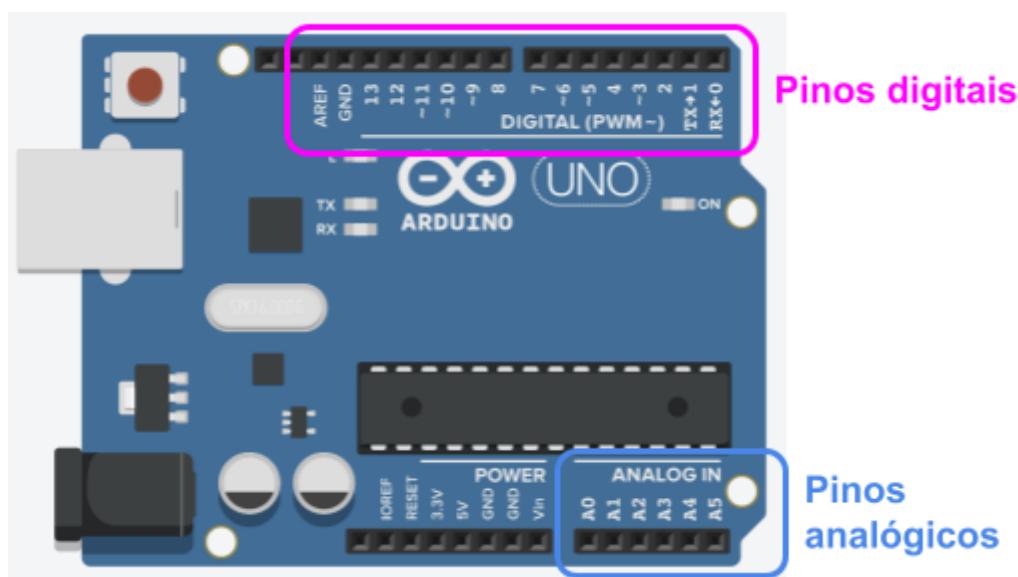
Então, resumindo, o Arduino é como o cérebro inteligente que você programa para dar vida aos seus próprios brinquedos e projetos legais!

Pinos: Analógicos e Digitais

O Arduino é como o cérebro de um projeto eletrônico que você pode criar. Ele tem pinos, que são como pontos de conexão, e esses pinos podem ser de dois tipos principais: digitais e analógicos. **Os pinos digitais** são como interruptores que podem estar ligados (HIGH) ou desligados (LOW). Eles são ótimos para ligar ou desligar componentes, como LEDs. Por exemplo, você pode conectar um LED ao pino digital e, ao programar o Arduino, fazer o LED piscar.

Os **pinos analógicos**, por outro lado, são como medidores que podem ter diferentes valores **entre 0 e 1023**. Eles são úteis para medir coisas como luz, som ou temperatura. Por exemplo, você pode conectar um sensor de luz ao pino analógico, e o Arduino lerá valores que indicam quanta luz está atingindo o sensor. Isso pode ser usado para criar um sistema de controle de luminosidade em sua luminária automática.

Ao entender como usar esses pinos, você pode controlar e interagir com seus componentes de maneiras incríveis. Os pinos digitais são como interruptores ligados/desligados, enquanto os pinos analógicos são como medidores que podem captar uma ampla gama de valores, permitindo uma variedade de aplicações em seus projetos criativos.

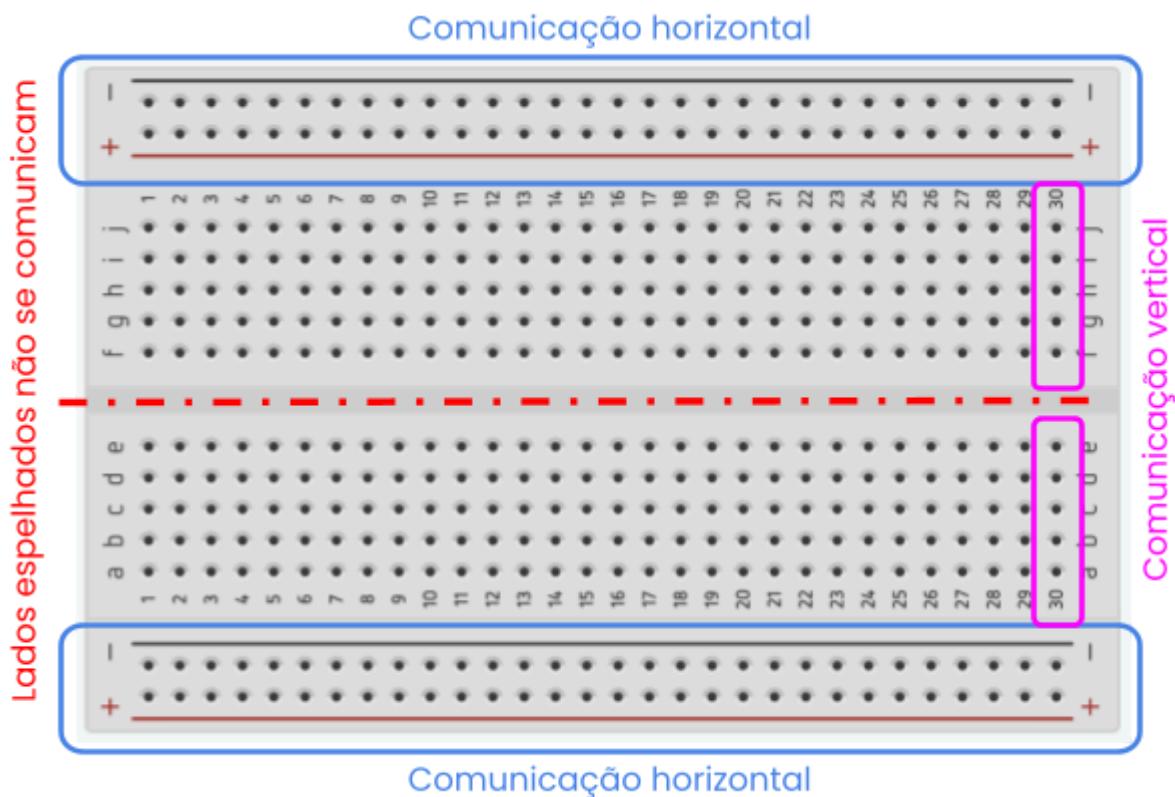


Pinos digitais

Pinos
analógicos

Protoboard (mini)

As mini-protoboards são como tabuleiros de jogo para suas invenções. Elas têm buracos onde você pode encaixar fios e componentes. Cada linha horizontal de buracos está conectada entre si, e o mesmo acontece com as colunas verticais. Isso facilita a conexão de componentes elétricos sem a necessidade de solda. Se você deseja conectar dois componentes, basta colocar os fios nos buracos da mesma linha ou coluna.



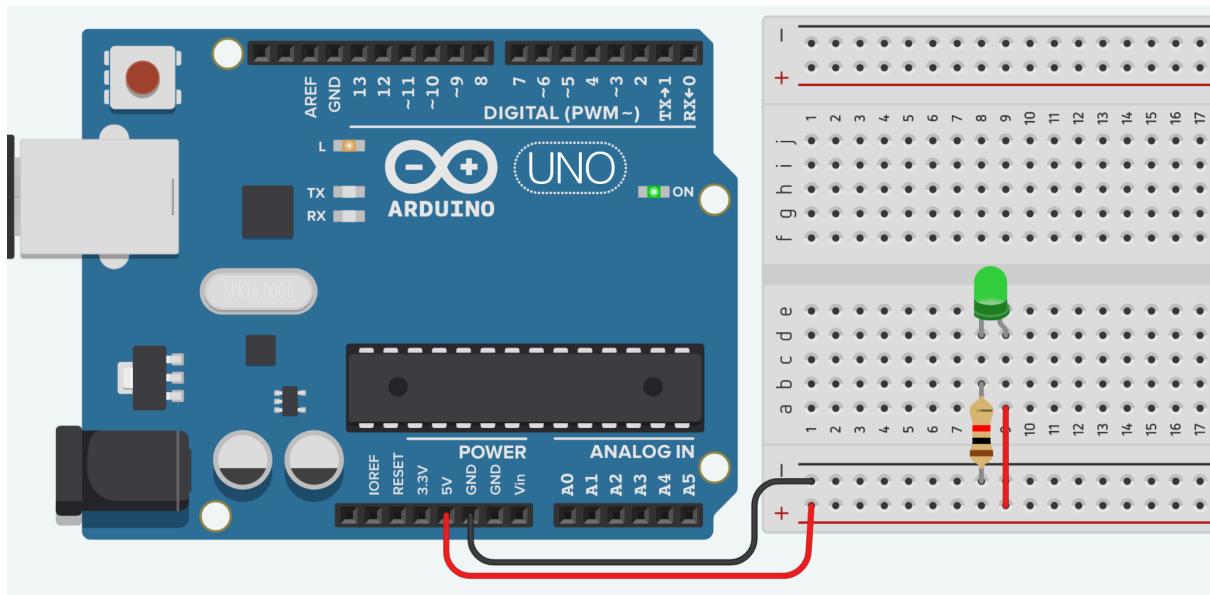
No entanto, é importante notar que não há conexão automática entre as linhas horizontais e as colunas verticais. Isso significa que se você deseja conectar um fio de uma linha a um da coluna, precisa usar um fio para fazer essa conexão.

Quanto ao espelhamento, em algumas protoboards, as linhas são espelhadas em lados opostos. Isso significa que a linha superior do lado direito está conectada com a linha inferior do lado esquerdo e vice-versa. No entanto, as colunas geralmente não são espelhadas. Portanto, se você deseja conectar componentes entre os lados espelhados, precisará usar um fio para fazer essa

conexão. Certifique-se de entender como a protoboard específica que você está usando funciona para garantir conexões precisas em seus projetos.

LED

Os LEDs são como pequenas lanternas que brilham quando conectadas à energia. Eles possuem um lado longo (+) e um lado curto (-). Para conectá-los ao Arduino, certifique-se de que o lado longo esteja no fio positivo de 3V (+) e o lado curto no fio negativo (-). Isso fará com que a luz acenda. Experimente diferentes cores e crie padrões de luzes divertidos!



Observação: Ao ligar um LED em um pino de 5V ou a uma saída digital do Arduino, utilize sempre um resistor junto ao conjunto. Este resistor pode ser ligado junto à perninha menor ou maior do LED, conectado ao GND ou VCC do sistema. Utilize a tabela ao lado como guia para escolher o melhor resistor para o sistema do seu projeto:

Resistor (OHMS) por cor de LED a depender da tensão (3v, 5v ou 9v)							
Tensão	amarelo	laranja	vermelho	verde	azul	roxo	branco
3v	33 Ω	33 Ω	33 Ω	-	-	-	-
5V	150 Ω	150 Ω	150 Ω	82 Ω	82 Ω	82 Ω	82 Ω
9V	330 Ω	330 Ω	330 Ω	270 Ω	270 Ω	270 Ω	270 Ω

Motores e rodas

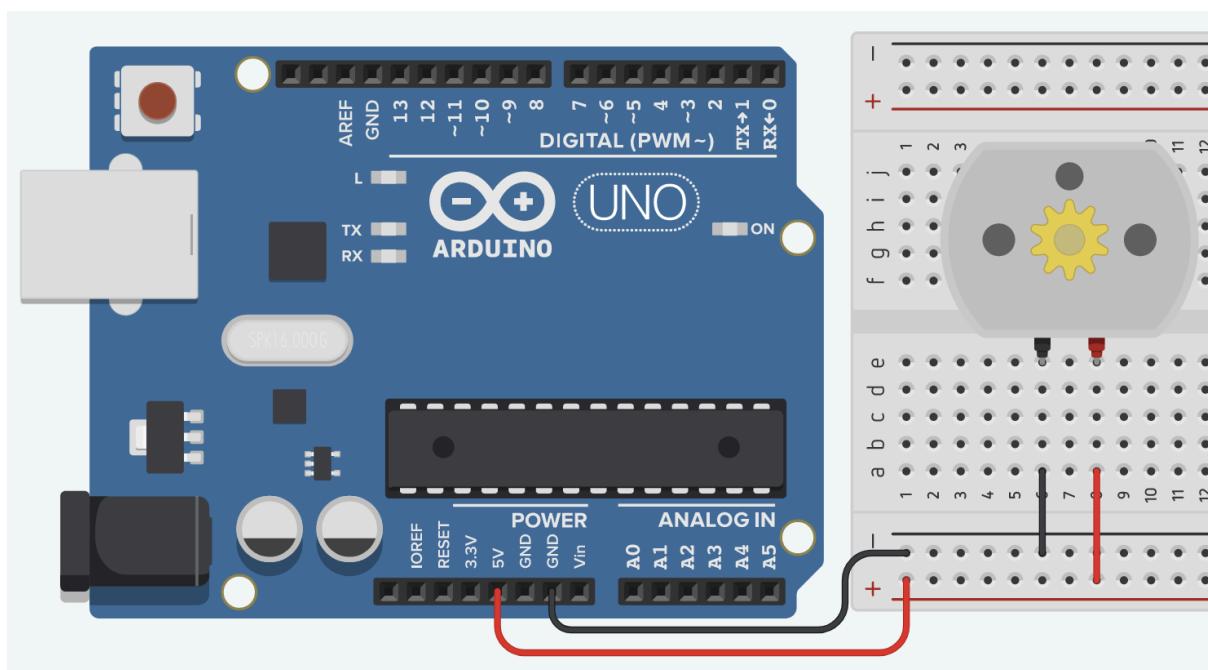
Motores são como os músculos dos seus projetos, dando movimento às coisas. Eles têm fios, e para conectá-los ao Arduino, use um fio para o pino positivo (+) e outro para o pino negativo (-). Rodas ajudam a distribuir esse movimento. Certifique-se de fixar as rodas corretamente para ter movimentos suaves.

• Motores com suporte

Este motor é especial porque já vem com um suporte para uma roda acoplada. Para conectá-lo ao Arduino, use fios: um para o pino positivo do motor ao pino de saída do Arduino e outro para o pino negativo do motor ao pino de terra do Arduino. Certifique-se de fixar a roda corretamente para garantir um movimento suave.

• Motores sem suporte

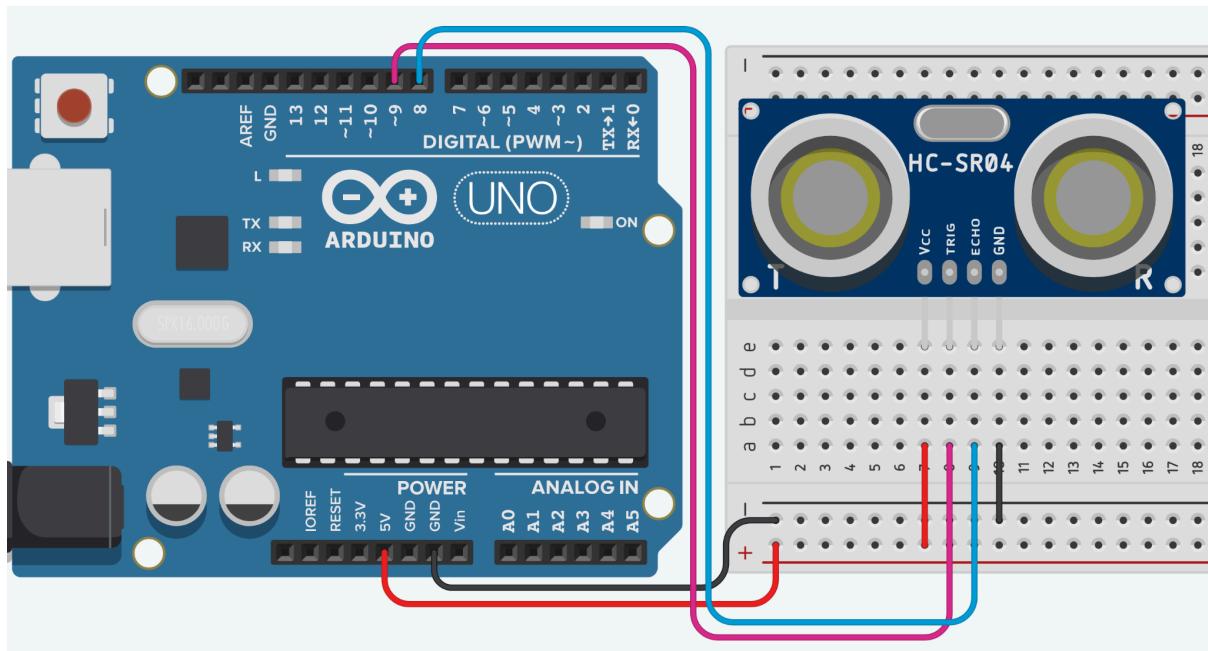
Similar ao motor com suporte, mas aqui você tem a liberdade de conectar diferentes objetos ou rodas conforme necessário. Para conectar ao Arduino, use fios: um para o pino positivo do motor ao pino de saída do Arduino e outro para o pino negativo do motor ao pino de terra do Arduino. Verifique a polaridade ao conectar os fios para garantir que o motor funcione corretamente (inverter a polaridade irá inverter também o sentido do giro do motor).



Cada componente tem suas próprias regras e truques, e ao entender como eles funcionam e se conectam ao Arduino, você pode criar projetos incríveis!

Módulo sensor de distância ultrassônico

Este sensor usa ondas sonoras para medir quanto longe algo está. Conecte-o ao Arduino: use um fio para o pino 5V ou VCC do Arduino para energizar o módulo, um para o trigger do sensor ao pino de entrada digital do Arduino, outro pino digital para o pino de eco e um terceiro para o pino de terra (GND). Posicione-o corretamente para obter medidas precisas.



O módulo de distância funciona enviando um sinal ultrassônico (um som muito alto que os humanos não podem ouvir) e medindo quanto tempo leva para esse som voltar depois de bater em um objeto. Para obtermos a distância deste objeto, precisamos fazer algumas contas. Veja como fazemos com um exemplo:

1. Enviar o Sinal (Pulso de Trigger):

- O Arduino envia um sinal chamado "pulso de trigger" para o módulo ultrassônico.
- Este pulso de trigger faz com que o módulo ultrassônico emita um som ultrassônico.

2. Receber o Sinal de Volta (Pulso de Echo):

- Esse som ultrassônico se move até atingir um objeto e rebater de volta para o módulo.

- O módulo ultrassônico detecta esse som de volta e gera um sinal chamado "pulso de echo".

3. Medir a Duração do Pulso de Echo:

- O Arduino mede quanto tempo o pulso de echo dura usando a função `pulseIn()`.
- Esse tempo é essencial porque quanto mais longo o tempo, mais distante está o objeto.

4. Calcular a Distância:

- Usando a fórmula:

$$\text{distância} = (\text{duração do pulso} * \text{velocidade do som}) / 2$$
- A velocidade do som é uma constante que diz o quanto rápido o som viaja. Em temperatura ambiente, essa velocidade é aproximadamente **343 metros por segundo**.

Vamos simplificar ainda mais com um exemplo: suponha que o tempo que o som levou para ir e voltar seja **1000 microsssegundos** (0,001 segundos). Então, usando a fórmula:

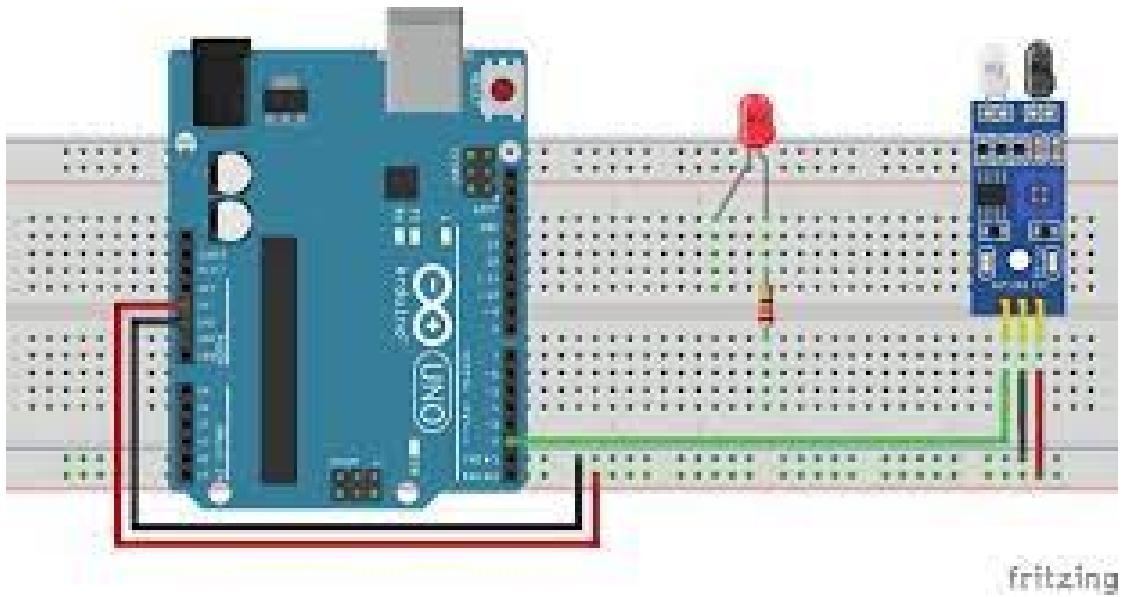
$$\text{distância} = (0,001 \text{ segundos} * 343 \text{ metros por segundo}) / 2$$

A distância seria aproximadamente 0,1715 metros, ou seja, **17,15 centímetros**.

Portanto, medir o tempo que o som leva para ir e voltar nos ajuda a calcular a distância até o objeto usando essa fórmula simples!

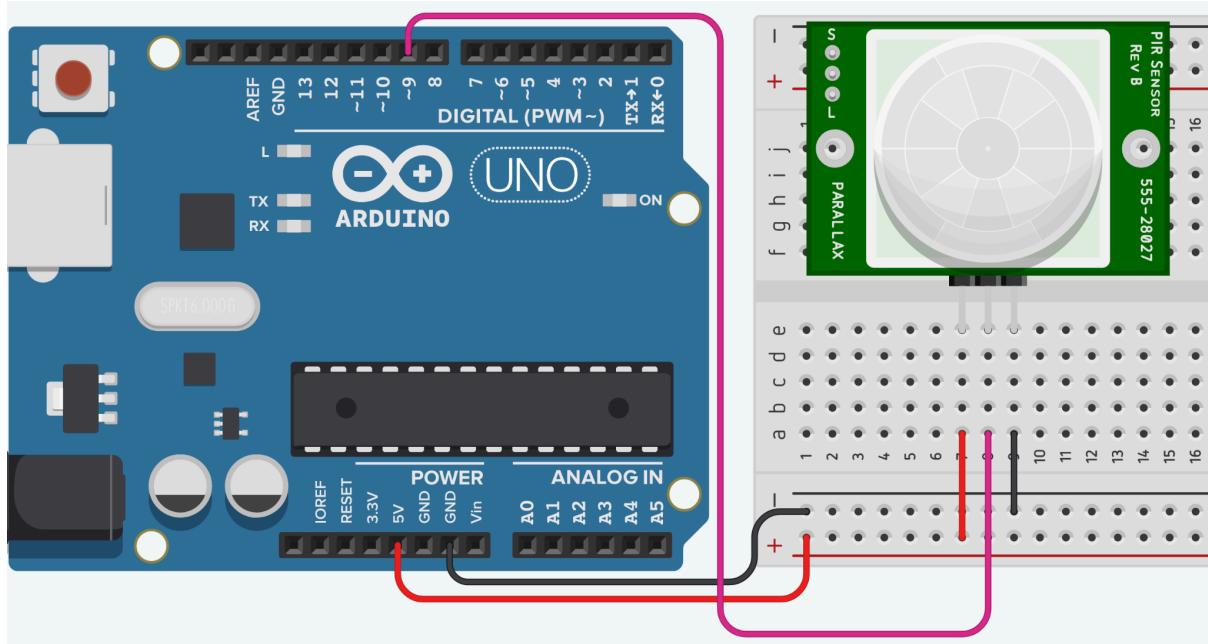
Módulo sensor de obstáculo infravermelho

Este sensor detecta objetos próximos usando luz infravermelha. Conecte-o ao Arduino usando fios: um para o pino de saída do sensor ao pino de entrada digital do Arduino, outro para o pino de energia e um terceiro para o pino de terra. Certifique-se de que não haja obstruções entre o sensor e o objeto para obter leituras precisas.



Módulo sensor de movimento (PIR)

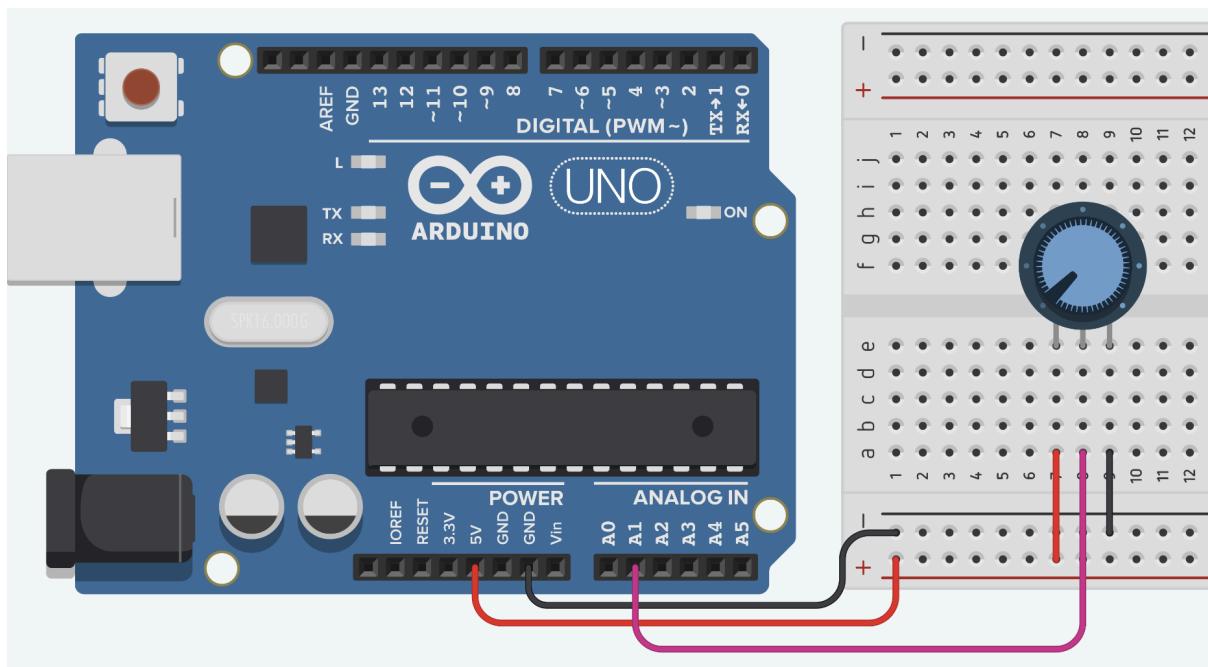
Este sensor possui a capacidade de poder sentir algum movimento. O módulo PIR geralmente possui três pinos: VCC (alimentação), OUT (saída) e GND (terra). Conecte o VCC ao 5V do Arduino, o GND ao GND do Arduino e o OUT a algum pino digital do Arduino, como o pino 9, por exemplo.



Quando o sensor detecta algum movimento, ele envia um sinal de nível alto (HIGH) ao Arduino. Desta maneira, podemos programá-lo para executar alguma ação ao detectar movimento, como acender uma lâmpada, por exemplo.

Potenciômetro

Imagine um controle deslizante ou botão giratório que você usa para ajustar o volume em um rádio. Esse controle é como um potenciômetro. Ele ajuda a controlar coisas, mas em vez de mexer no som, podemos usá-lo para controlar várias coisas em nossos projetos.



Quando giramos o controle (potenciômetro), ele muda o quanto de "eletricidade" está passando pela trilha. O Arduino pode sentir, através de um pino analógico, essas mudanças e nos contar, como se ele estivesse dizendo "Ei, a trilha ficou mais cheia de eletricidade agora!".

Resistores

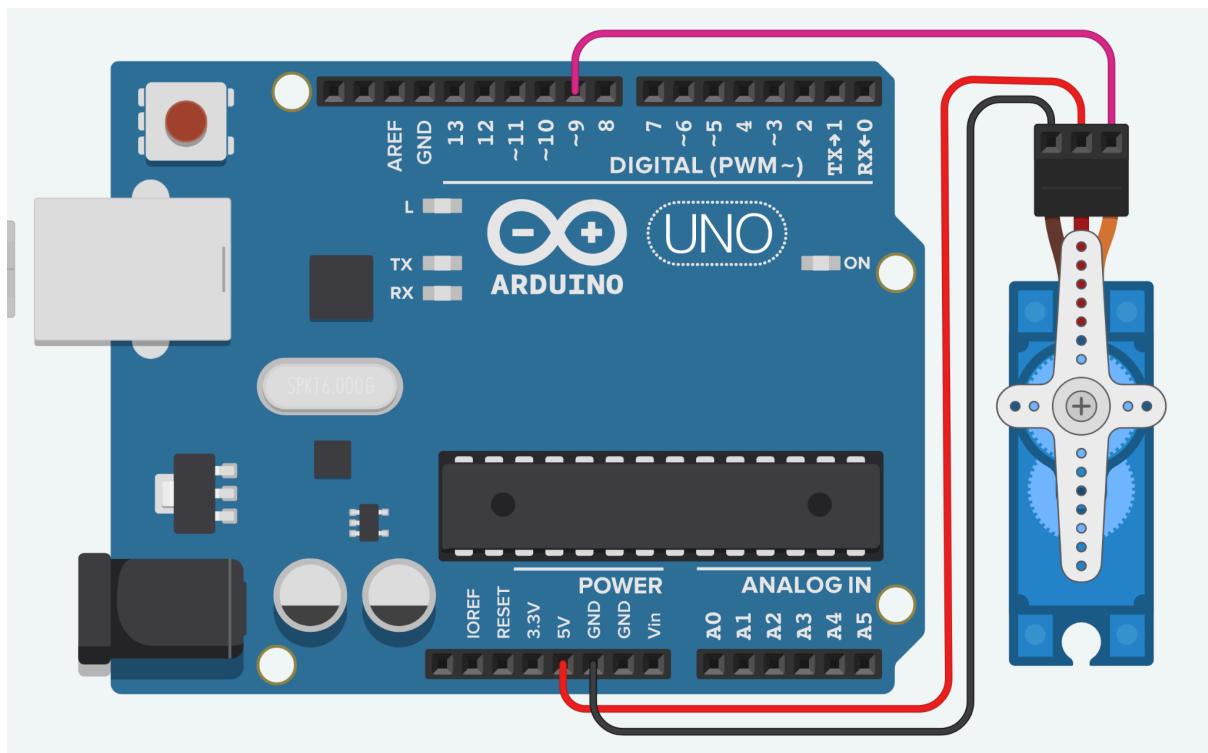
Vamos imaginar que você está em uma estrada e quer diminuir a velocidade do seu carro. O freio no carro é como um resistor. Os resistores são componentes eletrônicos que limitam a quantidade de corrente elétrica que passa por eles. Eles são como "obstáculos" que ajudam a controlar o fluxo de eletricidade em um circuito. Em projetos com Arduino, os resistores são usados por várias razões. Uma das razões mais comuns é limitar a corrente que passa por LEDs, protegendo-os de receberem muita eletricidade.



e queimarem. Eles também são usados em divisores de tensão, pull-up e pull-down em circuitos, ajudando a garantir que os pinos do Arduino recebam sinais elétricos adequados.

Servo-motor

Pense em um servo-motor como um braço robótico que você pode controlar com precisão. Ele é projetado para manter uma posição específica com base em sinais elétricos. O servo-motor possui um mecanismo interno que ajusta automaticamente a posição do eixo para corresponder à posição desejada.

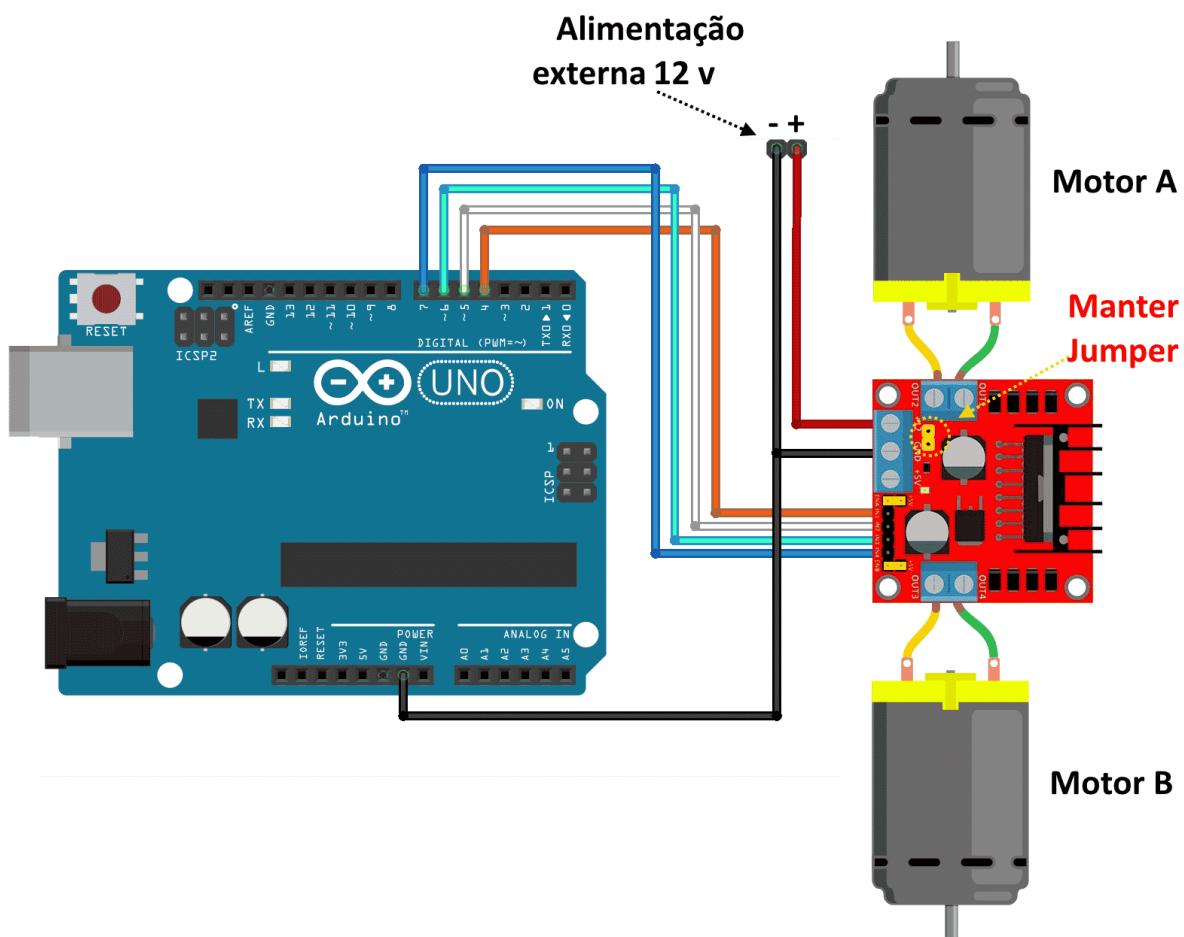


Um servo-motor geralmente tem três fios: alimentação (VCC), terra (GND) e controle (sinal). Conectamos o fio de alimentação ao 5V do Arduino, o fio de terra ao GND do Arduino e o fio de controle a um pino digital específico no Arduino, como o pino 9, por exemplo. Quando enviamos um sinal ao pino de controle do servo-motor, ele move o braço. O Arduino pode enviar comandos ao servo-motor para girar para a esquerda, para a direita ou parar em uma posição específica.

Ponte H

Imagine uma ponte em que você pode decidir se quer ou não permitir que os carros sigam em frente. Uma ponte H é um componente eletrônico que faz algo semelhante, mas para corrente elétrica. Ela permite controlar a direção (sentido) e a velocidade de um motor.

Ao enviar sinais elétricos aos pinos de controle, podemos dizer à ponte H para permitir que a corrente vá para frente, para trás ou parada. Isso é útil para controlar a direção e a velocidade de um motor.



A ponte H tem vários pinos, mas os principais são para os dois motores e para o controle. Geralmente, temos pinos para motor A (M1 e M2), motor B (M3 e M4), além de pinos de controle, como Enable1, Enable2, Input1, Input2, Input3 e Input4. Conectamos os pinos de controle (Enable1, Enable2) a pinos PWM (sinalizados com um "~") do Arduino (como ~5 e ~6). Conectamos os pinos de Input e Output aos pinos digitais do Arduino (como 4 e 7).

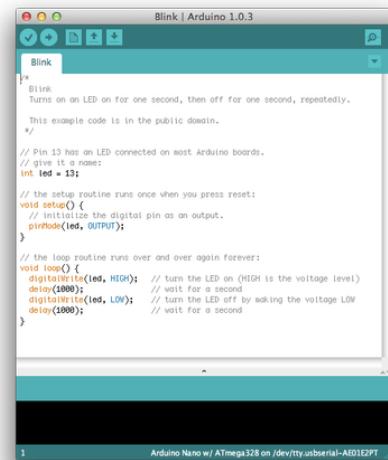
Resumindo, uma ponte H é como um controlador de tráfego para motores. Conectamos os pinos da ponte H aos pinos do Arduino e usamos código para controlar a direção e a velocidade do motor. É como ter um semáforo para motores no seu projeto!

Programação com Arduino

Ferramentas

Arduino IDE

Imagine que você está contando uma história para um robô e quer que ele siga suas instruções. O Arduino IDE (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado, em português) é como a linguagem que você usa para contar essa história ao seu Arduino. Ele é um software que facilita a criação, edição e upload de código para o seu Arduino.



Como Usamos o Arduino IDE em Nossos Projetos:

1. Como Instalar:

- Você pode baixar o Arduino IDE gratuitamente no site oficial:
<https://www.arduino.cc/en/software>
- Ele está disponível para várias plataformas, como Windows, Mac e Linux. Escolha a versão adequada ao seu sistema operacional.

2. Escrever o Código

- No Arduino IDE, você escreve o código usando a linguagem de programação C/C++. Pode parecer complicado no início, mas você verá que é como contar uma história ao Arduino, dizendo o que você quer que ele faça.

3. Conectar o Arduino ao Computador:

- Antes de enviar as instruções ao Arduino, você precisa conectar o Arduino ao computador usando um cabo USB. O Arduino IDE usará essa conexão para enviar o código que você escreveu para o Arduino.

4. Selecionar a Placa e a Porta:

- Para identificar a porta que seu Arduino está usando, certifique-se que o Arduino está conectado à uma porta USB do seu computador.
- No canto superior direito do Arduino IDE, clique em "Ferramentas" (*Tools*).
- Escolha o tipo de placa Arduino que você está usando. Aqui, usaremos o "Arduino Uno".
- Ao lado da opção da placa, haverá uma lista suspensa para a "Porta". As portas disponíveis serão listadas aí.
- A porta do Arduino geralmente é identificada pelo nome do dispositivo e um número da porta. Por exemplo, em sistemas Windows, pode ser algo como "COM3". Em sistemas MacOS ou Linux, pode ser "/dev/ttyUSB0" ou "/dev/cu.usbmodemxxxx".

5. Enviar o Código (Fazer o Upload):

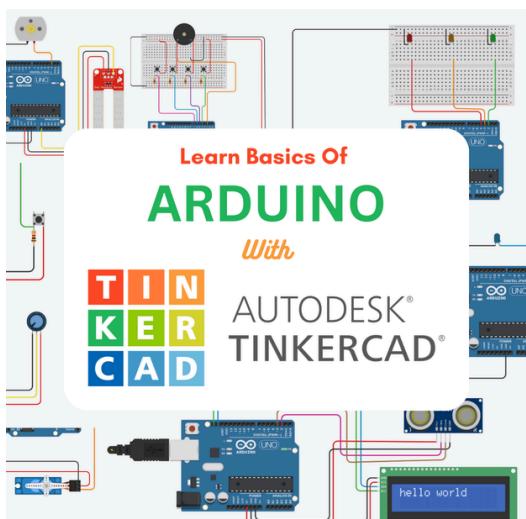
- Depois de escrever o código, você pressiona o botão "Upload". Isso faz com que o Arduino IDE envie o código para o Arduino. É como dar a história escrita para o robô seguir.

6. Observar a Execução no Monitor Serial:

- O Arduino IDE tem uma ferramenta chamada Monitor Serial, onde você pode ver mensagens ou dados que o Arduino envia de volta para o computador. Isso é útil para entender o que está acontecendo no seu projeto.

Plataformas online (simuladores)

Imagine uma ferramenta que permite criar e experimentar com eletrônica e programação sem precisar de fios, componentes físicos ou mesmo um Arduino



real. O Tinkercad é exatamente isso - uma plataforma online que permite criar, simular e experimentar com projetos de eletrônica e programação de maneira virtual.

Você pode acessar o Tinkercad em <https://www.tinkercad.com/>. Se você ainda não tem uma conta, pode criar uma gratuitamente. Usaremos o Tinkercad para criar as ilustrações e simular os projetos apresentados nesta cartilha.

Projetos

1. Protótipo de Semáforo

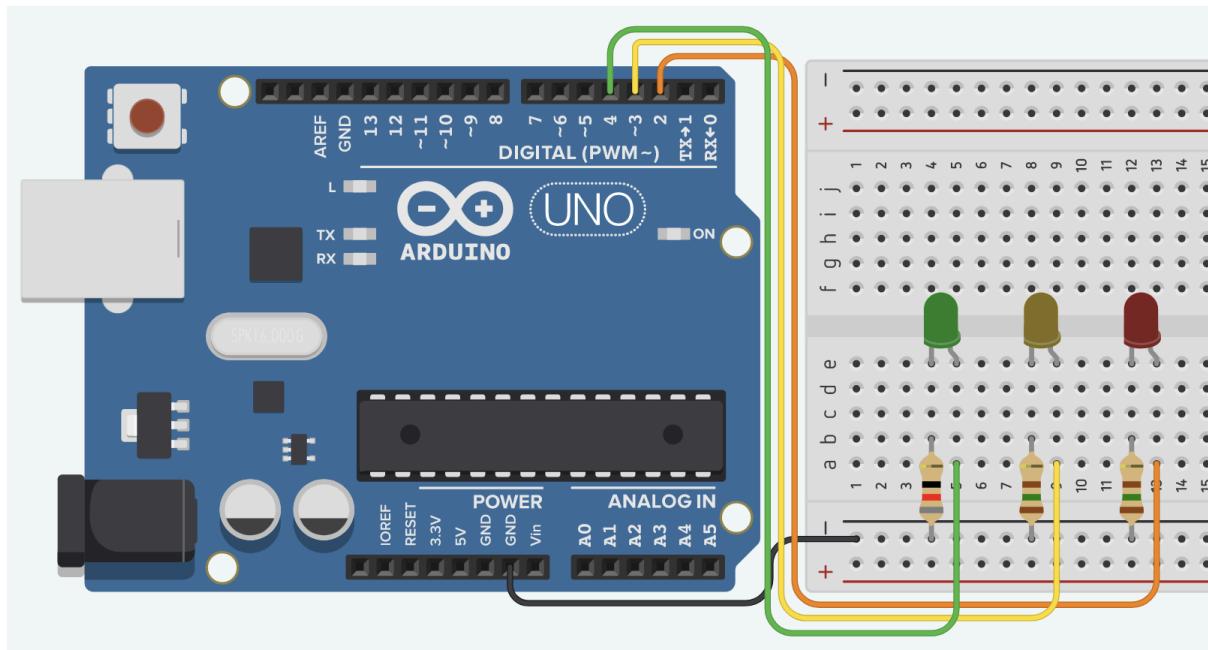
Crie um protótipo de semáforo de trânsito que simula a mudança de sinal usando LEDs.

Materiais

- 3 leds (verde, amarelo e vermelho)
- 1 protoboard
- 3 resistores
- Jumpers diversos

Montagem

- Conecte o **LED Vermelho** ao pino digital 2 do Arduino.
- Conecte o **LED Amarelo** ao pino digital 3 do Arduino.
- Conecte o **LED Verde** ao pino digital 4 do Arduino.
- Conecte um resistor $150\ \Omega$ no LED Vermelho e Amarelo.
- Conecte um resistor $82\ \Omega$ no LED Verde.
- Conecte um jumper da trilha negativa (-) da protoboard ao GND do Arduino.



Código

```
// Definindo os pinos para os LEDs
const int pinVermelho = 2; // Pino digital 2 para o LED vermelho
const int pinAmarelo = 3; // Pino digital 3 para o LED amarelo
const int pinVerde = 4; // Pino digital 4 para o LED verde

void setup() {
    // Configura os pinos dos LEDs como saídas
    pinMode(pinVermelho, OUTPUT);
    pinMode(pinAmarelo, OUTPUT);
    pinMode(pinVerde, OUTPUT);
}

void loop() {
    // Ciclo de Semáforo
    // Sinal Verde por 30 segundos
    digitalWrite(pinVerde, HIGH);
    delay(30000); // 30 segundos

    // Sinal Amarelo por 3 segundos
    digitalWrite(pinVerde, LOW);
    digitalWrite(pinAmarelo, HIGH);
    delay(3000); // 3 segundos

    // Sinal Vermelho por 30 segundos
    digitalWrite(pinAmarelo, LOW);
    digitalWrite(pinVermelho, HIGH);
    delay(30000); // 30 segundos

    // Reinicia o ciclo
    digitalWrite(pinVermelho, LOW);
}
```

Explicando o código

1. Definindo os Pinos:

- Identificamos os pinos digitais 2, 3 e 4 do Arduino conectados aos LEDs vermelho, amarelo e verde, respectivamente.

2. Configurando os LEDs:

- Utilizamos o bloco void `setup()` para configurar os pinos que iremos utilizar em nosso projeto, dizendo ao Arduino quais serão usadas para entrada e saída (input/output) de dados.
- Ao utilizar a função `pinMode`, dizemos ao Arduino qual modo iremos trabalhar com aquele respectivo pino. Aqui os pinos serão usados para enviar ("output") sinais aos LEDs (como ligar e desligar).
- Usamos a função `digitalWrite(pino, HIGH)` para definir o estado do LED enviando um sinal, ao enviar sinal HIGH ligamos o LED e, ao enviar um sinal LOW, desligamos o LED.

3. Ciclo de Semáforo:

- A função `delay` é usada quando queremos gerar um atraso até que o próximo comando seja executado, informando em milissegundos o tempo que deve ser esperado para dar continuidade ao código.
- Começa o ciclo:
 - Luz Verde: Acender o LED verde e esperar 30 segundos.
 - Luz Amarela: Apagar o LED verde e acender o amarelo por 3 segundos.
 - Luz Vermelha: Apagar o LED amarelo e acender o vermelho por 30 segundos.
- Repete esse ciclo continuamente. Todo o código escrito no bloco `void loop() {}` será executado repetidamente.

Esse código simula um semáforo, seguindo o mesmo padrão que vemos nas ruas. Ele controla quando as luzes ficam verdes, amarelas e vermelhas, dando um tempo específico para cada uma delas. Essa é uma maneira simples de começar a entender como o Arduino pode controlar luzes de forma programada.

Como desafio extra, pode-se adicionar um botão ao projeto para simular um sinal para informar que um pedestre quer atravessar a rua. Além disso, um buzzer pode ser adicionado também, para enviar um som de alerta para deficientes visuais.

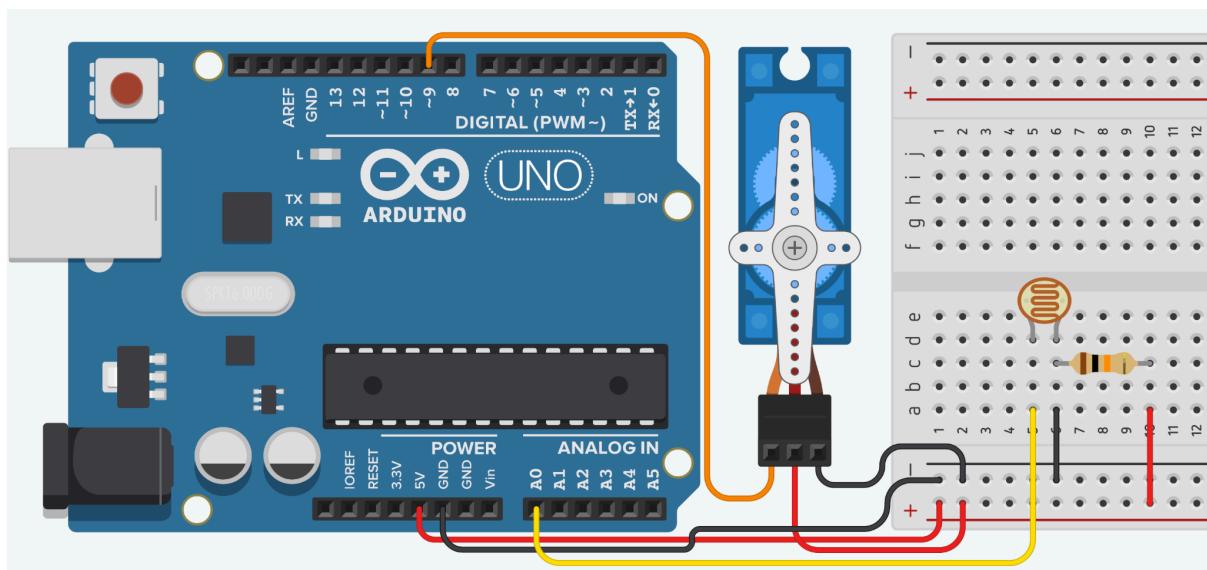
2. Girassol Solar

Crie um girassol artificial que se move em direção à luz usando o sensor LDR.

Materiais

1. Arduino Uno
2. Protoboard
3. Servo-motor
4. Sensor de Luz (LDR - Light Dependent Resistor)
5. Cabos Jumper (fios de conexão)
6. Suporte para o girassol (pode ser feito com papel colorido)

Montagem



1. Conectando o servo-motor:
 - a. Conecte o fio de controle do motor de servo ao pino digital 9 do Arduino.
 - b. Conecte o fio de alimentação do motor de servo ao pino 5V do Arduino.
 - c. Conecte o fio de terra (GND) do motor de servo ao pino GND do Arduino.
2. Conectando o sensor de Luz:
 - a. Conecte um terminal do LDR ao pino analógico A0 do Arduino.
 - b. Conecte o outro terminal do LDR a uma resistência de 10k ohms.
 - c. Conecte a outra extremidade da resistência ao pino 5V do Arduino.

- d. Conecte a extremidade central da resistência (conectada ao LDR) ao pino GND do Arduino.
3. Montagem do suporte do Girassol
- a. Fixe o sensor de luz (LDR) em um lado do suporte, representando a "cabeça" do girassol.
 - b. Fixe o servo-motor na base do suporte, representando o "caule" do girassol.
 - c. Use papel colorido para criar as "pétais" ao redor do sensor de luz.

Código

```
#include <Servo.h> // Inclui a biblioteca Servo necessária para controlar o motor de servo.

// Definindo os pinos para o servo e o sensor de luz
const int pinoServo = 9; // Pino digital para o controle do motor de servo
const int pinoLDR = A0; // Pino analógico para o sensor de luz (LDR)

Servo motorServo; // Criando um objeto Servo para controlar o motor de servo

void setup(){
    motorServo.attach(pinoServo); // Inicializa o motor de servo
    Serial.begin(9600); // Inicializa a comunicação serial (para fins de depuração)
}

void loop(){
    int intensidadeLuz = analogRead(pinoLDR); // Lê a intensidade de luz do sensor de luz

    // Mapeia a leitura do sensor para a faixa de movimento do servo
    int angulo = map(intensidadeLuz, 0, 1023, 0, 180);

    motorServo.write(angulo); // Move o servo para o ângulo calculado

    // Exibe a leitura do sensor no Monitor Serial (opcional)
    Serial.print("Intensidade de Luz: ");
    Serial.println(intensidadeLuz);

    // Aguarda um curto intervalo antes da próxima leitura
    delay(100);
}
```

Explicando o código

1. `#include <Servo.h>`: Aqui estamos chamando um amigo especial para nos ajudar a mexer o girassol - o `Servo.h` é uma biblioteca que permite ao servo-motor mexer conforme a nossa vontade.
2. **Definição de Pinos:**
 - `const int pinoServo = 9;`: Este é nosso servo-motor. Ele tem um fio conectado ao pino digital 9 do Arduino.
 - `const int pinoLDR = A0;`: Este é o "olhinho" que consegue "ver" a luz, chamado LDR. Um de seus fios está ligado no pino analógico A0.
3. **Objeto Servo:**
 - `Servo motorServo;`: Aqui estamos criando um "personagem" chamado `motorServo`. Ele será responsável por nos ajudar a girar a cabeça do nosso girassol.
4. **Setup:**
 - `motorServo.attach(pinoServo);`: Preparamos nosso amigo servo-motor para a ação. Ele se conecta ao pino 9 (`pinoServo`) para saber quando deve ajudar o girassol a olhar para a luz.
 - `Serial.begin(9600);`: Isso é como um diário secreto do Girassol que escreve o que ele sente. Isso é opcional, mas é legal para sabermos o que se passa no código ao acompanhar pelo monitor.
5. **Loop:**
 - `analogRead(pinoLDR);`: O Girassol usa seu "olhinho" (LDR) para "ver" ou "sentir" a luz ao redor, naquele exato momento.
 - `map(intensidadeLuz, 0, 1023, 0, 180);`: Transformamos o que o Girassol sente em algo que ele entende, um valor entre 0 e 180.
 - `motorServo.write(ângulo);`: O servo-motor ajuda o Girassol a "olhar" para a luz, movendo sua cabeça na direção certa.
 - `Serial.print e Serial.println`: Escrevemos no diário secreto do Girassol o que ele sentiu. Podemos ler este diário no serial monitor.
 - `delay(100);`: O Girassol tira uma pausa rápida antes de abrir os "olhos" novamente para "sentir" a luz.

Este código faz o Girassol se mover em direção à luz, como se estivesse seguindo o sol. O LDR funciona como o "olho" e o servo-motor ajuda a mover o Girassol na direção correta.

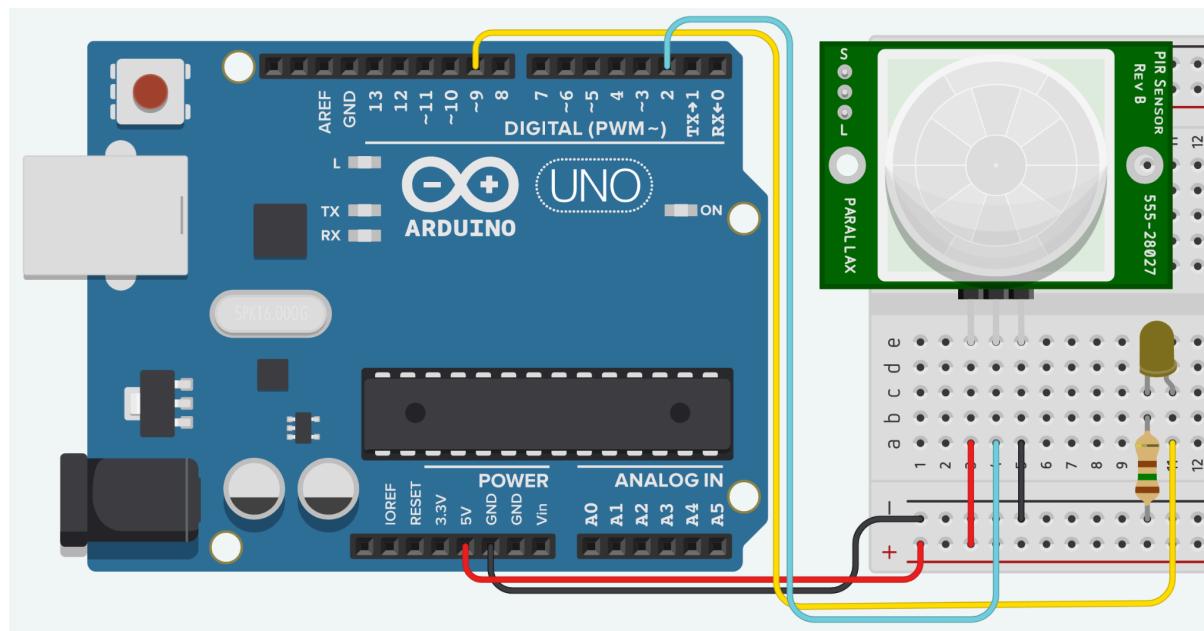
3. Luminaria automatica

Crie uma luminária inteligente que se acende automaticamente em ambientes quando detecta movimento, proporcionando praticidade e economia de energia.

Materiais

1. Arduino Uno
2. Protoboard
3. Sensor de Movimento PIR (Passive Infrared)
4. LED (qualquer cor)
5. Resistor (limita a corrente do LED)
6. Cabos Jumper (fios de conexão)

Montagem



1. Conectando o Sensor de movimento (PIR):
 - a. Conecte o pino de saída do sensor PIR ao pino digital 2 do Arduino.
 - b. Conecte o pino de alimentação do sensor PIR ao pino 5V do Arduino.
 - c. Conecte o pino de terra (GND) do sensor PIR ao GND.
2. Conectando um LED corretamente:
 - a. Conecte o anodo (perninha mais longa) do LED ao pino digital 9 do Arduino.
 - b. Conecte a catodo (perninha mais curta) do LED a um resistor (verifique qual o melhor resistor de acordo com a cor e a nossa tabela na seção LED do capítulo "Conhecendo os componentes").
 - c. Conecte outra extremidade do resistor ao terra (GND).

3. Observações importantes:

- a. Posicione o sensor PIR de tal forma que seja possível detectar a aproximação de alguém ou algum movimento.
- b. O LED está fazendo o papel de uma lâmpada comum, imagine que estamos prototipando uma lâmpada que acenda sozinha ao detectar a presença de uma pessoa no corredor, por exemplo.
- c. O sensor de movimento pode ser substituído por um fotoresistor (LDR), usado no projeto Girassol (nº 2), caso queira programar que a lâmpada acenda quando o ambiente estiver escuro, por exemplo.

Código

```
int pirPin = 2; // Pino de entrada do sensor PIR
int ledPin = 9; // Pino de controle do LED
int pirState = LOW; // Variável para armazenar o estado do sensor de movimento
int val = 0; // Variável para armazenar o valor lido do sensor PIR

void setup() {
    pinMode(pirPin, INPUT); // Configura o pino do sensor PIR como entrada
    pinMode(ledPin, OUTPUT); // Configura o pino do LED como saída
    Serial.begin(9600); // Inicializa a comunicação serial (opcional, para depuração)
}

void loop() {
    val = digitalRead(pirPin); // Lê o valor do sensor PIR

    if (val == HIGH) { // Verifica se há movimento detectado
        digitalWrite(ledPin, HIGH); // Se houver movimento, acende o LED
        if (pirState == LOW) { // O estado pirState está LOW?
            Serial.println("Movimento Detectado!"); // Escrevemos no nosso "diário"
            pirState = HIGH; // Atualiza o estado do sensor
        }
    } else { // Se não houver movimento, apaga o LED
        digitalWrite(ledPin, LOW);
        if (pirState == HIGH) { // O estado pirState está HIGH?
            Serial.println("Sem Movimento"); // Escrevemos no nosso "diário"
            pirState = LOW; // Atualiza o estado do sensor
        }
    }
}
```

Explicação do Código:

1. Definição de Pinos:

- `pirPin`: O pino onde conectamos o sensor de movimento PIR.
- `ledPin`: O pino onde conectamos o LED.

2. Variáveis para Controle:

- `pirState`: Armazena o estado atual do sensor de movimento (movimento ou sem movimento).
- `val`: Armazena o valor lido do sensor PIR (HIGH ou LOW).

3. Setup:

- Configuramos os pinos como entrada (sensor PIR) e saída (LED).
- Inicializamos a comunicação serial para depuração (opcional).

4. Loop:

- Lemos o valor do sensor PIR para saber se há movimento.
- Se houver movimento, acendemos o LED e atualizamos o estado.
- Se não houver movimento, apagamos o LED e atualizamos o estado.
- As mensagens "Movimento Detectado!" e "Sem Movimento" são exibidas no Monitor Serial (opcional).

Este código faz com que a luminária se acenda automaticamente quando detecta movimento e se apague quando não há movimento. É como uma luminária mágica que sabe quando você está por perto

4. Jogo da Batata Quente

Construa um jogo da Batata Quente eletrônico onde um LED acende de forma intermitente, e quando o jogador pressiona um botão, a "batata" é passada para o próximo jogador.

Materiais

- | | |
|------------------------------|-----------------------------------|
| 1. Arduino Uno | 4. 4 Resistores (conforme tabela) |
| 2. 4 Protoboards | 5. 4 Botões (push-button) |
| 3. 4 LEDs (cores diferentes) | 6. Cabos Jumper (fios de conexão) |

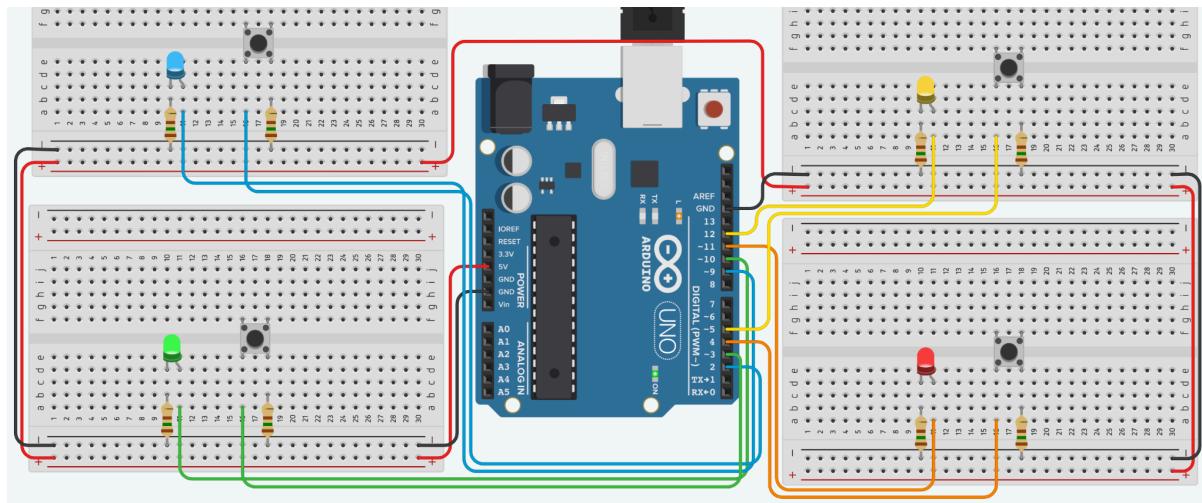
Montagem

1. Conectando os LED's

- a. Conecte o anodo (perninha mais longa) do LED a um resistor (conforme tabela de resistor x cor).
 - b. Conecte a outra extremidade do resistor a um pino digital específico (Azul: [pino 9](#), Verde: [pino 10](#), Vermelho: [pino 11](#) e Amarelo: [pino 12](#)).
 - c. Conecte o catodo (perna mais curta) do LED ao terra (GND) na protoboard.

2. Conectando os botões

- a. Conecte um terminal do botão a um pino digital específico (Azul: **pino 2**, Verde: **pino 3**, Vermelho: **pino 4**, Amarelo: **pino 5**).
 - b. Conecte o outro terminal do botão ao terra (**GND**) na protoboard.



3. Observações:

- a. Cada LED e botão têm cores correspondentes para indicar o jogador no jogo.
 - b. Os LEDs piscam intermitentemente, e os jogadores devem pressionar os botões correspondentes quando o LED estiver aceso para passar a "batata" para o próximo jogador.
 - c. O jogo continua até que um jogador pressione o botão quando o LED estiver apagado.

Código

```
const int azulLED = 9;    // Pino do LED Azul
const int verdeLED = 10;   // Pino do LED Verde
const int vermelhoLED = 11; // Pino do LED Vermelho
const int amareloLED = 12; // Pino do LED Amarelo

const int azulBotao = 2;   // Pino do Botão Azul
const int verdeBotao = 3;  // Pino do Botão Verde
const int vermelhoBotao = 4; // Pino do Botão Vermelho
const int amareloBotao = 5; // Pino do Botão Amarelo

// Variáveis de controle
int jogadorPerdedor = 0; // Armazena o jogador que perdeu

// Funções
void piscarLED(int pinoLED, int tempo);
void passarBatata(int pinoLED, int pinoBotao);
void jogadorPerdeu(int pinoLED);

void setup() {
    pinMode(azulLED, OUTPUT); // Configura o pino do LED como saída
    pinMode(verdeLED, OUTPUT);
    pinMode(vermelhoLED, OUTPUT);
    pinMode(amareloLED, OUTPUT);

    pinMode(azulBotao, INPUT_PULLUP); // Configura o pino do Botão como entrada
    pinMode(verdeBotao, INPUT_PULLUP);
    pinMode(vermelhoBotao, INPUT_PULLUP);
    pinMode(amareloBotao, INPUT_PULLUP);
}

void loop() {
    // Jogo com o LED Azul
    piscarLED(azulLED, 1000);
    passarBatata(azulLED, azulBotao);

    // Jogo com o LED Verde
    piscarLED(verdeLED, 1000);
    passarBatata(verdeLED, verdeBotao);

    // Jogo com o LED Vermelho
    piscarLED(vermelhoLED, 1000);
    passarBatata(vermelhoLED, vermelhoBotao);
```

```

// Jogo com o LED Amarelo
piscarLED(amareloLED, 1000);
passarBatata(amareloLED, amareloBotao);
}

// Função para fazer o LED piscar
void piscarLED(int pinoLED, int tempo) {
    digitalWrite(pinoLED, HIGH); // Acende o LED
    delay(tempo / 2); // Espera metade do tempo
    digitalWrite(pinoLED, LOW); // Apaga o LED
    delay(tempo / 2); // Espera a outra metade do tempo
}

// Função para passar a batata quando o botão é pressionado
void passarBatata(int pinoLED, int pinoBotao) {
    while (digitalRead(pinoBotao) == HIGH) {
        // Aguarda o jogador pressionar o botão
    }

    // Quando o botão é pressionado, passa a batata para o próximo jogador
    digitalWrite(pinoLED, LOW); // Apaga o LED
    delay(1000); // Aguarda um momento (1s)

    // Verifica se o jogador pressiona o botão com o LED apagado (jogador perdeu)
    if (digitalRead(pinoBotao) == LOW) {
        jogadorPerdeu(pinoLED);
    }
}

// Quando o botão é pressionado, passa a batata para o próximo jogador
digitalWrite(pinoLED, LOW); // Apaga o LED
delay(1000); // Aguarda um momento
}

// Função para indicar que o jogador perdeu
void jogadorPerdeu(int pinoLED) {
    jogadorPerdedor = pinoLED; // Armazena o jogador que perdeu
    for (int i = 0; i < 5; i++) {
        digitalWrite(jogadorPerdedor, HIGH); // Acende o LED do jogador perdedor
        delay(100);
        digitalWrite(jogadorPerdedor, LOW); // Apaga o LED do jogador perdedor
        delay(100);
    }
    delay(1000); // Aguarda um momento antes de continuar
}

```

Explicando o código:

Cada LED é como uma batata colorida, e eles ficam "quentes" quando acendem. Cada jogador tem seu botão, pronto para passar a batata para o próximo jogador quando o LED está "quente". Agora, imagine que os LEDs são como batatas coloridas nas mãos dos jogadores.

piscarLED: Essa função é como a música do nosso jogo. Os LEDs brilham por um tempo e depois se apagam, criando uma dança de batatas. É como se estivéssemos passando as batatas uns para os outros em uma roda.

passarBatata: Aqui, o jogo fica emocionante! Quando o LED está "quente" e brilhante, os jogadores pressionam seus botões para passar a batata. Se apertarem na hora certa, a batata vai para o próximo jogador. Mas, cuidado! Se apertarem com o LED apagado, a batata cai, e o jogador perde.

jogadorPerdeu: Quando alguém aperta o botão errado e perde, é como se tivesse segurado a batata quente por muito tempo. Todos os LEDs se apagam, e o LED do jogador que perdeu pisca rápido, para sinalizar quem perdeu o jogo.

5. Jogo Genius

Construa um jogo onde os LEDs acendem em sequência e o jogador deve pressionar o botão na ordem correta.

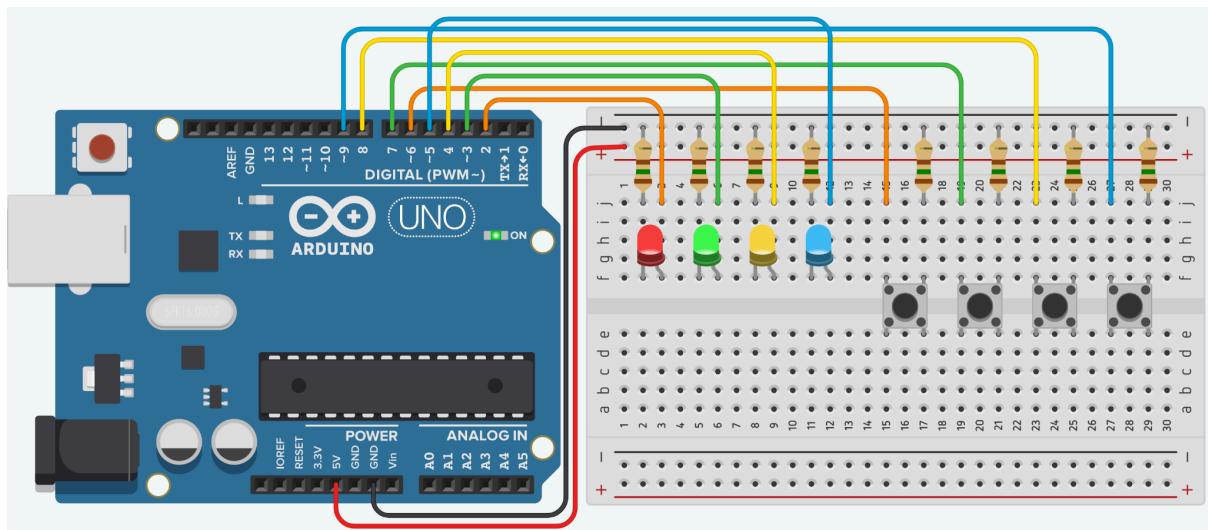
Materiais

- | | |
|--------------------------|-------------------|
| 1. LEDs Coloridos (4) | 3. Resistores (4) |
| 2. Botões de Pressão (4) | 4. Jumpers |
| | 5. Arduino UNO. |

Montagem

1. Conectando os LEDS:
 - a. **LED 1 (Vermelho):** Pino digital 2, com resistor para o GND.
 - b. **LED 2 (Verde):** Pino digital 3, com resistor para o GND.
 - c. **LED 3 (Amarelo):** Pino digital 4, com resistor para o GND.
 - d. **LED 4 (Azul):** Pino digital 5, com resistor para o GND.
2. Conectando os Botões:
 - a. **Botão 1 (Vermelho):** Conectado ao pino digital 6.

- b. **Botão 2 (Verde):** Conectado ao pino digital 7.
- c. **Botão 3 (Amarelo):** Conectado ao pino digital 8.
- d. **Botão 4 (Azul):** Conectado ao pino digital 9



Código

```
// Definindo os pinos dos LEDs
const int ledPinos[] = {2, 3, 4, 5};
// Definindo os pinos dos botões
const int botaoPinos[] = {6, 7, 8, 9};

// Definindo a quantidade inicial de cores na sequência
int numCores = 2;
// Array para armazenar a sequência de cores
int sequenciaCores[10]; // Pode ajustar conforme necessário

// Função para piscar o LED vermelho indicando game over
void piscarGameOver() {
    for (int i = 0; i < 10; i++) {
        digitalWrite(ledPinos[0], HIGH);
        delay(100);
        digitalWrite(ledPinos[0], LOW);
        delay(100);
    }
}
```

```

// Função para gerar uma nova sequência de cores
void gerarNovaSequencia() {
    for (int i = 0; i < numCores; i++) {
        // Gerando número aleatório entre 0 e 3 (representando as cores)
        sequenciaCores[i] = random(4);
        // Acendendo o LED correspondente à cor gerada
        digitalWrite(ledPinos[sequenciaCores[i]], HIGH);
        delay(500); // Pode ajustar o tempo de exibição de cada cor
        digitalWrite(ledPinos[sequenciaCores[i]], LOW);
        delay(100); // Pausa entre as cores
    }
}

void setup() {
    // Configurando os pinos dos LEDs como saídas
    for (int i = 0; i < 4; i++) {
        pinMode(ledPinos[i], OUTPUT);
    }
    // Configurando os pinos dos botões como entradas
    for (int i = 0; i < 4; i++) {
        pinMode(botaoPinos[i], INPUT_PULLUP);
    }
}

void loop() {
    // Aguardando o jogador tentar a sequência atual antes de iniciar uma nova
    while (digitalRead(botaoPinos[0]) == HIGH &&
           digitalRead(botaoPinos[1]) == HIGH &&
           digitalRead(botaoPinos[2]) == HIGH &&
           digitalRead(botaoPinos[3]) == HIGH) {
        delay(100); // Aguardando jogador pressionar qualquer botão
    }

    // Resetando o jogo
    numCores = 2;
}

```

```

while (true) {
    // Gerando uma nova sequência
    gerarNovaSequencia();

    // Aguardando o jogador tentar a sequência gerada
    for (int i = 0; i < numCores; i++) {
        // Aguardando o jogador pressionar o botão correspondente à cor
        while (digitalRead(botaoPinos[sequenciaCores[i]]) == HIGH) {
            delay(100); // Aguardando jogador pressionar o botão
        }
        delay(100); // Evitando leituras múltiplas do mesmo botão
        // Acendendo o LED correspondente à cor
        digitalWrite(ledPinos[sequenciaCores[i]], HIGH);
        delay(500); // Pode ajustar o tempo de exibição de cada cor
        digitalWrite(ledPinos[sequenciaCores[i]], LOW);
        delay(100); // Pausa entre as cores
    }

    // Incrementando o número de cores na próxima rodada
    numCores++;

    // Sinalizando que o jogador perdeu
    piscarGameOver();
}
}

```

Explicando o código:

- Definição dos pinos:** Os LEDs e botões são conectados a pinos específicos no Arduino. Os LEDs estão nos pinos 2, 3, 4 e 5, enquanto os botões estão nos pinos 6, 7, 8 e 9.
- Quantidade inicial de cores:** A variável `numCores` armazena a quantidade inicial de cores na sequência. No início do jogo, são duas cores.
- Array para armazenar a sequência de cores:** O array `sequenciaCores` armazenará as cores geradas aleatoriamente para cada rodada. Um array é como uma série de caixas numeradas, cada uma armazenando um item específico. Em programação, um array é uma estrutura que

permite agrupar vários valores sob um único nome. Cada valor no array ocupa uma posição, identificada por um número chamado índice. Ao referenciar o índice desejado, podemos acessar e manipular o valor contido na posição correspondente do array. Por exemplo, se tivermos um array de números, ao usar o índice 1, acessamos o segundo número na sequência. Isso porque o computador começa a contagem a partir do zero, logo o índice 1 é o segundo valor da sequência. Isso torna os arrays uma ferramenta poderosa para organizar e manipular conjuntos de dados de maneira eficiente em programação, tal como implementamos aqui.

4. **Função** piscarGameOver: Essa função faz o LED vermelho piscar para indicar que o jogador perdeu.
5. **Função** gerarNovaSequencia: Essa função gera uma nova sequência de cores aleatórias. Ela utiliza um loop para atribuir valores aleatórios a cada elemento do array `sequenciaCores`, representando as cores dos LEDs.
6. **Função** setup: Configuração inicial dos pinos, definindo se são de entrada ou saída.
7. **Loop principal (função loop):**
 - o **Espera o jogador iniciar a partida:** O jogo espera o jogador pressionar qualquer botão para começar.
 - o **Reinicia o jogo:** Quando o jogador pressionar algum botão, o jogo é reiniciado com duas cores na sequência.
 - o **Loop do jogo principal:**
 - **Gera nova sequência:** A função `gerarNovaSequencia` é chamada para criar a sequência de cores.
 - **Jogador tenta a sequência gerada:** O jogador deve pressionar os botões na ordem correta da sequência.
 - **Exibe a sequência:** Os LEDs acendem conforme a sequência gerada.
 - **Incrementa a dificuldade:** A cada rodada bem-sucedida, a quantidade de cores aumenta.
 - **Sinaliza game over se o jogador errar:** Se o jogador errar a sequência, o LED vermelho pisca para indicar game over.

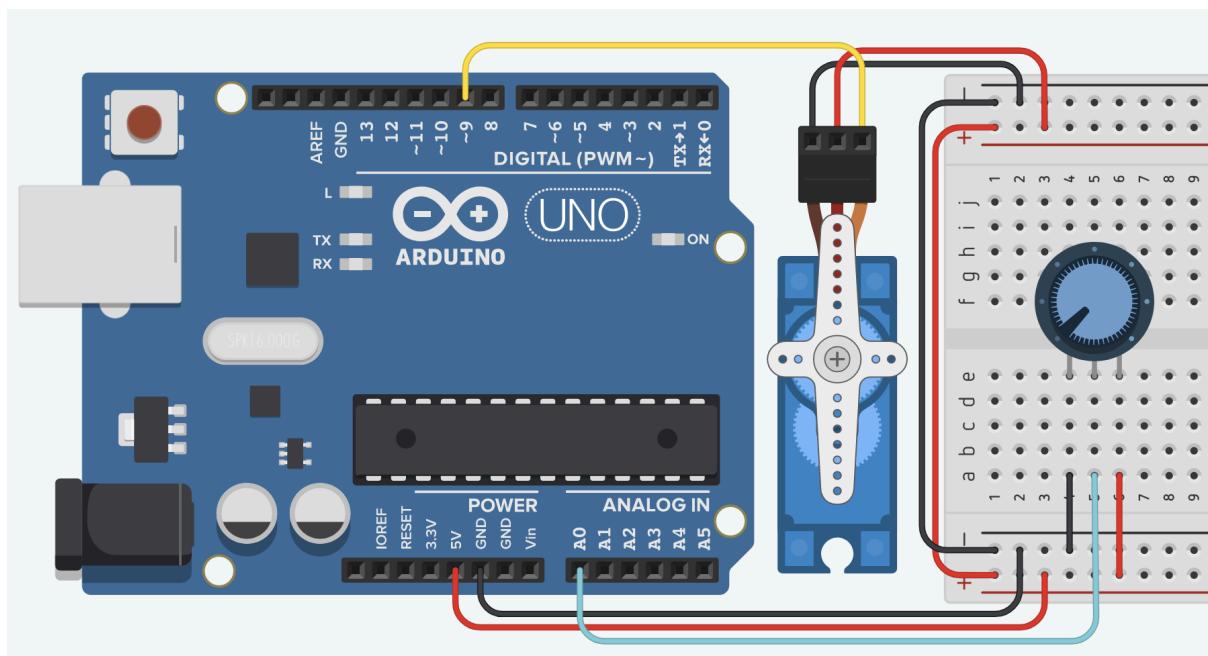
6. Controle de Servo-Motor com Potenciômetro

Use o potenciômetro para controlar a posição do servo-motor.

Materiais

1. Arduino UNO
2. Protoboard
3. Servo-motor
4. Potenciômetro
5. Jumpers (fios de conexão)

Montagem



1. Preparando o protoboard:
 - a. Conecte o fio de alimentação (+5V) do Arduino a uma linha positiva da protoboard.
 - b. Conecte o fio de terra (GND) do Arduino a uma linha negativa da protoboard.
2. Conectando o potenciômetro:
 - a. Conecte o pino central do potenciômetro a um pino analógico do Arduino (por exemplo, A0).
 - b. Conecte um dos pinos do potenciômetro à linha positiva da protoboard.
 - c. Conecte o outro pino do potenciômetro à linha negativa da protoboard.

3. Conectando o servo-motor:

- a. Conecte o fio de controle do servo-motor a um pino digital do Arduino (por exemplo, 9).
- b. Conecte o fio de alimentação (+) do servo-motor à linha positiva da protoboard.
- c. Conecte o fio de terra (-) do servo-motor à linha negativa da protoboard.

Código

```
// Incluímos a biblioteca do Servo-Motor
#include <Servo.h>

// Criamos um objeto Servo para controlar o servo-motor
Servo meuServo;

// Definimos o pino analógico ao qual o potenciômetro está conectado
const int pinoPotenciometro = A0;

void setup() {
    // Anexamos o objeto Servo ao pino digital 9
    meuServo.attach(9);
}

void loop() {
    // Leitura do valor do potenciômetro (varia de 0 a 1023)
    int valorPotenciometro = analogRead(pinoPotenciometro);

    // Mapeamos o valor lido para a faixa de movimento do servo (0 a 180 graus)
    int angulo = map(valorPotenciometro, 0, 1023, 0, 180);

    // Movemos o servo para a posição calculada
    meuServo.write(angulo);

    // Pequeno atraso para suavizar o movimento e evitar leituras muito rápidas
    delay(15);
}
```

Explicando o código:

1. Incluímos uma Biblioteca:

- No início, dizemos ao Arduino que vamos usar uma biblioteca especial chamada "Servo". Essa biblioteca é como um livro de instruções que ajuda a controlar o servo-motor.

2. Criamos um personagem para o Servo-Motor:

- Damos vida ao servo-motor criando um "personagem" chamado meuServo. Ele será responsável por mover o servo-motor como quisermos.

3. Indicamos o Pino do Potenciômetro:

- Dizemos ao Arduino em qual pino analógico está conectado o potenciômetro, como se estivéssemos apontando para a sala onde está guardado.

4. Configuramos o Jogo no Começo:

- Antes de começar, conectamos o servo-motor ao "mundo" do Arduino, dizendo que ele vai usar o pino digital 9.

5. Começa a Diversão no Loop:

- Agora, em um loop infinito, lemos o valor do potenciômetro, que é como espiar para ver em que posição está naquele momento.

6. Traduzindo o Desejo do Potenciômetro:

- Transformamos o valor do potenciômetro em um ângulo para o servo-motor. É como traduzir uma linguagem para outra.

7. Dizendo ao Servo-Motor o que Queremos:

- Finalmente, pedimos ao servo-motor para ir para a posição desejada. É como dar uma ordem ao servo-motor para se mover.

8. Pequeno Descanso:

- Adicionamos um pequeno descanso para que o servo-motor não se mova muito rápido e pareça suave, como se estivesse movendo em câmera lenta.

Observação:

Este projeto é o princípio básico para projetos mais complexos. Usamos esta base para construir, por exemplo, um braço mecânico. Podemos fazer uma combinação de vários servo motores em sequência para controlar as movimentações do braço e de seu antebraço.

Agradecimentos Especiais

Queremos expressar nossa profunda gratidão a todos que tornaram possível a realização deste projeto de extensão com Arduino na Escola Padrão Parque das Nações, em Poços de Caldas, Minas Gerais. Cada contribuição, apoio e orientação foram fundamentais para o sucesso dessa iniciativa.

Agradecemos ao Vice-Diretor da Escola Padrão, Matheus dos Santos, por abrir as portas e permitir que a inovação e o aprendizado prático florescessem em sua instituição.

À equipe de professores e colaboradores da Escola Padrão, nosso reconhecimento pelos esforços contínuos em proporcionar um ambiente educacional inspirador. Em particular, agradecemos pela inestimável ajuda na conferência e criação do inventário. Sem este trabalho meticoloso, não seríamos tão assertivos na escolha e implementação dos projetos desta cartilha.

Um agradecimento especial aos professores orientadores da PUC-Minas, Luciana de Nardin e Luis Gustavo Fogaroli. Sua dedicação e mentoria foram a âncora deste projeto, guiando-nos com sabedoria e experiência.

À Pontifícia Universidade Católica de Minas Gerais – Poços de Caldas (PUC-MG PPC), nossa instituição de ensino, expressamos nossa gratidão pela oportunidade de aplicar nossos conhecimentos em um contexto prático, enriquecendo não apenas nossas vidas acadêmicas, mas também contribuindo para a comunidade local.

Por fim, agradecemos aos nossos pares, membros do nosso grupo de trabalho. Caio Augusto de Oliveira, Luis Felipe Lima Balduino, Matheus Eduardo Silva, Victor Matos e Yury Regis Neiva Pereira – cada um desempenhou um papel vital na realização deste projeto, contribuindo com dedicação e habilidade.

Juntos, celebramos a colaboração, a educação prática e o espírito inovador que moldou este projeto de extensão. O sucesso alcançado é um reflexo do trabalho conjunto e do compromisso com a busca do conhecimento.