INSTITUTO FEDERAL
Triângulo Mineiro
Campus Uberlândia Centro

GPETEC
Grupo de Pesquisa em Educação, Tecnologia e Ciências.

Programando com **Arduino**

Uberlândia

2023

2023, IFTM Campus Uberlândia Centro

GPETEC - Grupo de Pesquisa em Educação, Tecnologia e Ciências

CRIAR - Clube de Robótica IFTM Arduíno e Raspberry

Projetos Participantes

Ensino: Robótica no Campus UdiCentro (Edital 01/2023)

Extensão: Robótica no Ensino Médio (Edital UdiCentro 01/2020)

Iniciação à Automação no CSEUB (Edital 09/2019)

Pesquisa: Pesquisando a Robótica Educacional como Alternativa Didático-Pedagógica (Edital 03/2020 PIVIC)

Pesquisando a Robótica no Ambiente Educacional (Edital 15/2018 – PIBIC)

Autores

Walteno Martins Parreira Júnior - Professor da Licenciatura em Computação, Coordenador de Projeto de Pesquisa/Extensão e Vice-Lider do Grupo de Pesquisa GPETEC

Carlos Magno Medeiros Queiroz - Professor da Licenciatura em Computação, Coordenador de Projeto de Extensão e Membro do Grupo de Pesquisa GPETEC

Kenedy Lopes Nogueira - Professor da Licenciatura em Computação e Lider do Grupo de Pesquisa GPETEC

Cristiano Borges dos Santos - Técnico Administrativo e Coordenador de Projeto de Extensão, Membro do Grupo de Pesquisa GPETEC

Fernando Guimarães Silva - Discente Licenciatura em Computação e Bolsista de Projeto de Ensino, Membro do Grupo de Pesquisa GPETEC

Revisão

De responsabilidade dos autores.

Capa / Arte

Walteno Martins Parreira Júnior

Dados Internacionais de Catalogação na Publicação

SUMÁRIO

Apresentação

Este material é fruto de uma somatória de esforços para o desenvolvimento da área de Robótica Educacional no Campus Uberlândia Centro nos últimos anos tanto entre os alunos do ensino médio quanto com os discentes do Curso de Licenciatura em Computação.

Inicialmente este texto foi desenvolvido como uma apostila denominada **Robótica – Tinkercad** pelo Professor Walteno para as aulas de Robótica Educacional do Curso de Licenciatura em Computação e utilizado juntamente com outro texto denominado **Robótica Educacional** que foi desenvolvido ao longo de alguns projetos de extensão e pesquisa orientado por vários servidores e discentes do Campus nos anos de 2018 e 2019 e que aborda a utilização do Arduino.

O Grupo de Pesquisa em Educação, Tecnologia e Ciências (GPETEC) possui duas linhas de pesquisas que contribuem nesta área, que são a) Automação, Controle de Processos, Instrumentação e Robótica e b) Desenvolvimento de aplicativos tecnológicos e softwares educacionais. E também há o Clube de Robótica IFTM Arduíno e Raspberry (CRIAR) que é parte do GPETEC e que apoia as atividades educativas e de competição em que os discentes participam.

Assim, esperamos que este material instrucional possa contribuir para o desenvolvimento da Robótica Educacional em nossas instituições educacionais.

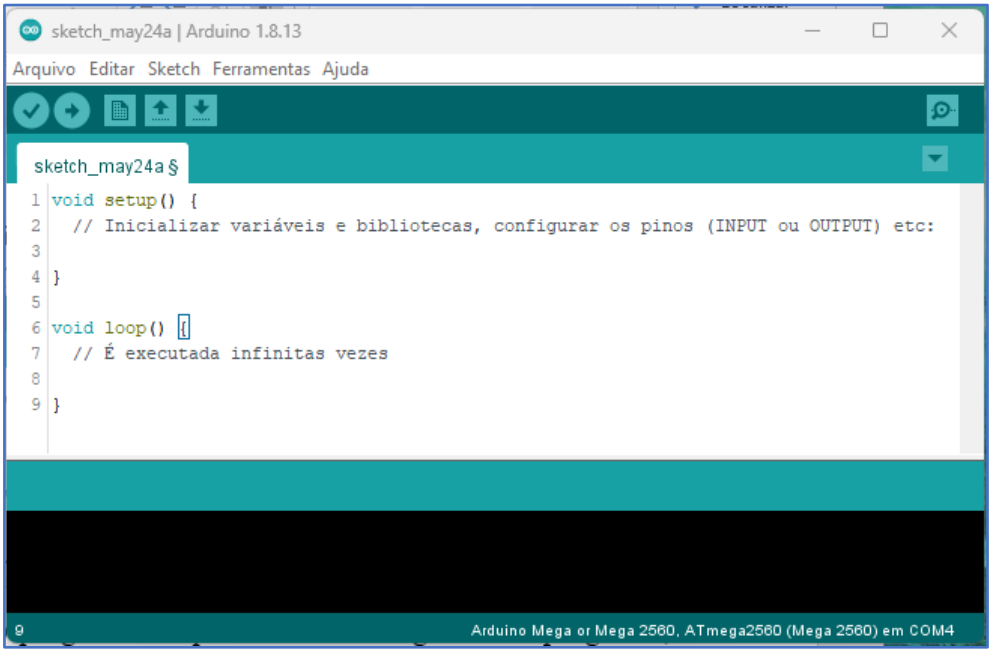
Cristiano Borges dos Santos

Walteno Martins Parreira Júnior

1. O Ambiente do Software Arduino (IDE)

1.1. Tela Inicial

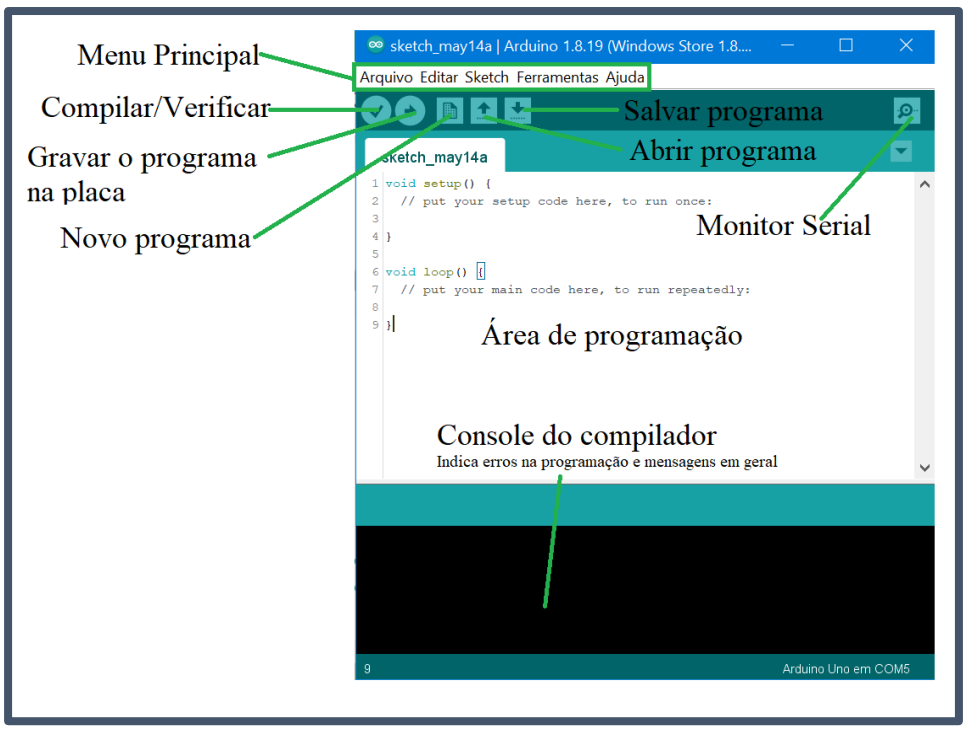
Figura 1 – Tela inicial da IDE



Fonte: Autoria própria (2023)

1.2. Ícones do Menu

Figura 2 – Tela da IDE



Fonte: Autoria própria (2023)

2. Programação

2.1. Introdução

Os programas para o Arduino são implementados tendo como referência a linguagem C++. Preservando sua sintaxe clássica na declaração de variáveis, nos operadores, nos ponteiros, nos vetores, nas estruturas e em muitas outras características da linguagem.

Elas podem ser divididas em três partes principais: As estruturas, os valores (variáveis e constantes) e as funções.

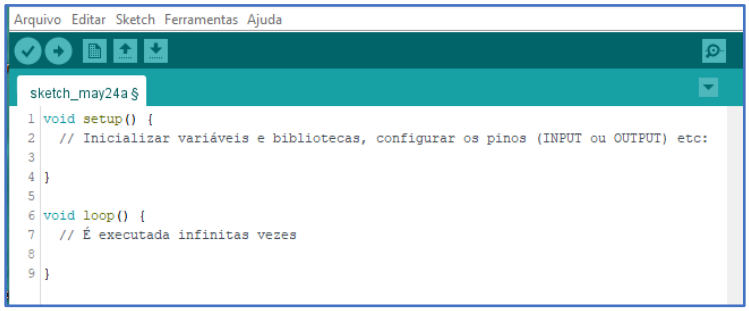
2.2. Funções

O Arduino permite várias funções na programação. Duas funções são obrigatórias em um programa.

Setup() // A função setup() é chamada quando um sketch inicia. Use-a para inicializar variáveis, configurar o modo dos pinos (INPUT ou OUTPUT), inicializar bibliotecas, etc. A função setup() será executada apenas uma vez, após a placa ser alimentada ou acontecer um reset

Loop() // A função loop() é executada infinitas vezes. Inicialmente ela é inicializada logo após a função setup() e vai ficar executando até receber um comando de parada ou não ter mais energia para execução.

Figura 3 – Iniciando um Programa ou sketch



Fonte: Autoria própria (2023)

Outras funções podem ser criadas pelo programador para facilitar / organizar o programa, facilitando o entendimento ou reduzindo o tamanho do código.

2.3. Constantes

HIGH // O significado de HIGH (em relação a um pino) depende se o pino está configurado como entrada ou saída (INPUT ou OUTPUT). Quando um pino é configurado como INPUT com pinMode(), e lido com digitalRead(), o Arduino (ATmega) irá retornar HIGH se:

- uma tensão maior que 3.0V está presente no pino (em placas 5V)
- uma tensão maior que 2.0V está presente no pino (em placas 3.3V)

LOW // O significado de LOW depende também se o pino está configurado como entrada ou saída (INPUT ou OUTPUT). Quando um pino é configurado como INPUT com pinMode(), e lido com digitalRead(), o Arduino (ATmega) irá retornar LOW se:

- uma tensão menor que 1.5V está presente no pino (em placas 5V)
- uma tensão menor que 1.0V (aproxidamente) está presente no pino (em placas 3.3V)

INPUT // Quando configurados como INPUT, esses pinos exigem multíssimo pouco dos circuitos que estão verificando, equivalente a um resistor de 100 Megohm na frente do pino. Isso faz com que sejam muito úteis para ler sensores.

OUTPUT // são ditos estarem em um estado de *baixa-impedância*. Isso significa que esses podem fornecer uma quantidade substancial de corrente para outros circuitos.

Bibliotecas: Serial, Servo, Tone, etc.

Sintaxe básica: #define, #include, ; , ...

2.1.1. As estruturas de referências:

Estruturas de controle: if, else, break, ...

Operadores aritméticos e de comparação: +, -, =, ==, !=, ...

Operadores booleanos: &&, ||, !

Acesso a ponteiros: *, &

Operadores compostos: ++, {, +=, ...

Operadores de bits: |, ^ , , ...

2.1.2. Os valores de referências:

Tipos de dados: byte, array, int , char , ...

Conversões: char(), byte(), int(), ...

Variável de escopo e de qualificação: variable scope, static, volatile, ...

Utilitários: sizeof() - diz o tamanho da variável em bytes

2.1.3. Funções e constantes no Arduino

O Arduino já provê várias funções e constantes para facilitar a programação:

Setup() // Stream é a classe base para streams de caracteres ou binárias. Não é chamada diretamente, mas invocada quando você usa uma função que depende dessa classe.

Loop() // A função setup() é chamada quando um sketch inicia. Use-a para inicializar variáveis, configurar o modo dos pinos (INPUT ou OUTPUT), inicializar bibliotecas, etc. A função setup() será executada apenas uma vez, apoós a placa ser alimentada ou acontecer um reset.

Constantes:

HIGH // O significado de HIGH (em relação a um pino) depende se o pino está configurado como entrada ou saída (INPUT ou OUTPUT). Qaundo um pino é configurado como INPUT com pinMode(), e lido com digitalRead(), o Arduino (ATmega) irá retornar HIGH se:

- uma tensão maior que 3.0V está presente no pino (em placas 5V)
- uma tensão maior que 2.0V está presente no pino (em placas 3.3V)

LOW // O significado de LOW depende também se o pino está configurado como entrada ou saída (INPUT ou OUTPUT). Qaundo um pino é configurado como INPUT com pinMode(), e lido com digitalRead(), o Arduino (ATmega) irá retornar LOW se:

- uma tensão menor que 1.5V está presente no pino (em placas 5V)
- uma tensão menor que 1.0V (aproxidamente) está presente no pino (em placas 3.3V)

INPUT // Quando configurados como INPUT, esses pinos exigem multíssimo pouco dos circuitos que estão verificando, equivalente a um resistor de 100 Megohm na frente do pino. Isso faz com que sejam muito úteis para ler sensores.

OUTPUT // são ditos estarem em um estado de *baixa-impedância*. Isso significa que esses podem fornecer uma quantidade substancial de corrente para outros circuitos.

Bibliotecas: Serial, Servo, Tone, etc.

2.4. FUNÇÕES

Essas funções estão implementadas e disponíveis em bibliotecas que executam funcionalidades básicas do microcontrolador. Algumas não estão implementadas no Tinkercad, somente no Arduino.

2.4.1. ENTRADAS E SAÍDAS DIGITAIS

- **pinMode()** // Configura o pino especificado para funcionar como uma entrada ou saída.
- **digitalWrite()** // Aciona um valor HIGH ou LOW em um pino digital.
- **digitalRead()** // Configura o pino especificado para funcionar como uma entrada ou saída.

2.4.2. ENTRADAS E SAÍDAS ANALÓGICAS

- **analogReference()** // Configura a tensão de referência para a entrada analógica (o valor máximo do intervalo de entrada).
- **analogRead()** // Lê o valor de um pino analógico especificado. A placa Arduino possui um conversor analógico-digital 10 bts de 6 canais (8 canais nos Mini e Nano, 16 no Mega, 7 canais em placas MKR).
- **analogWrite()** // Aciona uma onda PWM (descrição (Em Ingês)) em um pino. Pode ser usada para variar o brilho de um LED ou acionar um motor a diversas velocidades.

2.4.3. APENAS ZERO, DUE E FAMÍLIA MKR

- **analogReadResolution()** // analogReadResolution() é um extensão da API Analog para o Arduino Due, Zero e família MKR.Configura o tamanho (em bits) do valor retornado por analogRead(). O padrão é 10 bits (retorna valores entre 0-1023) para compatibilidade com placas baseadas em microcontroladores AVR.
- **analogWriteResolution()** // analogWriteResolution() é uma extensão da API Analog para os Arduinos Due, Zero e MKR. nalogWriteResolution() configura a resolução da função analogWrite(). O padrão é 8 bits (valores entre 0-255) para compatibilidade com placas baseadas em microcontroladores AVR.

2.4.4. ENTRADAS E SAÍDAS AVANÇADAS

- **tone()** // Gera uma onda quadrada na frequência especificada (e duty cycle 50%) em um pino. A duração pode ser especificada, do contrário a onda continua até uma chamada de noTone(). O pino pode ser conectado a um buzzer piezo ou outro speaker para tocar tons.
- **noTone()** // Interrompe a geração de uma onda quadrada iniciada pela função tone(). Não tem nenhum efeito se nenhum tom está sendo gerado.
- **shiftIn()** // Recebe um byte de dados, um bit de cada vez. Começa com ou o bit mais significante (o mais à esquerda) ou o menos significante (mais à direita). Para cada bit, o pino de clock é colocado em estado em HIGH, o próximo bit é lido no pino de dados (data), e então o pino é colocado em estado LOW novamente.
- **shiftOut()** // Transfere um byte de dados um bit de cada vez. Começa com ou o bit mais significante (o mais à esquerda) ou o menos significante (mais à direita). Cada bit é escrito em sequência em um pino data, logo após o pino clock é pulsado (colocado em HIGH, depois LOW) para indicar que aquele bit está disponível.

- **pulseIn()** // Captura a duração de um pulso em um pino (que pode ser HIGH ou LOW). Por exemplo, se o valor HIGH é passado para a função, a função pulseIn() espera o pino ir para do estado 'LOW' para HIGH, começa a temporizar, então espera o pino ir para o estado LOW e para de temporizar. Retorna o tamanho do pulso em microssegundos ou desiste e retorna 0 se não receber nenhum pulso dentro de um tempo máximo de espera especificado.
- **pulseInLong()** // É uma alternativa à função pulseIn(), sendo melhor para lidar com pulsos longos e situações afetadas por interrupções.

2.4.5. FUNÇÕES TEMPORIZADORAS

- **millis()** //Retorna o número de milissegundos passados desde que a placa Arduino começou a executar o programa atual. Esse número irá sofrer overflow (chegar ao maior número possível e então voltar pra zero), após aproximadamente 50 dias.
- **micros()** //Retorna o número de microssegundos passados desde que a placa Arduino começou a executar o programa atual. Esse número irá sofrer overflow (chegar ao maior número possível e então voltar pra zero), após aproximadamente 70 minutos. Em placas Arduino 16 MHz (ex. UNO e Nano), essa função possui uma resolução de quatro microssegundos (isto é, o número retornado é sempre um múltiplo de quatro). Em placas Arduino 8 MHz (ex. LilyPad), essa função possui uma resolução de oito microssegundos.
- **delay()** //Pausa o programa por uma quantidade especificada de tempo (em milissegundos). Cada segundo equivale a 1000 milissegundos.
- **delayMicroseconds()** //Pausa o programa pela quantidade de tempo especificada como parâmetro (em microssegundos). Há mil microssegundos em um milissegundo, e um milhão de microssegundos em um segundo.Atualmente, o maior valor que irá produzir um delay preciso é 16383. Isso pode mudar em versões futuras do Arduino. Para delays mais longos que alguns milhares de microssegundos, você deve usar delay() em vez disso.

2.4.6. FUNÇÕES MATEMÁTICAS

- **abs()** //Calcula o módulo (ou valor absoluto) de um número.
- **constrain()** //Restringe um número a ficar dentro de um intervalo.

- **map()** // Remapeia um número de um intervalo para outro. Isto é, um valor de deMenor seria mapeado para paraMenor, um valor de deMaior para paraMaior, valores dentro de uma faixa para valores dentro da outra faixa, etc. Não restringe valores a ficar dentro do intervalo, porque valores fora do intervalo são as vezes úteis e pretendidos. A função constrain() pode ser usada tanto antes como depois dessa função, se limites para os intervalos são desejados.
- **max()** // Calcula o maior de dois números.
- **min()** // Calcula o menor de dois números.
- **pow()** // Calcula o valor de um número elevado a uma potência. pow() pode ser usada para transformar um número em uma potência fracionária. Isso pode ser útil para gerar mapeamentos exponenciais de valores ou curvas.
- **sq(0)** // Calcula o quadrado de um número: o número multiplicado por si mesmo.
- **sqrt()** //Calcula a raiz quadrada de um número.

2.4.7. FUNÇÕES TRIGONOMÉTRICAS

- **sin()** // Calcula o seno de um ângulo (em radianos). O resultado ficará entre -1 e 1.
- **cos()** // Calcula o cosseno de um ângulo (em radianos). O resultado é dado entre -1 e 1.
- **tan()** // Calcula a tangente de um ângulo (em radianos). O resultado é dado entre -infinito e +infinito (limitado pelo tamanho do tipo de dado double).

2.4.8. NÚMEROS ALEATÓRIOS

- **randomSeed()** // Inicializa o gerador de números pseudoaleatórios, fazendo o começar em um ponto arbitrário em sua sequência aleatória. Essa sequência, enquanto muito longa, e aleatória, é sempre a mesma. Se é importante que uma sequência de valores gerados por random() seja diferente em execuções subsequentes de um sketch, use randomSeed() para inicializar o gerador de números aleatórios com uma entrada significantemente aleatória, como analogRead() em um pino desconectado. Por outro lado, pode ser ocasionalmente útil usar sequências pseudoaleatórias exatamente repetidas. Isso pode ser conseguido chamando-se randomSeed() com um número fixo, antes de começar a usar a sequência aleatória.
- **random()** // A função random() gera números pseudoaleatórios.

2.4.9. CARACTERES

- **isAlpha()** // Analisa se um caractere é alfabético (isto é, se é uma letra). Retorna true (verdadeiro) se thisChar contém uma letra.
- **isAlphaNumeric()** // Analisa se um caractere é alfanumérico (isto é, uma letra ou um número). Retorna true (verdadeiro) se thisChar contém ou uma letra ou um número.

- **isAscii()** // Analisa se um caractere é Ascii. Retorna true se a variável thisChar contém um caractere Ascii.
- **isControl()** // Analisa se uma caractere é um caractere de controle. Retorna true (verdadeiro) se thisChar é um caractere de controle.
- **isDigit()** // Analisa se uma caractere é um digito (isto é, um número). Retorna true (verdadeiro) se thisChar é um número.
- **isGraph()** // Analisa se um caractere é imprimível com algum conteúdo (espaços são imprimíveis, mas não possuem conteúdo). Retorna true se thisChar é imprimível.
- **isHexadecimalDigit()** // Analisa se uma caractere é um digito hexadecimal (A-F, 0-9). Retorna true se thisChar contém um digito hexadecimal.
- **isLowerCase()** // Analisa se uma caractere é minúsculo (isto é, uma letra minúscula). Retorna true se thisChar contém uma letra minúscula.
- **isPrintable()** // Analisa se uma caractere é imprimível (isto é, qualquer caractere que produz uma saída, até mesmo um espaço). Retorna true se thisChar é imprimível. Por impimível diz-se os caracteres que podem ser impressos, como letras e números. Alguns caracteres são para controle e não podem ser impressos como o new line ('\n') e o tab ('\t').
- **isPunct()** // Analisa se uma caractere é imprimível (isto é, qualquer caractere que produz uma saída, até mesmo um espaço). Retorna true se thisChar é imprimível. Por impimível diz-se os caracteres que podem ser impressos, como letras e números. Alguns caracteres são para controle e não podem ser impressos como o new line ('\n') e o tab ('\t'). **isSpace()** // Analisa se uma caractere é o caractere de espaço. Retorna true se thisChar contém um espaço.
- **isUpperCase()** // Analisa se uma caractere é maiúsculo (isto é, uma letra maisúcula). Retorna true se thisChar é uma letra maiúscula.
- **isWhitespace()** // Analisa se uma caractere é maiúsculo (isto é, uma letra maisúcula). Retorna true se thisChar é uma letra maiúscula.

2.4.10. BITS E BYTES

- **bit()** // Computa o valor do bit especificado (o bit 0 é igual a 1, bit 1 igual a 2, bit 2 igual a 4, etc.).
- **bitClear()** // Limpa (escreve um 0) em um bit de uma variável numérica.
- **bitRead()** // Lê o valor de um bit em um número.
- **bitSet()** // Ativa (escreve 1 em) um bit de uma variável numérica.
- **bitWrite()** // Escreve em um bit especificado de um valor numérico.

- **highByte()** // Extrai o byte mais significativo (o mais à esquerda) de uma word (valor de 16 bits), ou o segundo byte menos significativo de um tipo de dado maior.
- **lowByte()** // Extrai o byte menos significativo (mais à direita) de uma variável (ex. uma word).

2.4.11. INTERRUPÇÕES EXTERNAS

- **attachInterrupt()** // O primeiro parâmetro de attachInterrupt() é o número da interrupção. É recomendado usar digitalPinToInterrupt(pino) para converter o número do pino digital para o número específico da interrupção. Por exemplo, se você usar o pino 3, passe digitalPinToInterrupt(3) como o primeiro parâmetro de attachInterrupt().
- **detachInterrupt()** // Desativa a interrupção especificada.

2.4.12. INTERRUPÇÕES

- **interrupts()** // Reativa interrupções (quando elas tiverem sido desativadas por nointerrupts().Interrupções permitem certas tarefas importantes acontecerem ao fundo e são, por padrão, ativadas. Algumas funções não irão funcionar enquanto as interrupções estiverem desativadas, e dados recebidos podem ser ignorados. Interrupções podem levemente interferir no timing do código, no entanto, e podem ser desativadas em seções particularmente críticas do código.
- **noInterrupts()** // Desativa interrupções (você pode reativá-las com interrupts()). Interrupções permitem certas tarefas importantes acontecerem ao fundo e são, por padrão, ativadas. Algumas funções não irão funcionar enquanto as interrupções estiverem desativadas, e dados recebidos podem ser ignorados. Interrupções podem levemente interferir no timing do código, no entanto, e podem ser desativadas em seções particularmente críticas do código.

2.4.13. COMUNICAÇÃO

- **Serial** // Usada para comunicação entre uma placa Arduino e um computador ou outros dispositivos. Todas as placas Arduino possuem pelo menos uma porta serial (também conhecida como UART ou USART), enquanto alguns possuem várias.
- **Stream** // Stream é a classe base para streams de caracteres ou binárias. Não é chamada diretamente, mas invocada quando você usa uma função que depende dessa classe.
- **USB**

- **Keyboard** // As funções da biblioteca Keyboard permitem placas baseadas nos micros 32u4 ou SAMD funcionar como um teclado e enviar sinais de teclas pressionadas a um computador conectado através do porta micro USB nativa.
- **Mouse** // As funções da biblioteca mouse permitem placas baseadas nos micros 32u4 ou SAMD controlar o movimento do cursor em um computador conectado através do porta micro USB nativa. Quando a posição do cursor é atualizada, essa é sempre relativa a sua posição anterior.

2.5. VARIÁVEIS

Constantes são expressões predefinidas na linguagem Arduino. Essas são usadas para tornar os programas mais legíveis. As constantes da linguagem Arduino são classificadas nos grupos:

- **HIGH** //O significado de HIGH (em relação a um pino) depende se o pino está configurado como entrada ou saída (INPUT ou OUTPUT). Qaundo um pino é configurado como INPUT com pinMode(), e lido com digitalRead(), o Arduino (ATmega) irá retornar HIGH
- **LOW** // O significado de LOW depende também se o pino está configurado como entrada ou saída (INPUT ou OUTPUT). Quando um pino é configurado como INPUT com pinMode(), e lido com digitalRead(), o Arduino (ATmega) irá retornar LOW
- **INPUT** // Pinos configurados como INPUT com pinMode() são ditos estarem em um estado de alta-impedância. Quando configurados como INPUT, esses pinos exigem multíssimo pouco dos circuitos que estão verificando, equivalente a um resistor de 100 Megohm na frente do pino.
- **OUTPUT** // Pinos configurados como OUTPUT com pinMode() são ditos estarem em um estado de baixa-impedância. Isso significa que esses podem fornecer uma quantidade substancial de corrente para outros circuitos. Os pinos de um ATmega podem fornecer ou absorver correntes de até 40 mA (miliamperes) para/de outros dispositivos/circuitos. Isso faz com que sejam úteis para alimentar LEDs, pois LEDs tipicamente usam menos de 40 mA.
- **INPUT_PULLUP** // O microcontrolador ATmega na maioria das placas Arduino possui resistores pull-up internos (resistores conectados a alimentação internamente) que você

pode acessar. Se você preferir usar estes em vez de resistores de pull-up externos, você pode usar o argumento INPUT_PULLUP na função pinMode().

- **LED_BUILTIN** // O microcontrolador ATmega na maioria das placas Arduino possui resistores pull-up internos (resistores conectados a alimentação internamente) que você pode acessar. Se você preferir usar estes em vez de resistores de pull-up externos, você pode usar o argumento INPUT_PULLUP na função pinMode().
- **true** // true é frequentemente dito ser definido como 1, o que está correto, porém true possui uma definição mais ampla. Qualquer inteiro que não seja zero é true, em um sentido booleano. Então -1, 2 e -200 são todos definidos como true, também, em um sentido booleano.
- **false** // false é o mais fácil dos dois de ser definido. Sendo definido apenas como 0 (zero).
- **Constantes de Ponto Flutuante** // Similar as constantes inteiras, constantes de ponto flutuante são usadas para tornar o código mais legível. Constantes de ponto flutuante são trocadas em tempo de compilação para o valor calculado para a expressão.

  Código de Exemplo

  n = 0.005; // 0.005 é uma constante de ponto flutuante

- **Constantes Inteiras** // Constantes Inteiras são números usados diretamente em um sketch, como 123. Por padrão, esses números são tratados como int mas você pode mudar isso com os modificadores U e L (ver abaixo).

Normalmente, constantes inteiras são tratadas como inteiros na base 10 (decimais), mas notações especiais (modificadores) podem ser usadas para escrever-se números em outras bases.

BASE	EXEMPLO	MODIFICADOR	COMENTÁRIO
10 (decimal)	123	nenhum	
2 (binário)	B1111011	prefixo 'B'	funciona apenas com valores 8-bit (0 a 255) caracteres 0 e 1 válidos
8 (octal)	0173	prefixo "0"	caracteres 0-7 válidos
16 (hexadecimal)	0x7B	prefixo "0x"	caracteres 0-9, A-F, a-f válidos

Decimal (base 10)

Essa é a matemática de senso comum a qual você já está acostumado. Constantes sem prefixos são assumidas como valores decimais.

Código de Exemplo:

```
n = 101; // o mesmo que 101 decimal   ((1 * 10^2) + (0 * 10^1) + 1)
```

Binário (base 2)

Apenas os caracteres 0 e 1 são válidos.

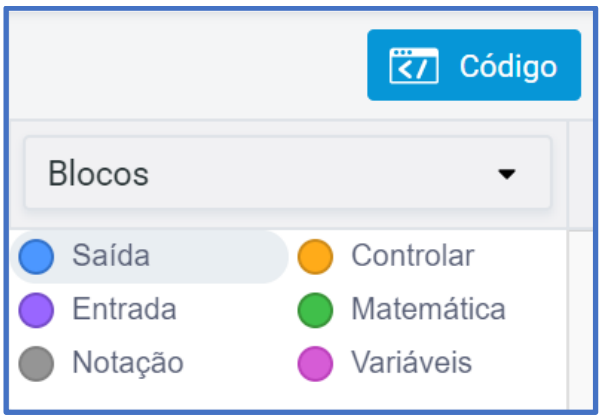
2.6. Blocos

Os blocos são peças em forma de quebra-cabeça que são usados para criar o código para o projeto. Os blocos conectam-se uns aos outros conforme o seu tipo e juntos formam um conjunto de instruções, de modo que cada bloco tenha sua própria forma e um conector de forma apropriada possa ser inserido para evitar erros de sintaxe.

Para construir um programa, deve-se arrastar blocos para o editor de código criando um conjunto de instruções que serão executadas no simulador ou no Arduino quando gravado nele.

Os blocos estão organizados em categorias, algumas básicas das linguagens de programação e outras específicas para a programação no Arduino.

Os blocos organizados no editor de código são automaticamente traduzidos para código fonte na linguagem de programação. O código traduzido aparece na aba “texto” sempre estruturado a partir das duas funções básicas: setup() e loop().



3. Exemplos de Programação

3.1. Primeira atividade:

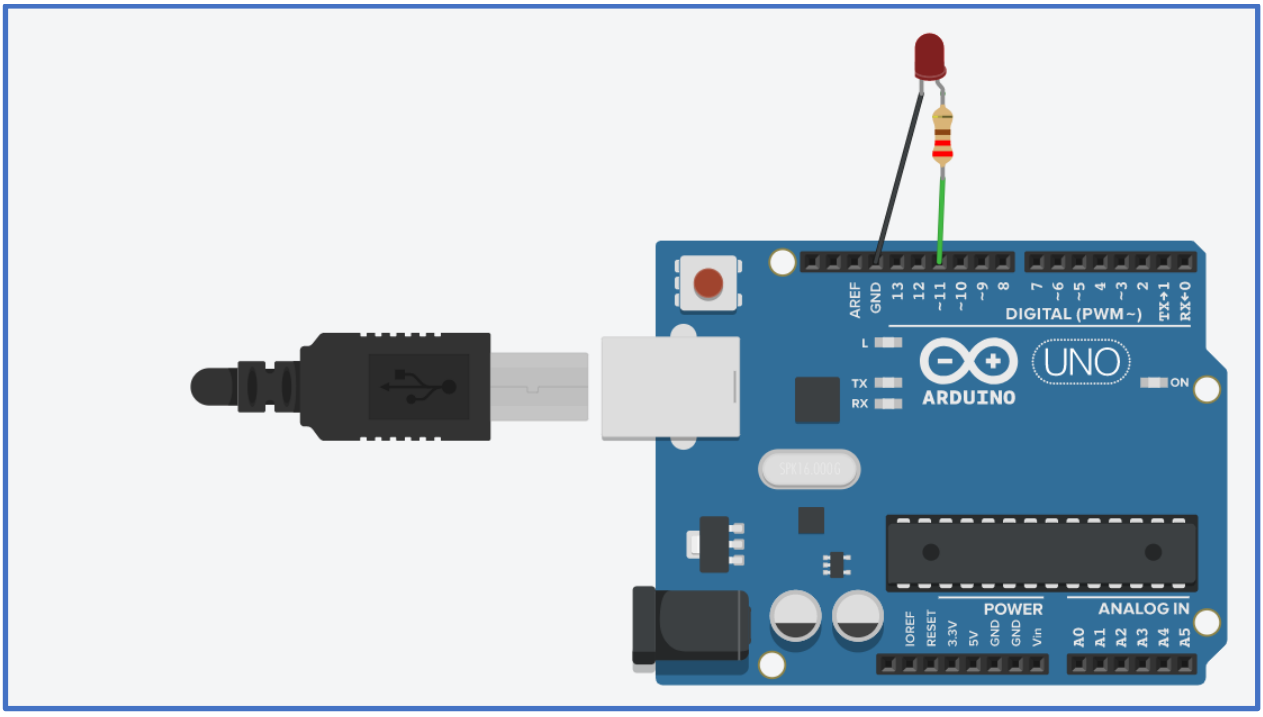
Aula 01 pisca Led

Link da atividade: https://www.tinkercad.com/things/hu36y1k3x2L-aula-01-pisca-led

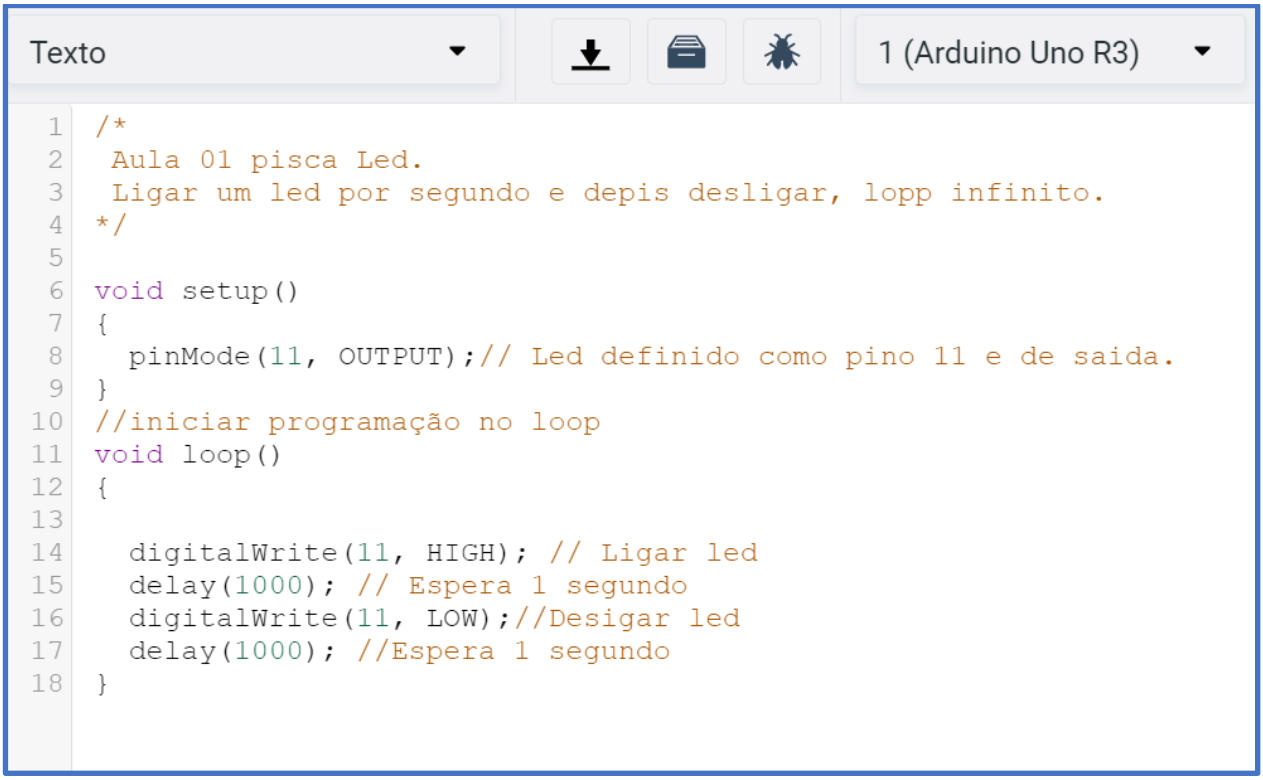
Material a ser utilizado:

Quantidades	Componentes
1	Arduino Uno R3
1	220 Ω Resistor
1	Vermelho LED
2	Jumper Macho/Macho

Montagem:



Programação em texto:




```
/*
 Aula 01 pisca Led.
 Ligar um led por segundo e depis desligar, lopp infinito.
*/

void setup()
{
  pinMode(11, OUTPUT);// Led definido como pino 11 e de saida.
}
//iniciar programação no loop
void loop()
{

  digitalWrite(11, HIGH); // Ligar led
  delay(1000); // Espera 1 segundo
  digitalWrite(11, LOW);//Desigar led
  delay(1000); //Espera 1 segundo
}
```

3.2. Segunda atividade:

Faça um semáforo com as matérias sugeridos abaixo.

Aula 02 Semáforo com Led

Seguindo o exemplo da atividade 01 em que aprendemos a fazer um pisca led.

Agora acesse o link no Tinkercad e faça você mesmo seu semáforo

Link da atividade: https://www.tinkercad.com/things/0FOonuGzJwe-aula-02-semaforo

Material a ser utilizado:

Quantidades	Componentes
1	Arduino Uno R3
3	220 Ω Resistor
1	LED Verde
1	LED Vermelho
1	LED Amarelo
1	Placa Protoboard
7	Jumper Macho/Macho

3.3. Terceira atividade:

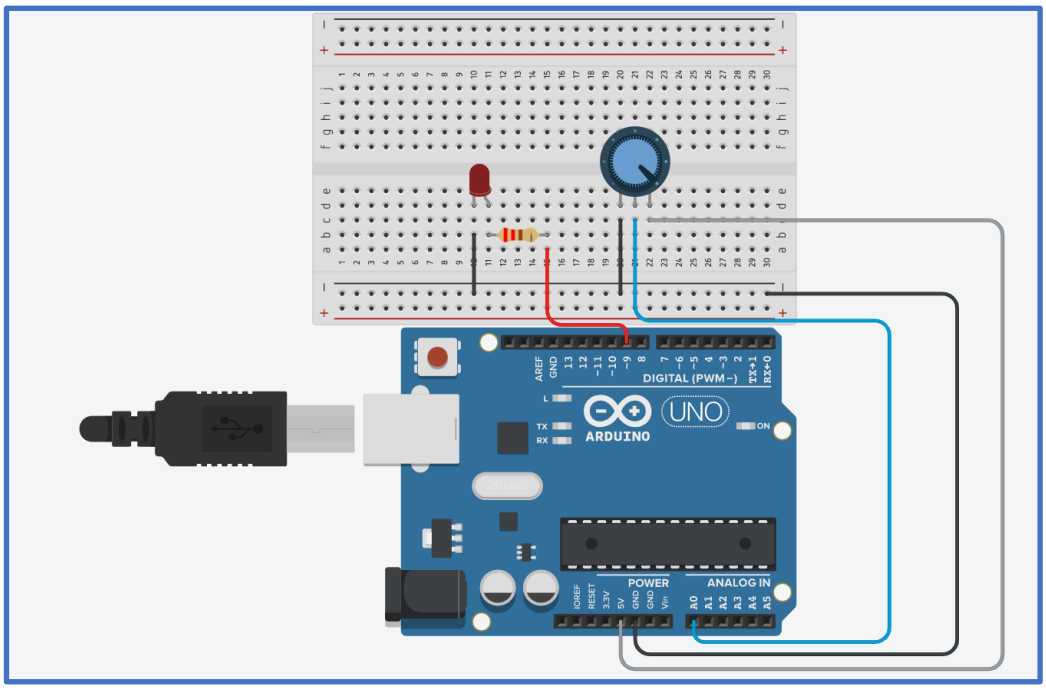
Aula 03 Controlando a luz do LED com Potenciômetro

Descrição: nessa atividade iremos controlar a luz do LED com o potenciômetro

Material a ser utilizado:

Quantidades	Componentes
1	Arduino Uno R3
3	250 kΩ, Potenciômetro
1	LED Vermelho
1	Placa Protoboard
6	Jumper Macho/Macho

Montagem:



Código gerado:

Texto

1 (Arduino Uno R3)

```
/*
Aula 03 Controlando a luz do led com Potenciômetro
*/

#define potPin 0  // Define o pino analógico em que o potenciômetro vai ser conectado
#define ledPin 9   // Define o pino que terá um LED e um resistência ligada ao terra

int valPot = 0; //Variável que armazena o valor da leitura do potenciômetro

void setup() {

pinMode(ledPin,OUTPUT); // Configura o pino do LED como saída

}

void loop() {

valPot =  analogRead(potPin); //Faz a leitura analógica do pino em que o potenciômetro esta ligado
valPot = map(valPot,0,1023,0,255); //Utilizando a função map() para transformar uma escala de 0-1023 em uma escala 0 a 255
analogWrite(ledPin,valPot ); // Aciona o LED proporcionalmente ao valor da leitura analógica

}
```

Monitor serial

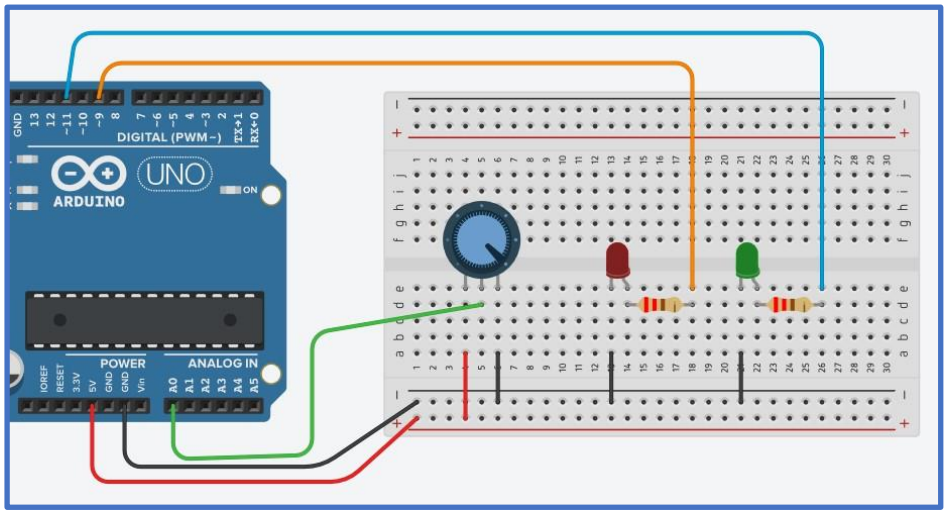
3.4. Quarta atividade:

Potenciômetro controlando os LEDs

Material a ser utilizado:

Quantidades	Componentes
1	Arduino Uno R3
3	250 kΩ, Potenciômetro
2	Led s
1	Placa Protoboard
7	Jumper Macho/Macho
2	Resistor 330 ohms

Montagem:



Código de bloco gerado:

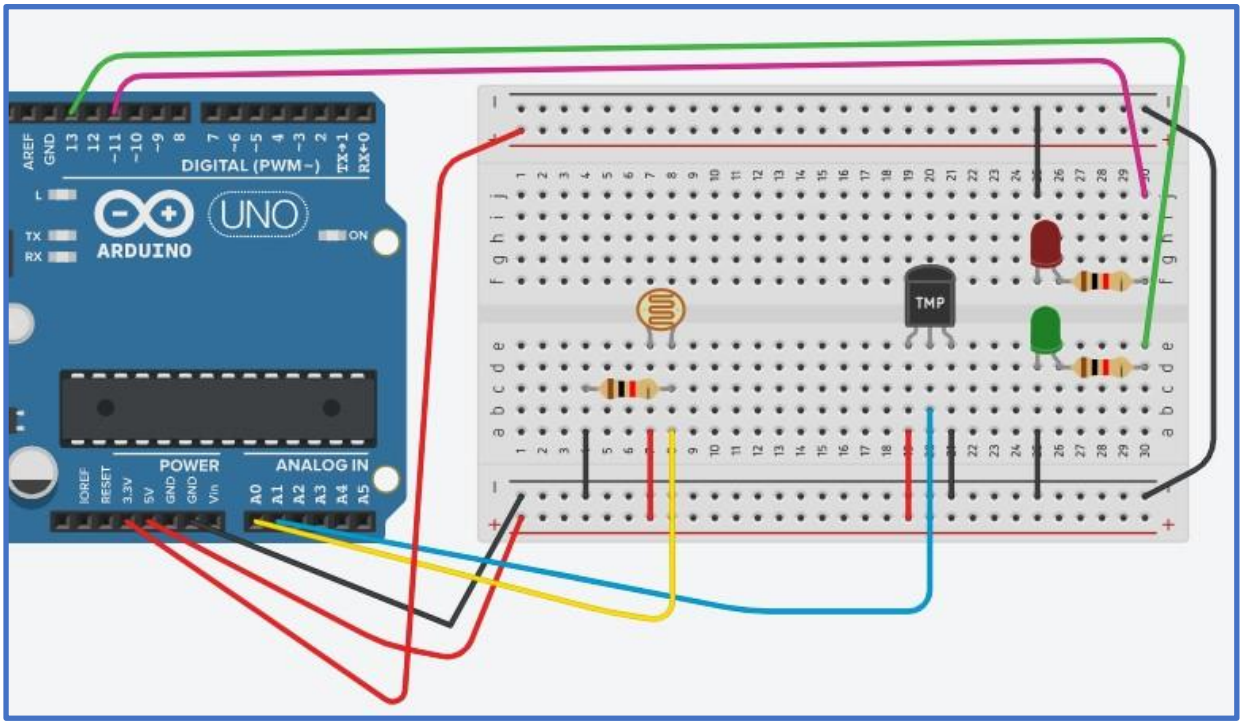
3.5. Quinta atividade:

Atividade usando Sensores

Material a ser utilizado:

Quantidades	Componentes
1	Protoboard
1	Arduino Uno
1	Sensor de Luminosidade
1	Sensor de Temperatura
14	Jumpers
2	LED
3	Resistor 330 ohms

Montagem:



Código gerado:

```
int lightSensor = A0;
int tempSensor = A1;

void setup() //É executado uma vez no início
da operacao
{
 pinMode(tempSensor, INPUT);
 pinMode(lightSensor, INPUT);
 Serial.begin(9600);
}

void loop()
{
 //imprimir uma linha para a leitura
 int lightReading = analogRead(lightSensor);
 Serial.print("Leitura Luz: ");
 Serial.println(lightReading);

 int tempReading = analogRead(tempSensor);
 // valor entre 0 e 750
 //Usando entrada de 5v
 float voltage = tempReading * 5.0;

// Dividindo por 1024
voltage /= 1024.0;

//Convertendo em intervalo de 10mv
float tempC = (voltage - 0.5) * 100;

// testa a temperatura e controla lampadas
if (tempC < 30.){
   digitalWrite(13,HIGH);    //led verde
} else {
   digitalWrite(13,LOW);
}
if (tempC > 50.){
   digitalWrite(11,HIGH);    //led vermelho
} else {

digitalWrite(11,LOW);
}

Serial.print(tempC);
Serial.println(" ºC");   //imprimindo o valor

 delay(100); //Usando intervalo de tempo
}
```

3.6. Sexta atividade

Atividade usando LED-RGB acionado por potenciômetro

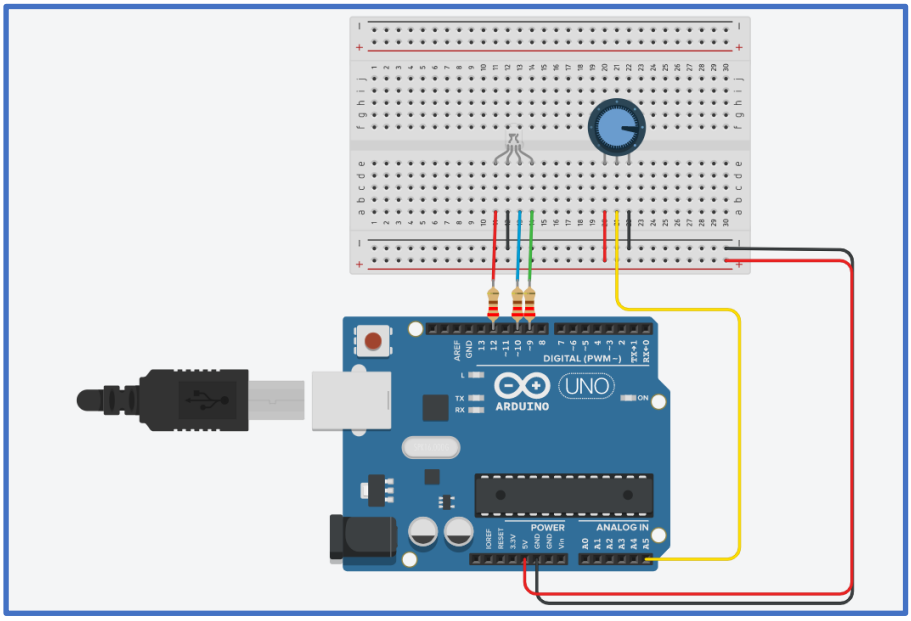
Agora acesse o link no Tinkercad e veja como fazer a montagem

Link da atividade: https://www.tinkercad.com/things/hEpR3ZGptNO

Material a ser utilizado:

Quantidades	Componentes
1	Arduino Uno R3
1	Led RGB
1	Placa Protoboard
9	Jumper Macho/Macho
3	Resistor 220 ohms
1	Potenciômetro

Montagem:



Código gerado:

```
int R = 12;
int B = 10;
int G = 9;
int potenciometro = 5;

void setup()
{
  Serial.begin(9600);
  pinMode(R, OUTPUT);
  pinMode(B, OUTPUT);
  pinMode(G, OUTPUT);
}

//Funções para executar o brilho
selecionado
void redFunc(){         //vermelho
  digitalWrite(G,LOW);
  digitalWrite(B,LOW);
  digitalWrite(R,HIGH);
}
void blueFunc(){         //azul
  digitalWrite(G,LOW);
  digitalWrite(B,HIGH);
  digitalWrite(R,LOW);
}
void greenFunc(){         //verde
  digitalWrite(G,HIGH);
  digitalWrite(B,LOW);
  digitalWrite(R,LOW);
}
void yellowFunc(){         //amarelo
  digitalWrite(G,50);
  digitalWrite(B,0);
  digitalWrite(R,255);
}
void purpleFunc(){         //roxo
  digitalWrite(G,0);
  digitalWrite(B,207);
  digitalWrite(R,255);
}
void whiteFunc(){         //branco
  digitalWrite(G,HIGH);
  digitalWrite(B,HIGH);
  digitalWrite(R,HIGH);

}
// FINAL DAS FUNÇÕES

void loop()
{
  //potenciometro simula a leitura da
temperatura
  float sinal;
  sinal = analogRead(potenciometro);
  Serial.println(sinal);
  if(sinal>=0 && sinal <=150)
    whiteFunc();
  else if(sinal>150 && sinal <=300)
                    blueFunc();
    else if(sinal>300 && sinal<=450)
                    greenFunc();
      else if(sinal>450 && sinal<=600)
                     yellowFunc();
       else if(sinal>600 && sinal<=750)
                    redFunc();
              else whiteFunc();
}
```

3.7. Sétima atividade:

Atividade usando LEDs controlado por um botão

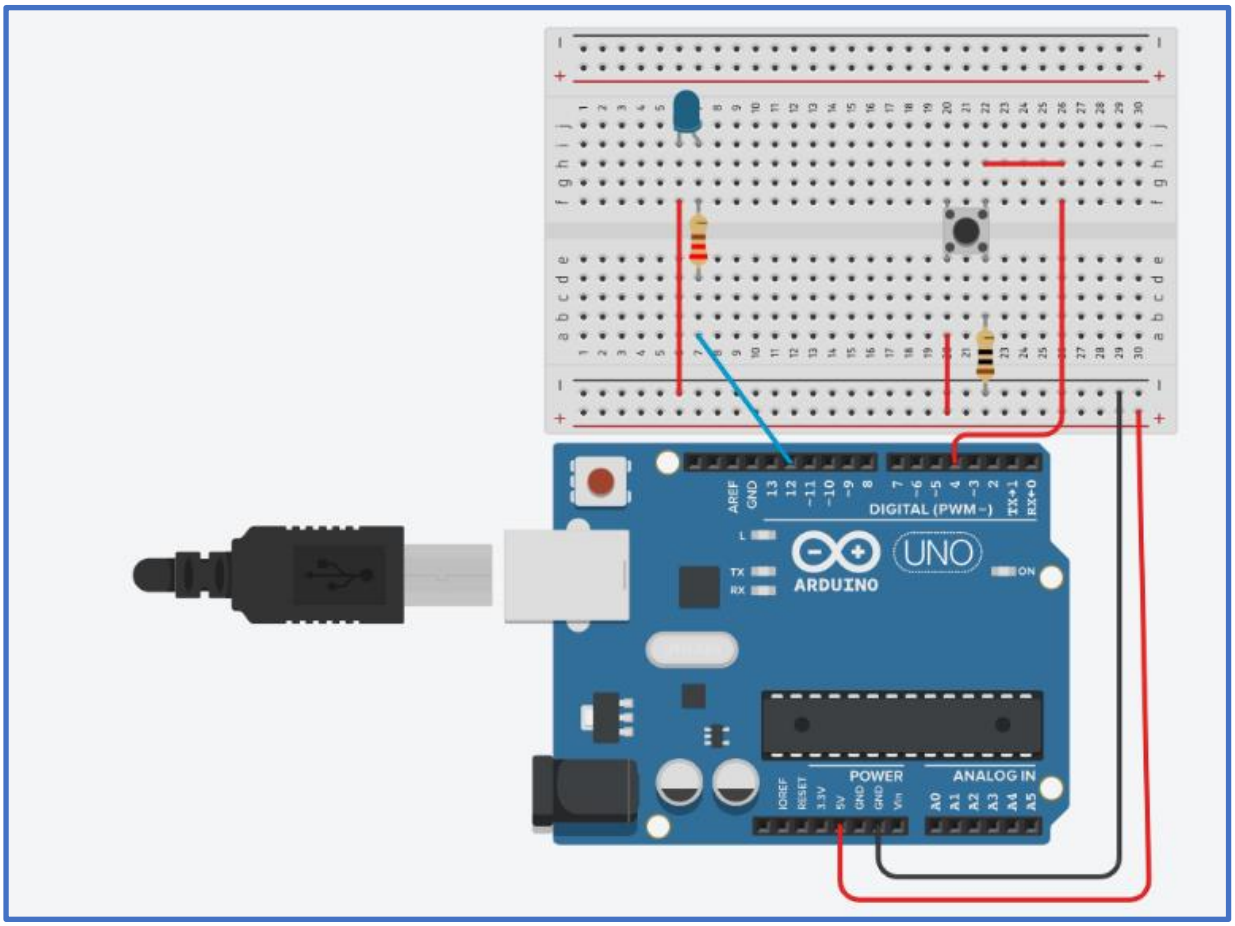
Agora acesse o link no Tinkercad e veja como fazer a montagem

Link da atividade: https://www.tinkercad.com/things/8Hh6QXX1P8X

Material a ser utilizado:

Quantidades	Componentes
1	Arduino Uno R3
5	Led
1	Placa Protoboard
6	Jumper Macho/Macho
1	220 Ω Resistor
1	10 Ω Resistor
1	Botão

Montagem:



Código gerado:

```
void setup()
{
  pinMode(4, INPUT);
  pinMode(12, OUTPUT);
}

void loop()
{
  // Se o botão estiver apertado:
  if (digitalRead(4) == HIGH) {
    // Ligue o LED.
    digitalWrite(12, HIGH);
    // Caso contrário,
  } else {
    // Desligue o LED.
    digitalWrite(12, LOW);
  }
  delay(10); // espera 10 milissegundos
}
```

Referências

KENSHIMA, Gedeane. **Nas linhas do Arduino**. São Paulo: Novatec Editora, 2020.

MCROBERTS, Michael. **Arduino Básico**, tradução Rafael Zanolli - São Paulo, Novatec Editora, 2011.

INSTITUTO FEDERAL

Triângulo Mineiro

Campus Uberlândia Centro

Grupo de Pesquisa em Educação, Tecnologia e Ciências.