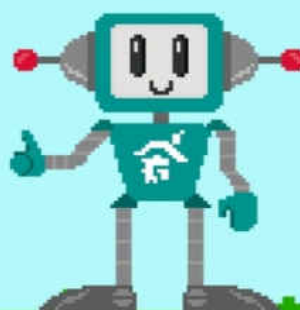


EDIÇÃO DE ANIVERSÁRIO
CASA DA ROBÓTICA

♥♥♥♥♥♥
ANOS

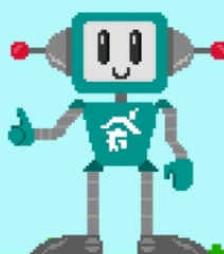
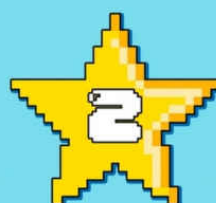
CRIANDO JOGOS COM ARDUINO



EDIÇÃO DE ANIVERSÁRIO
CASA DA ROBÓTICA



CRIANDO JOGOS COM ARDUINO



OceanofPDF.com

Copyright © 2021 da WL Componentes Eletrônicos Ltda .

Todos os direitos reservados e protegidos pela Lei 9.610 de 19/02/1998. É proibida a reprodução desta obra, mesmo parcial, por qualquer processo, sem prévia autorização, por escrito dos autores.

Editores: Luan Silva Santana e Wellington Assunção Azevedo

Produção: Carol Correia Viana

Revisão: Kleber Rocha Bastos

Diagramação: Carol Correia Viana

Ilustração de capa: Thiago Sá

Colaboração: Carlos Dyorgenes Santana e Anielle Barbosa Menezes

Dados Internacionais de Catalogação na Publicação (CIP) (Câmara Brasileira do Livro, SP, Brasil)

Viana, Carol Correia

Criando jogos com arduino 2 [livro eletrônico] / Carol Correia Viana. -- 1. ed. -- Vitória da Conquista, BA : Casa da Robótica, 2021.

Outros colaboradores

Bibliografia ISBN 978-65-992390-4-5

1. Arduino (Linguagem de computador) 2. Ciência da Computação 3. Robótica – Estudo e ensino I. Título

21-75622

CDD-372.358

Índices para catálogo sistemático:

1. Robótica: Estudo e ensino 372.358

Aline Grazielle Benitez - Bibliotecária - CRB-1/3129

OceanofPDF.com

APRESENTAÇÃO

Obrigado por adquirir nosso e-book: Criando Jogos com Arduino 2 .

É com grande alegria que celebramos nosso sexto aniversário e, para comemorar este momento especial, convidamos você para se divertir com a gente na criação de jogos com Arduino.

Este e-book foi carinhosamente preparado para você, que assim como nós, é apaixonado por eletrônica, programação, games e, é claro, Arduino.

O Arduino é uma plataforma eletrônica de software e hardware aberto. Sua versatilidade chama atenção de professores, alunos, *hobbistas* e entusiastas do movimento *Maker* .

O desenvolvimento de games pode ser o ponto de partida para quem deseja aprender Arduino e uma diversão para aqueles que já sabem utilizar a plataforma.

Pensando nisso, elaboramos 7 jogos com Arduino para você recriar. Os jogos são detalhados passo a passo, apresentando o esquemático do circuito para montagem e a programação necessária.

Como forma de facilitar o entendimento das ligações, o modo de programação e permitir a simulação disponibilizamos os jogos no Tinkercad, ferramenta online gratuita que permite a simulação de circuitos elétricos e programação.

Para criação de um dos jogos propostos utilizamos o Pictoblox, software de programação em blocos baseado em Scratch que permite a interação com Arduino. Com isso, vamos introduzir a possibilidade de unir

o hardware com o software, o que torna o aprendizado muito mais prazeroso.

Esperamos que você aproveite este e-book com entusiasmo e que ele auxilie a sua jornada de estudos. Se divirta e boa leitura!

Um grande abraço,

Equipe Casa da Robótica

OceanofPDF.com

OceanofPDF.com

SUMÁRIO

1. INTRODUÇÃO

2. PLATAFORMA ARDUINO

O QUE É O ARDUINO?

EXPLORANDO UMA PLACA UNO SMD R3 ATMEGA328

PRIMEIROS PASSOS

EXPLORANDO O ARDUINO IDE

3. FUNDAMENTOS DE PROGRAMAÇÃO

ESTRUTURAS BÁSICA DE UM SKETCH

VARIÁVEIS

FUNÇÕES

BIBLIOTECAS

OPERADORES

ESTRUTURA DE CONTROLE DE FLUXO

4. CRIANDO JOGOS COM ARDUINO

PIANO ELETRÔNICO

JOGO DE DADOS COM LEDS – DADO ELETRÔNICO

TORTA NA CARA – PASSA OU REPASSA

DESARME A BOMBA

BUZZ WIRE COM CONTADOR

PIANO HERO

SUPER CDRBOT

5. CONSIDERAÇÕES FINAIS

REFERÊNCIAS

SOBRE NÓS

OceanofPDF.com

OceanofPDF.com

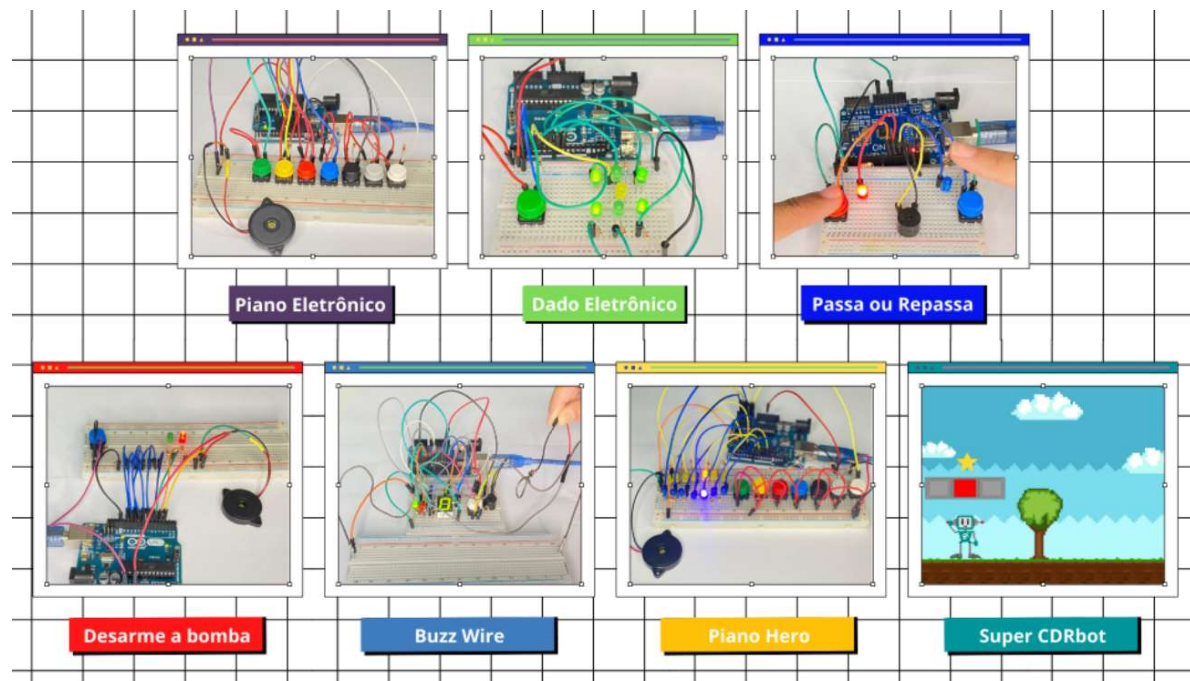
1. INTRODUÇÃO

A versatilidade do Arduino garante que ele seja utilizado em uma variedade de projetos como na construção de robôs, braços articulados, monitoramento de temperatura e, até mesmo, na criação de jogos.

A plataforma Arduino permite o desenvolvimento de uma infinidade de jogos. Você já pensou em desenvolver seu próprio jogo? Com o Arduino isso é possível.

Neste e-book elaboramos 7 jogos com Arduino para você recriar, ilustrados na Figura 1 .

Figura 1 – Jogos desenvolvidos com Arduino.



O desenvolvimento de jogos pode ser o ponto de partida para quem deseja aprender Arduino e uma diversão para aqueles que já sabem utilizar a plataforma.

Então, venha aprender e se divertir com a gente!

Os jogos propostos para você recriar são:

- Piano Eletrônico;
- Jogo de dados com LEDs – Dado Eletrônico;
- Torta na Cara – Passa ou Repassa;
- Desarme a bomba;
- Buzz Wire com contador;
- Piano Hero;
- Super Robô World.

Os jogos são detalhados passo a passo, com lista de materiais, esquemático de ligação dos componentes e a programação necessária.

Para facilitar o entendimento das ligações, o modo de programação e permitir a simulação, disponibilizamos todos os jogos no Tinkercad.

Para criação do jogo Super Robô Hero utilizamos o Pictoblox, software de programação em blocos baseado em Scratch que permite a interação com Arduino.

Para desenvolvimento dos jogos você precisará de alguns componentes eletrônicos básicos, além do Arduino. A Tabela 1 apresenta a lista de materiais.

LISTA DE MATERIAIS
Placa UNO SMD compatível com Arduino UNO

Cabo USB
Protoboard
Botões do tipo push button
LEDs difusos de 5 mm
LED flash rápido
Resistores de 10 kΩ
Resistores de 220 Ω
Buzzer passivo
Buzzer ativo
Display de 7 segmentos ânodo comum
Jumpers macho-macho
Arame de metal ou fio desencapado de 0,8 mm

Este e-book encontra-se estruturado em 5 capítulos, sendo este o primeiro deles.

No segundo capítulo você encontrará alguns conceitos importantes para iniciar seus estudos. Para isso, expomos um pouco sobre a plataforma Arduino e o microcontrolador UNO R3 Atmega328, compatível ao projeto Arduino UNO.

O Capítulo 3 foi preparado para apresentar a você os principais fundamentos da linguagem de programação utilizada pelo Arduino, abordando sobre a declaração e tipos de variáveis, funções, bibliotecas, operadores e estrutura de condição e repetição.

O Capítulo 4 apresenta o passo a passo para criação dos jogos. Também são listados os materiais, a forma de montagem do circuito e a

programação necessária para o funcionamento dos jogos.

O Capítulo 5 destina-se às considerações finais deste e-book. No fim deste e-book, você encontrará uma lista de referências caso queira se aprofundar nas pesquisas sobre Arduino e sobre os projetos desenvolvidos aqui.

Você poderá conhecer um pouco sobre a Equipe Casa da Robótica na seção Sobre nós.

2. PLATAFORMA ARDUINO

O QUE É O ARDUINO?

O Arduino é uma plataforma eletrônica de código aberto baseada em software e hardware de fácil utilização, sendo ideal para iniciantes e para qualquer pessoa que deseja construir projetos eletrônicos.

As placas Arduino permitem a conexão de circuitos eletrônicos aos seus terminais, o que possibilita a leitura de entradas – luminosidade em um sensor, o acionamento de um botão ou uma mensagem SMS, e a transformação dessas informações em uma saída para controle de algum dispositivo, ligando um LED, ativando um motor ou enviando uma mensagem, por exemplo.

As placas Arduino podem ser conectadas ao computador por meio do barramento serial universal (USB), o que permite a sua utilização como placa de interface e controle de dispositivos por meio do seu computador.

A plataforma Arduino oferece uma série de vantagens em relação a outras plataformas, o que a tornou popular entre professores, alunos, amadores e projetistas, tais como:

- Possuir ambiente multiplataforma. O Arduino pode ser programado nos principais sistemas operacionais comercializáveis;
- Contar com uma IDE de programação própria;
- Poder ser programado utilizando um cabo USB;
- Possuir hardware e software de fonte aberta;
- Ter sido desenvolvido em um ambiente educacional.

Diante da sua popularização, a plataforma Arduino cresceu e atualmente conta com diversas versões de mercado. A Figura 2 ilustra algumas versões da placa Arduino.

Figura 2 - Algumas versões da placa Arduino



Além disso, existe uma série de placas compatíveis com o projeto Arduino. Uma vez que seu hardware é aberto, a replica destas placas são permitidas e possuem as mesmas características, pinagens e forma de uso.

Existem placas Arduino bem pequenas (Nano, micro, mini), de tamanho médio ou tradicional (Uno, Duemilanove, Leonardo), e as placas de maiores dimensões (Mega, Due).

Diante de tanta variedade, você deve estar se perguntando: Qual placa devo usar no meu projeto?

A escolha da versão ideal vai depender das necessidades de seu projeto, mas recomendamos:

- Placas de Arduino pequenas: Para projetos que precisam ser leves e ocupar pouco espaço;
- Placas de Arduino tamanho médio e tradicional: Para projetos de tamanho padrão, como robôs, interfaces homem-máquina, central de monitoramento, entre outros;
- Placas de Arduino maiores dimensões: Para projetos que demandem maior memória e número de portas de entrada e saída.

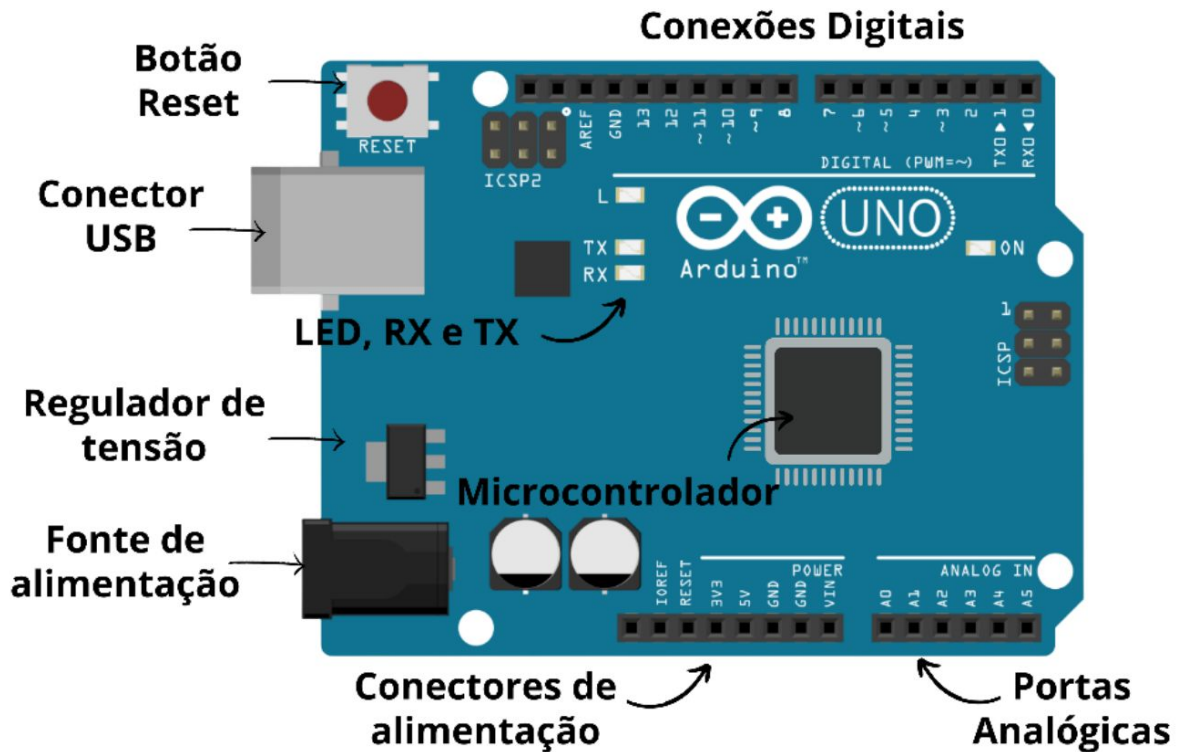
EXPLORANDO UMA PLACA UNO SMD R3 ATMEGA328

Conhecer os elementos que compõem o microcontrolador Arduino é de suma importância antes de iniciar os projetos. Nessa seção, vamos conhecer a placa UNO SMD R3 Atmega328.

A placa UNO (Figura 3) é considerada a melhor placa para quem deseja iniciar os estudos de eletrônica e programação, por ser a mais utilizada e documentada da família Arduino. Por essas razões, os exemplos deste livro serão baseados nela.

Vamos explorar a placa UNO SMD R3 para nos familiarizar com seus componentes.

Figura 3 - Placa UNO SMD R3 ATMEGA238.



Fonte de alimentação

O circuito interno da placa UNO deve ser alimentado com uma tensão contínua de 5 V. Você pode alimentá-lo conectando-o a uma porta USB do computador, que fornecerá a alimentação e a comunicação de dados. Você também pode usar uma fonte de alimentação externa, que forneça uma saída contínua entre 7 e 12 V, por meio da utilização de um plug P4 ou o pino Vin.

Regulador de tensão

Na placa UNO, o regulador de tensão tem como finalidade transformar qualquer tensão (entre 7 V e 12 V) que esteja sendo fornecida pelo conector de alimentação externa em uma tensão contínua de 5 V.

Conectores de alimentação elétrica

Os conectores de alimentação elétrica fornecem energia para dispositivos externos e são constituídos pelos pinos:

- Reset que possui a mesma função do botão Reset;
- 3,3 V e 5 V que fornecem tensão de 3,3 e 5 V, respectivamente;
- GND fornece potencial de terra aos dispositivos externos;
- Vin fornece ao dispositivo externo a mesma tensão que está sendo recebida pelo pino de alimentação externa.

Entradas analógicas

A placa UNO possui 6 portas analógicas que estão indicadas como Analog In, de A0 a A5. Esses pinos são dedicados a receber valores de grandezas analógicas, por exemplo, a tensão de um sensor. As grandezas analógicas variam continuamente no tempo dentro de uma faixa de valores.

Conexões digitais

A placa UNO possui 14 portas digitais que estão indicadas como Digital, de 0 a 13. Estas portas podem ser utilizadas para receber ou enviar dados de grandezas digitais. Ao contrário das grandezas analógicas, as grandezas digitais não variam continuamente no tempo, mas sim entre valores definidos (0 ou 1, ligado ou desligado, sim ou não, 0 V ou 5 V).

Estes pinos digitais operam em 5V e corrente máxima de 40 mA. Além disso, alguns deles possuem funções especiais, como:

- Pinos 3, 5, 6, 9, 10 e 11 podem ser usados como saídas PWM, simulando uma saída analógica;

- Pinos 0 e 1 (RX e TX) podem ser utilizados para comunicação serial;
- Pinos 2 e 3 podem ser configurados para interrupção externa.

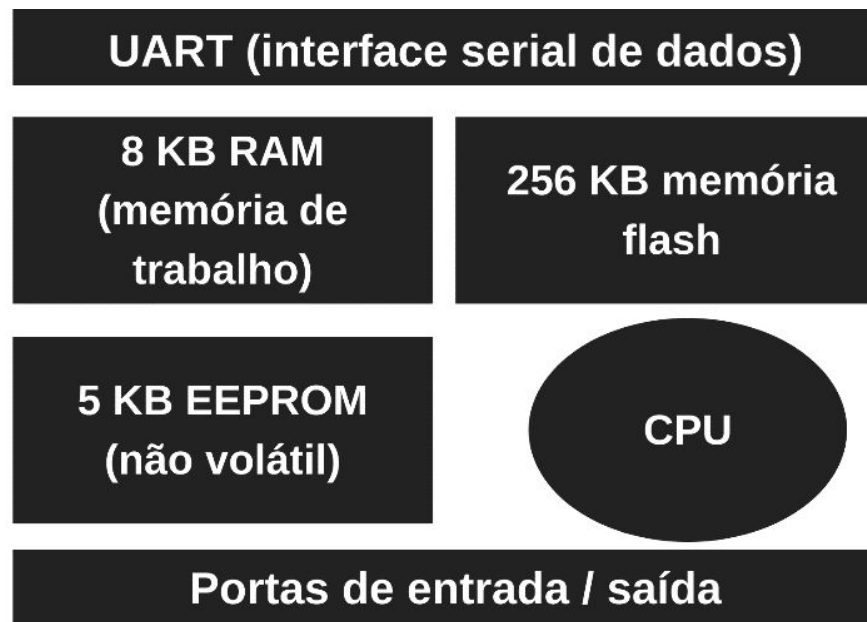
Microcontrolador

O microcontrolador utilizado na placa UNO é o ATmega328, um pequeno chip de 28 pinos que se encontra no centro da placa e é considerado o cérebro desse dispositivo. Esse único chip é um pequeno computador, contendo memória, processador e toda eletrônica necessária aos pinos de entrada e saída.

É no microcontrolador que tudo acontece, é nele que fica gravado o código desenvolvido para execução. O microcontrolador permite que a placa Uno funcione de forma autônoma, em outras palavras, uma vez transferido o código, não existe mais a necessidade de comunicação com o computador. Um fato que deve ser lembrado é que ao gravar um código, o anterior é descartado, ficando apenas o último código gravado.

Algumas características do microcontrolador ATmega328 encontram-se detalhadas na Figura 4 .

Figura 4 - Diagrama de blocos do ATmega328.



Fonte: Adaptado de Monk (2017).

Botão Reset

O botão Reset tem como única função reinicializar a placa microcontroladora.

Outros componentes

Além dos componentes citados, a placa UNO também conta com: um oscilador a cristal, capaz de realizar 16 milhões de ciclos ou oscilações por segundo; um conector serial de programação, outro meio de programar a placa UNO; e um chip de interface USB, que converte os níveis de sinal usados pelo padrão USB em níveis que podem ser usados pela placa UNO.

PRIMEIROS PASSOS

Para que a placa UNO execute qualquer ação, você precisará escrever um código ou Sketch em linguagem C/C++ utilizando gratuitamente o software Arduino IDE, que se encontra disponível na versão online e offline.

O Arduino Web Editor é a interface de desenvolvimento online do Arduino, com ele é possível codificar, salvar os esboços na nuvem, fazer backup e enviar o código feito para qualquer placa compatível com o Arduino a partir do navegador de internet. O Arduino Web Editor encontra-se disponível no link: <https://create.arduino.cc/editor>.

O Arduino IDE é a versão offline desta ferramenta de desenvolvimento e pode ser executado no Windows, Mac OS X e Linux. O download do Arduino IDE encontra-se disponível no link: <https://www.arduino.cc/en/Main/Software#download>.

Download e Instalação do Arduino IDE no Windows

Em nosso blog, você encontrará o passo a passo para instalar o Arduino IDE no computador Windows. Acesse:

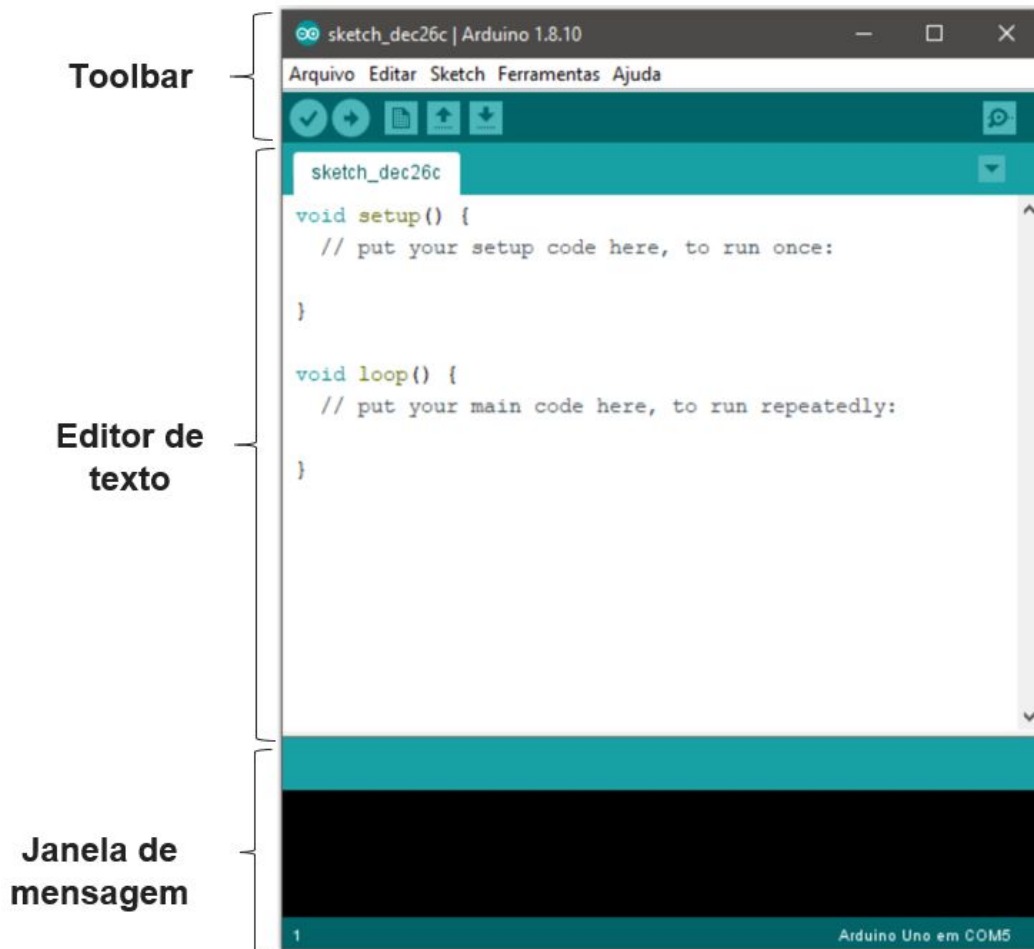
www.blogdarobotica.com/instalando-o-arduino-ide

EXPLORANDO O ARDUINO IDE

Ao abrir o Arduino IDE, você verá uma tela semelhante à Figura 5 . Caso esteja utilizando o Linux ou Mac OS X, pode haver pequenas

diferenças, mas o IDE é basicamente o mesmo para todos os sistemas operacionais.

Figura 5 - Visual do Arduino IDE.



O Arduino IDE pode ser dividido em três partes: A Toolbar no início da tela, o editor de texto no centro e a janela de mensagens na base.

No topo da Toolbar há uma barra de menus contendo comandos comuns com os itens: Arquivo, Editar, Sketch, Ferramentas e Ajuda. Os comandos e funções disponíveis na barra de ferramenta podem ser consultados ao acessar o comando Ajuda > Ambiente.

Ainda na Toolbar, encontra-se os botões de atalho, que fornecem acesso rápido às funções mais utilizadas. A seguir são mostrados os ícones

e o detalhamento de suas funções.

-  Verificar : Analisa se há erros em seu código
-  Upload : Compila seu código e o envia para a placa microcontroladora
-  Novo : Cria um novo Sketch
-  Abrir : Usado para abrir um Sketch existente
-  Salvar : Salva o seu Sketch atual
-  Monitor serial : Exibe os dados seriais enviados pela placa microcontroladora

O editor de texto é o campo destinado a escrita dos códigos. Os códigos escritos usando Arduino ou placas compatíveis são conhecidos como Sketches e são salvos com a extensão de *arquivo .ino* . Este editor de texto tem características de um editor tradicional, contendo funções de cortar, copiar, colar, selecionar tudo, entre outras.

A janela de mensagem fornece mensagens de feedback ao salvar e exportar arquivos, bem como exibe informações de erros no código ou ao compilar.

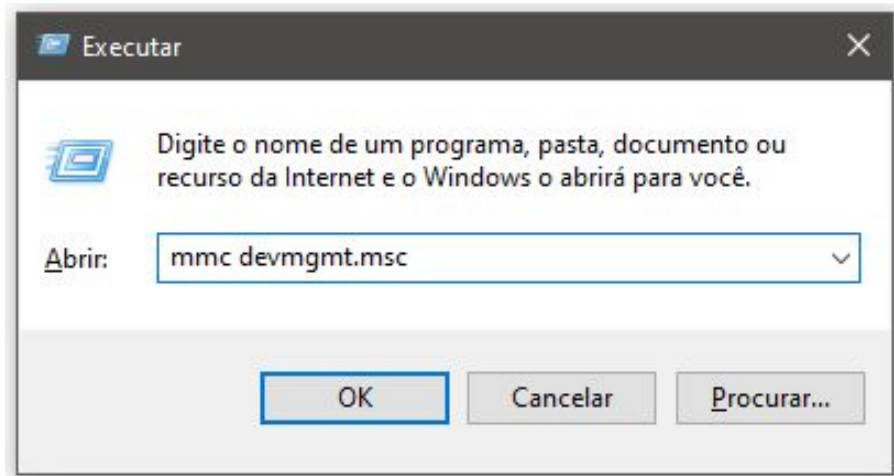
Conectando a placa UNO ao computador

Agora que já conhecemos a placa UNO e sua interface de desenvolvimento, vamos conectá-la ao computador. Esta conexão é realizada por meio de um cabo USB do tipo A para o tipo B.

Conecte sua placa UNO ao seu computador através da USB. Para verificar se o seu computador Windows está identificando a placa, vamos realizar um teste acessando o gerenciador de dispositivos. Um atalho para

chegar nesse painel é pressionar as teclas “ *Windows + R* ” e, em seguida, digitar “ *mmc devmgmt.msc* ” sem as aspas no menu Executar, conforme a Figura 6 .

Figura 6 - Atalho para acessar o gerenciador de dispositivos.

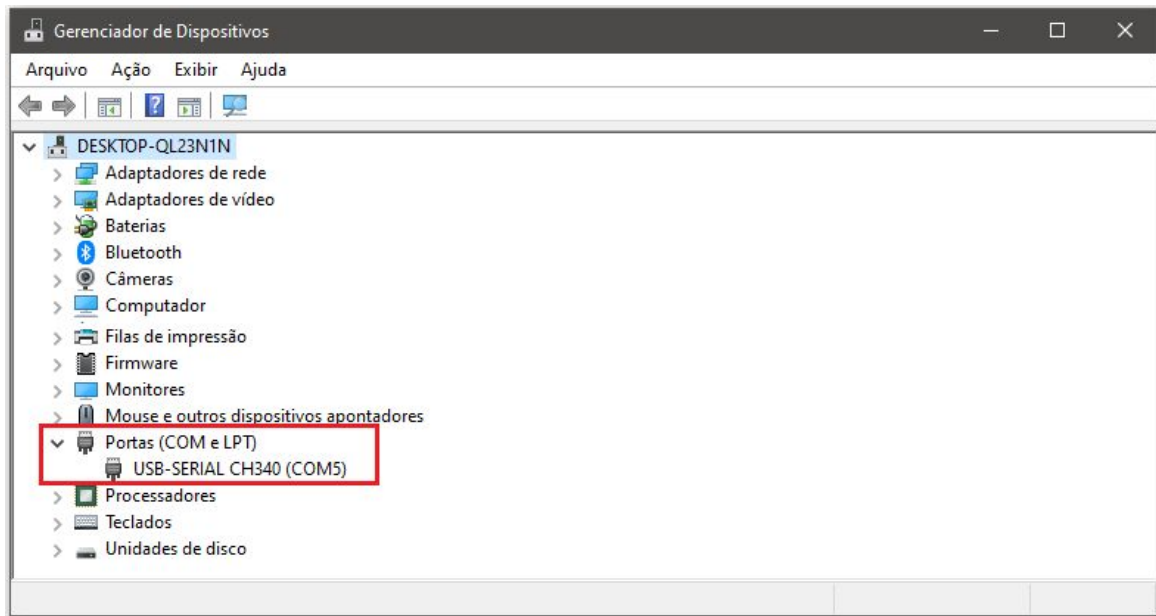


Após digitar esse comando, clique em “OK”. Em seguida, será aberta a tela do Gerenciador de Dispositivos, como na Figura 7 .

Para verificar se o driver da placa UNO foi reconhecido navegue até a opção Portas (COM e LPT) e expanda clicando na setinha do lado do nome.

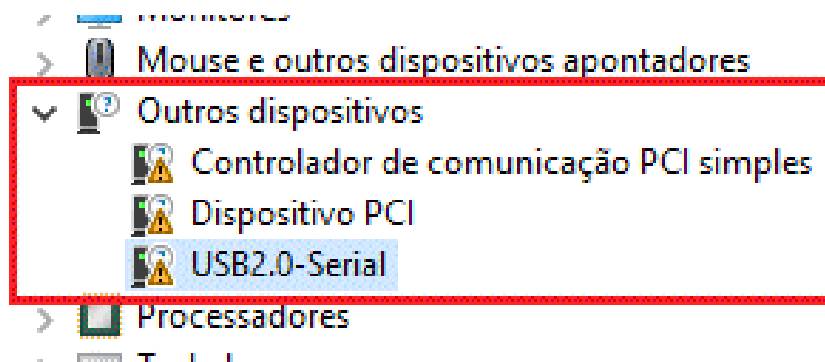
No exemplo abaixo, a placa UNO foi reconhecida com sucesso pela porta COM de número 5, essa informação será útil posteriormente.

Figura 7 - Tela do gerenciador de dispositivos.



⚠️ Caso a placa UNO não seja reconhecida pelo seu computador, ela pode aparecer, com o ícone , em “Outros dispositivos”, como na Figura 8 .

Figura 8 - Indicador de que a placa Arduino não foi reconhecida.



Isso acontece devido à falta de um driver para a interpretação do dispositivo. Para resolver esse problema, basta baixar o driver disponível no link a seguir e executá-lo:

www.blogdarobotica.com/instalando-o-driver-serial-para-arduino/

Seleção da placa e da porta de comunicação da placa UNO

Agora que já temos a placa UNO conectada ao computador, vamos selecionar a placa e a porta de comunicação no Arduino IDE. Para tal, deve-se selecionar o modelo da placa utilizada no menu Ferramentas, para nossos exemplos usaremos o Arduino Uno, conforme Figura 9 .

Após a seleção do modelo, deve-se selecionar a porta de comunicação que foi atribuída a placa, ou seja, a porta que a placa UNO foi reconhecida. Como vimos anteriormente, em nosso exemplo, a placa UNO foi reconhecida pela COM de número 5. A Figura 10 mostra a seleção da COM através do menu Ferramentas.

Figura 9 - Seleção da placa Arduino.

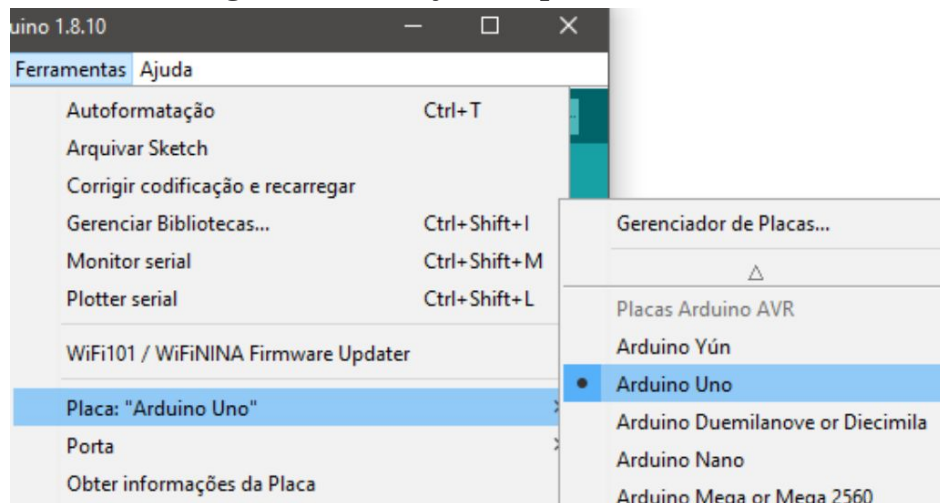
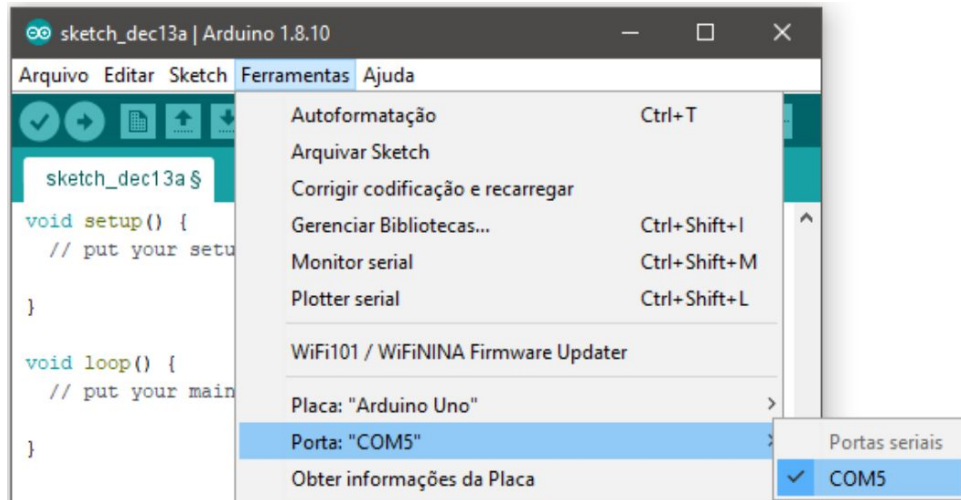


Figura 10 - Seleção da porta COM.



Após isto, sua placa UNO estará pronta e o ambiente de desenvolvimento configurado para uso.

OceanofPDF.com

3. FUNDAMENTOS DE PROGRAMAÇÃO

Programação pode ser definida como o processo de projetar, escrever, verificar e manter um código ou programa que comande as ações de uma máquina, como computador, celular, microcontrolador, entre outras.

Um código, programa ou sketch – como são denominados os códigos em Arduino, são compostos por um conjunto de instruções sequenciais definido pelo programador que descrevem as tarefas que a máquina deve realizar.

Em outras palavras, para que a máquina execute comandos específicos, é necessário elaborar um programa escrito contendo todas as instruções ordenadas sequencialmente em uma linguagem de programação.

Uma linguagem de programação é um idioma artificial desenvolvido para prover a comunicação entre uma pessoa (programador) e uma máquina (computador, microcontrolador, etc.).

Atualmente, existem centenas de linguagens de programação e, em todas elas, o programador pode definir como a máquina deve atuar, como armazenar ou transmitir os dados, quais ações tomar em diferentes situações, quando finalizar a sua operação, entre outras. Dentre as linguagens de programação mais populares, pode-se citar o C, C++, C#, Java, JavaScript, Python e PHP.

Os programas para microcontroladores compatíveis com o projeto Arduino são implementados tendo como referência a linguagem de programação C++, mantendo preservada sua sintaxe clássica de declaração

de variáveis, nos operadores, nas estruturas e em muitas outras características.

Antes de aprendermos a linguagem de programação do Arduino devemos conhecer alguns elementos básicos que compõem um Sketch. Desta forma, a estrutura de um sketch, variáveis, funções e bibliotecas serão detalhadas a seguir.

ESTRUTURAS BÁSICA DE UM SKETCH

Todos os Sketch Arduino devem ter a estrutura composta pelas funções `setup ( )` e `loop ( )` , conforme ilustra o exemplo a seguir.

1	<code>void setup () {</code>
2	Comando 1;
3	Comando 2;
4	...
5	<code>}</code>
6	<code>void loop () {</code>
7	Comando 3;
8	Comando 4;
9	...
10	<code>}</code>

A função `setup ( )` é responsável pelas configurações iniciais da placa microcontroladora, tais como definições de pinos de entrada e saída, inicialização da comunicação serial, entre outras. Essa função é executada apenas uma vez, na inicialização do programa.

A função `loop ( )` é a função principal do programa e é executada continuamente enquanto a placa microcontroladora estiver ligada. É nesta função que todos os comandos e operações deverão ser escritos.

OBSERVAÇÕES:

- As instruções dadas em um Sketch são realizadas de forma sequencial;
- É necessário incluir o sinal ; (ponto e vírgula) para que o programa identifique onde uma instrução deve ser finalizada.

VARIÁVEIS

As variáveis são expressões que podemos utilizar em programas para nomear e armazenar um dado para uso posterior, como dados de um sensor ou um valor calculado.

Antes de serem utilizadas, todas as variáveis devem ser declaradas, definindo seu tipo e, opcionalmente, seu valor inicial. Alguns tipos de variáveis encontram-se listadas na Tabela 1 .

Tabela 1 - Tipos de variáveis e suas descrições.

Tipo	Descrição
<code>char</code>	Utilizado para armazenar um valor de caractere.
<code>byte</code>	Usado para armazenar um número entre 0 e 255.
<code>int</code>	Utilizado para armazenar números inteiros.
<code>bool</code>	Empregado para armazenar dois valores: true (verdadeiro) ou false (falso).
<code>float</code>	Armazena números decimais que ocupam 32 bits (4 bytes).
<code>double</code>	Armazena números decimais que ocupam 64 bits (8 bytes).
<code>void</code>	Usada apenas em declarações de funções
<code>String</code>	Utilizada para armazenar cadeias de texto.

A declaração de uma variável pode ser melhor entendida no exemplo a seguir.

1	<code>int a;</code>
2	<code>float b;</code>
3	<code>char C;</code>
4	
5	<code>void setup () {</code>
6	<code>...</code>
7	<code>}</code>

Neste caso, declaramos uma variável com nome `a` do tipo `int`, uma variável com nome `b` do tipo `float` e uma variável de nome `C` do tipo `char`.

Quando as variáveis são declaradas antes da função `setup()`, significa que elas são variáveis globais, que podem ser usadas em todas as funções ao longo de todo o programa.

Por sua vez, as variáveis declaradas em funções específicas, como no `loop()`, são variáveis locais e só podem ser usadas dentro da sua função de origem.

Para definir um valor inicial para a variável, utilizamos o comando de atribuição, representado pelo símbolo `=`. Desta forma, quando escrevemos `int a = 10`; estamos atribuindo o valor `10` a variável de nome `a` do tipo `int`.

OBSERVAÇÕES:

- Os nomes das variáveis devem seguir algumas regras: Eles devem iniciar com letras ou sublinhado `_` (underline).
- As variáveis não podem ter nome idêntico a alguma das palavras reservadas pelo programa ou biblioteca;
- Letras maiúsculas e minúsculas fazem diferença! Se o nome da variável definida for algo como, por exemplo, `ledPin`, não será a mesma coisa que `LedPin` ou `LEDPIN`;
- O símbolo `=` tem o papel exclusivo de atribuição. A igualdade matemática é representada pela dupla igualdade `==`.

FUNÇÕES

Funções são blocos de instruções que podem ser chamados em qualquer parte do seu Sketch.

A principal motivação para o uso de funções em um programa é quando precisamos executar a mesma ação várias vezes. Desta forma, a segmentação do código em funções permite a criação de trechos de código modulares que executam uma tarefa definida e retornam à área do código a partir da qual a função foi chamada.

O uso de funções possui várias vantagens, como:

- Ajuda o programador a manter o sketch organizado;
- As funções codificam uma ação em um só lugar de forma que o trecho do código seja pensado e escrito apenas uma vez;
- Reduz as chances de erros na modificação quando o código precisa ser alterado;
- Facilita a reutilização de código em outros programas;
- Torna o código mais legível.

As duas funções principais na criação de um sketch no Arduino são `void setup ()` e `void loop ()`, mas existem várias outras funções predefinidas para controlar uma placa microcontroladora, conforme mostra a Tabela 2 .

Tabela 2 – Funções e suas descrições.

Controle	Função	Descrição
Entrada e saída digitais	<code>digitalRead ()</code>	Lê o valor de um pino digital especificado.
	<code>digitalWrite ()</code>	Escreve no pino digital HIGH ou LOW.

	<code>pinMode ()</code>	Configura o pino para funcionar como saída ou entrada.
Entrada e saída analógicas	<code>analogRead ()</code>	Lê o valor de um pino analógico especificado.
	<code>analogWrite ()</code>	Aciona uma onda PWM em um pino.
Funções temporizadoras	<code>delay ()</code>	Pausa o programa por um período (em milissegundos)
	<code>millis ()</code>	Retorna o número de milissegundos passados desde que a placa Arduino começou a executar o programa.
Entradas e saídas avançadas	<code>tone ()</code>	Gera uma onda quadrada na frequência especificada em um pino.
	<code>noTone ()</code>	Interrompe a função <code>tone ()</code>
Números aleatórios	<code>random ()</code>	Gera números pseudoaleatórios.
	<code>randomSeed ()</code>	Inicializa o gerador de números pseudoaleatórios.
Comunicação	<code>Serial</code>	Usada para comunicação entre uma placa Arduino e um computador ou outros dispositivos.

OBSERVAÇÕES:

- Para saber mais sobre as funções predefinidas do Arduino, acesse o site: <https://www.arduino.cc/reference/pt/>. Nele você encontrará outras funções não especificadas neste e-book e poderá consultar a descrição, sintaxe e parâmetros das funções que desejar;
- Os sinais analógicos são aqueles que variam continuamente ao longo do tempo. Por sua vez, os sinais digitais assumem valores discretos (0 ou 1);
- Configurar um pino digital em HIGH significa colocar o pino digital em nível lógico alto (1), ou seja, 5 V. Definir um pino digital

como LOW significa colocar o pino digital em nível lógico baixo (0), ou seja, 0 V.

Além dessas funções, você também pode escrever suas próprias funções. Uma função deve ser criada fora das funções `setup ( )` e `loop ( )` , seguindo a sintaxe descrita abaixo.

1	Tipo nome_da_funcao(declaração de parâmetros){
2	Declaração de variável; //opcional
3	Comando 1;
4	Comando 1;
5	...
6	}

Há dois tipos de funções: As que não retornam nenhum valor e as que retornam algum valor para a função onde está inserida.

As funções que não retornam nenhum valor são do tipo `void` . As funções do tipo `void` não retorna nenhum valor para a função que a chamou, ou seja, as ações executadas nessas funções não retornam números, variáveis ou caracteres para a função principal.

Por sua vez, as funções que retornam valor podem ser do tipo `int` , `float` , `string` , entre outras. Uma função do tipo `int` , por exemplo, retorna um valor inteiro para a função que a chamou. Existem duas formas de retorno de uma função, uma delas é quando o finalizador de função `}` é encontrado e a outra é usando a declaração `return` .

A declaração `return` termina uma função e retorna um valor desejado.

OBSERVAÇÕES:

- Se a função retorna um valor, é obrigatório que seja determinado o tipo de retorno, que pode ser um número inteiro, um caractere ou um número real;
- As variáveis declaradas no parâmetro são variáveis de entrada, cujos tipos também devem ser especificados;
- Barra dupla (`/ /`) pode ser utilizada para fazer um breve comentário em alguma linha do código. A função do comentário é deixar o código claro tanto para o programador quanto para outras pessoas. Os comentários são ignorados pelo compilador do código;
- Também é possível comentar várias linhas do código, para isso você deve incluir os comandos `/ *` na linha de início e `* /` ao final da linha que finaliza o trecho a ser comentado.

BIBLIOTECAS

As bibliotecas são um conjunto de instruções desenvolvidas para executar tarefas específicas relacionadas a um determinado dispositivo. O uso de bibliotecas facilita a conexão a sensores, a uma tela, a um módulo etc., além de poupar tempo do programador.

Algumas bibliotecas já vêm instaladas com o Arduino IDE, outras podem ser incluídas a partir de download e você também pode criar a sua própria. Muitas vezes o fabricante de um sensor, módulo, atuador, entre outros, fornece uma biblioteca para facilitar a programação.

Uma biblioteca padrão é chamada através da seguinte instrução:

1	<code>#include < nome_da_biblioteca ></code>
---	--

Por sua vez, uma biblioteca criada pelo usuário segue a sintaxe:

--	--

OBSERVAÇÕES:

- Usamos a diretiva **#include** e quando desejamos incluir uma biblioteca externa ao nosso Sketch. Com isso, teremos acesso a muitas bibliotecas escritas especialmente para a linguagem Arduino;
- Outra diretiva que será utilizada em nossos projetos é a **#define** , que permite dar um nome a um valor constante antes do programa ser compilado. Uma vantagem da utilização dessa diretiva é que as variáveis definidas a partir dela não ocupam espaço na memória de programa do chip;
- Instruções com **#include** e **#define** não são terminadas com ; (ponto e vírgula).

OPERADORES

Os operadores podem ser classificados de várias formas, como: aritméticos, de comparação, booleanos, de incremento e de decremento.

Operadores aritméticos

Os operadores aritméticos são as representações que utilizamos para realizar as operações aritméticas básicas (Tabela 3), como somar, subtrair, dividir, multiplicar, entre outras.

Tabela 3 - Operadores aritméticos.

Operador	Nome	Sintaxe	Resultado
+	Adição	$x = y + z$	x é igual a soma de y mais z

-	Subtração	$x = y - z$	x é igual a subtração de y menos z
*	Multiplicação	$x = y * z$	x é igual a multiplicação de y vezes z
/	Divisão	$x = y / z$	x igual a divisão de y por z
%	Resto da divisão	$x = y \% z$	x é igual ao resto da divisão de y por z

Operadores de comparação e booleanos

Para programar corretamente um microcontrolador, é necessário aprender a usar de forma adequada os operadores de comparação e booleanos.

Os operadores de comparação, como o próprio nome diz, compara dois valores retornando verdadeiro (true) ou falso (false). Observe na Tabela 4 a seguir os operadores de comparação.

Tabela 4 - Operadores de comparação.

Operador	Nome	Sintaxe	Resultado
==	Igual	$x == y$	Retorna true (verdadeiro) se x for igual a y
!=	Diferente	$x != y$	Retorna true (verdadeiro) se x for diferente de y
<	Menor que	$x < y$	Retorna true (verdadeiro) se x for menor que y
>	Maior que	$x > y$	Retorna true (verdadeiro) se x for maior que y
<=	Menor ou igual a	$x <= y$	Retorna true (verdadeiro) se x for menor ou igual a y
>=	Maior ou igual a	$x >= y$	Retorna true (verdadeiro) se x for maior ou igual a y

Os operadores booleanos são utilizados para testes lógicos entre elementos em um teste condicional. Assim como os operadores de comparação, os operadores booleanos (Tabela 5) também retornam verdadeiro (true) e falso (false) conforme o resultado dos testes. Os operadores booleanos são:

Tabela 5 - Operadores booleanos.

Operador	Nome
&&	E lógico
	OU lógico

!	NÃO lógico
---	------------

Operador E lógico

O E lógico resulta em verdadeiro apenas se ambos os operandos forem verdadeiros. Representamos o E lógico com & & .

Exemplo: Se quisermos verificar se um determinado valor de temperatura se encontra entre uma faixa de valores (entre 30 e 50°C), podemos utilizar:

Se temperatura >=30 && temperatura <=50

Se o valor encontrado de temperatura for maior ou igual a 30 e menor ou igual a 50 a condição será satisfeita, retornando verdadeiro (true).

Operador OU lógico

O OU lógico resulta em verdadeiro se pelo menos um dos operadores for verdadeiro. Representamos o OU lógico com || .

Exemplo: Se quisermos verificar se o valor de temperatura é igual a pelo menos um valor determinado (30°C ou 50°C), podemos utilizar:

Se temperatura == 30 || temperatura == 50

Neste caso, se o valor encontrado de temperatura for igual a 30 ou igual a 50 a condição será satisfeita, retornando verdadeiro (true).

Operador NÃO lógico

O NÃO lógico resulta em verdadeiro se o operando for falso. Representamos o NÃO lógico com ! .

Exemplo: Se quisermos mudar o estado de uma variável x de verdadeiro para falso e vice-versa, podemos utilizar:

!x

Neste caso, se x for verdadeiro (true), o uso do NÃO lógico transformará seu estado para falso (false). De mesmo modo, se x for falso (false), o NÃO lógico transformará seu estado para verdadeiro (true).

Operadores de incremento e decremento

Os operadores de incremento e decremento são operadores unitários utilizados com a finalidade de adicionar ou subtrair em uma unidade o valor de uma variável. O operador de incremento é representado por ++ e o de decremento por --.

Os operadores de incremento e decremento podem ser prefixos ou pós-fixos dependendo de serem posicionados antes ou depois do nome da variável a ser implementada ou decrementada.

Tabela 6 - Operadores de incremento e decremento.

Tipo	Operador	Significado
Prefixo	++x	Incrementa x em um e retorna o novo valor de x
	--x	Decrementa x em um e retorna o novo valor de x
Pós-fixos	x++	Incrementa x em um e retorna o valor antigo de x
	x--	Decrementa x em um e retorna o valor antigo de x

Exemplo:

1	x=2;
---	------

2	y=++x; //x agora contém 3 e y também
3	y=x++; //x contém 4, mas y ainda contém 3

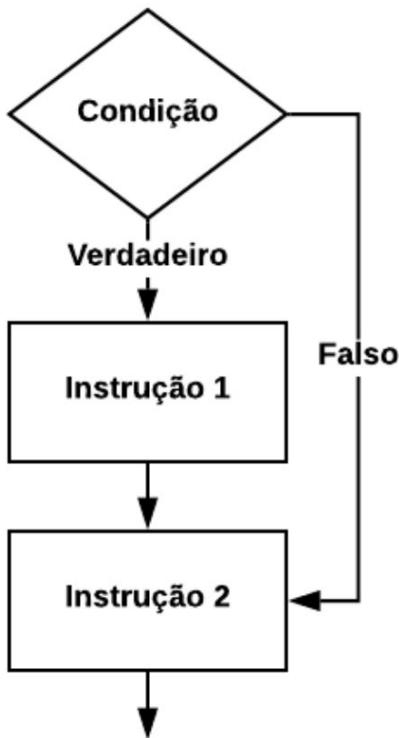
ESTRUTURA DE CONTROLE DE FLUXO

As estruturas de controle são blocos de programação que analisam variáveis e decidem como o programa deve se comportar com base em parâmetros pré-definidos. Em outras palavras, a estrutura de controle de fluxo determina como o programa funcionará diante de certas condições ou parâmetros.

A Figura 1 1 ilustra um exemplo de estrutura de controle em que, dependendo do resultado obtido na condicional (verdadeiro ou falso), o programa executará diferentes instruções.

Neste caso, se o resultado da condicional for verdadeiro o programa executará a Instrução 1 e, em sequência, a instrução 2. No entanto, se o resultado da condicional for falso, apenas a Instrução 2 será executada pelo programa.

Figura 11 - Estrutura de controle de fluxo.



IF

O comando I F (se) é uma estrutura de controle de fluxo de seleção. Usamos esse comando para checar uma condição. Se a condição for satisfeita (verdadeiro) uma lista de instruções delimitadas por {} (chaves) serão executadas.

No entanto, se a condição não for satisfeita (falso), esta lista de instruções não será executada e o programa seguirá a sequência de comandos escritos depois do I F .

A sintaxe do comando I F na programação Arduino é:

1	<code>if (condição){</code>
2	<code> Comando 1;</code>
3	<code> Comando 2;</code>
4	<code> ... }</code>

IF...ELSE

A combinação dos comandos IF...ELSE E permite maior controle sobre o fluxo de código que o comando I F , por possibilitar que múltiplos testes sejam agrupados. O comando ELSE (senão) será executado se a condição do comando IF (se) resultar em falso.

A sintaxe dos comandos IF...ELSE na programação Arduino é:

1	if (condição1){
2	Comando 1;
3	}
4	else {
5	Comando 2;
6	}

Dentro do comando ELSE E podemos adicionar outro comando I F , de modo que múltiplos testes podem ser executados ao mesmo tempo. Desta forma, a sintaxe ficará da seguinte maneira:

1	if (condição1){
2	Comando 1;
3	}
4	else if (condição2){
5	Comando 2;
6	}
7	else {
8	Comando 3;
9	}

FOR

O comando FOR (para) é uma estrutura de controle de fluxo de repetição. Este comando permite que certo trecho do código seja executado um determinado número de vezes. O comando for é útil para qualquer

operação repetitiva e é usado frequentemente com vetores para operar em coleções de dados.

A sintaxe do comando FOR é a seguinte:

1	<code>for</code> (inicialização; condição; incremento){
2	Comando 1;
3	...
4	}

A inicialização ocorre primeiro e apenas uma vez. A cada repetição do loop, a condição é testada. Se verdadeira, o bloco de comandos e o incremento são executados. Do contrário, quando a condição for falsa, o loop termina.

SWITCH...CASE

Da mesma forma que o comando I F , o comando SWITCH...CASE (selecione...caso) é uma estrutura de controle de fluxo de seleção.

Este comando permite especificar códigos diferentes para serem executados em várias condições, funcionando da seguinte maneira: um comando SWITCH compara o valor de uma variável aos valores especificados nos comandos CAS E . Quando um comando CASE é encontrado, cujo valor é igual ao da variável, o código para esse comando CAS E é executado.

A sintaxe do comando SWITCH...CASE é a seguinte:

1	<code>switch</code> (var){
2	<code>case</code> valor1:
3	comando1;
4	<code>break</code> ;

5	<code>case</code> valor2:
6	comando2;
7	<code>break</code> ;
8	<code>default</code> :
9	comando3;
10	<code>break</code> ;
11	}

A variável `var` será comparada ao valor dos vários `case`s, podendo ser do tipo `int` ou `char` . Os parâmetros `valor1` e `valor2` são constantes do tipo `int` ou `char` .

OBSERVAÇÕES:

- A palavra-chave `break` é utilizada para interromper o comando `switch` , devendo ser escrito ao final de cada `case` . Sem o comando `break` o comando `switch` continuará executando as expressões seguintes, até encontrar um `break` .
- A palavra-chave `default` é opcional e é executada quando a variável de comparação do `switch` não corresponde a nenhum valor constante dos `case` .

WHILE

O WHILE (enquanto) é uma estrutura de controle de fluxo de repetição. As instruções contidas em um WHILE serão repetidas continuamente, e infinitamente, até que a sua condicional se torne falsa. Em outras palavras, as instruções contidas no código WHILE serão executadas enquanto a condição for satisfeita (verdadeiro)

A forma geral do WHILE é:

--	--

1	<code>while (condição){</code>
2	Comando 1;
3	...
4	<code>}</code>

DO...WHILE

A estrutura DO...WHILE (faça...enquanto) funciona de forma semelhante às estruturas WHILE e FOR, com exceção da condição ser testada ao final do loop, de tal modo que as instruções contidas no DO...WHILE serão executadas pelo menos uma vez.

A sintaxe do comando DO...WHILE é a seguinte:

1	<code>do {</code>
2	Comando 1;
3	<code>} while (condição);</code>

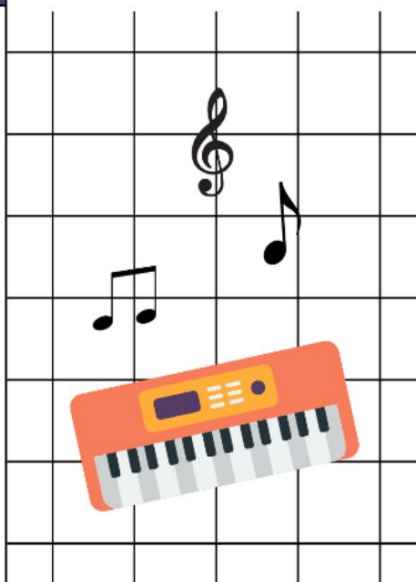
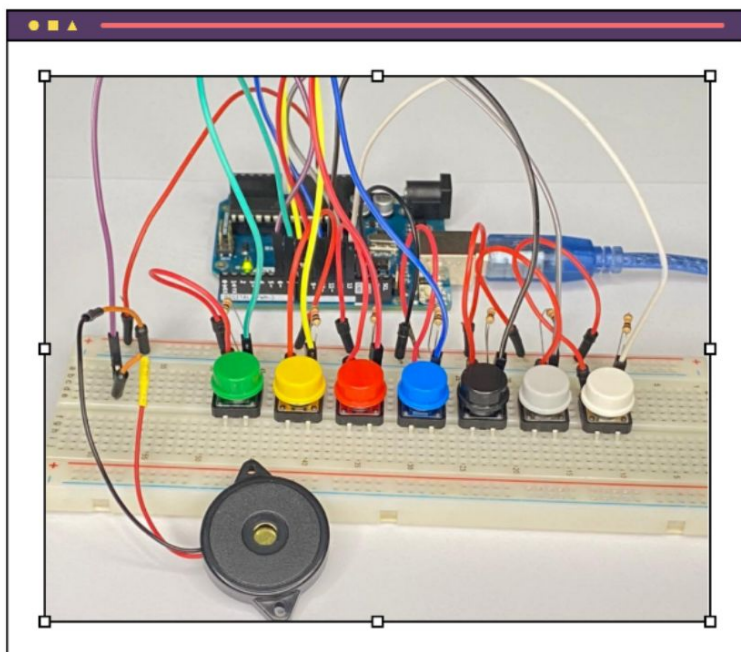
4. CRIANDO JOGOS COM ARDUINO

PIANO ELETRÔNICO

Como estão as suas habilidades musicais? Que tal testá-las em um piano eletrônico construído com o Arduino?

Neste projeto aprenderemos como transformar seu Arduino em um pequeno piano, capaz de tocar as 7 notas musicais: Dó, Ré, Mi, Fá, Sol, Lá e Si.

Para este projeto você precisará de botões, resistores e de um buzzer passivo, que é para melodias e reprodução de notas. Para tocar as notas musicais usaremos a função `tone ( )` .



MATERIAIS NECESSÁRIOS

- 1 x [Placa UNO SMD compatível com Arduino UNO](#);
- 1 x [Cabo USB](#);
- 1 x [Protoboard](#);
- 7 x [Botões do tipo push button](#);
- 7 x [Resistores de 10 k \$\Omega\$](#) ;
- 1 x [Buzzer passivo](#);
- [Jumpers macho-macho](#).

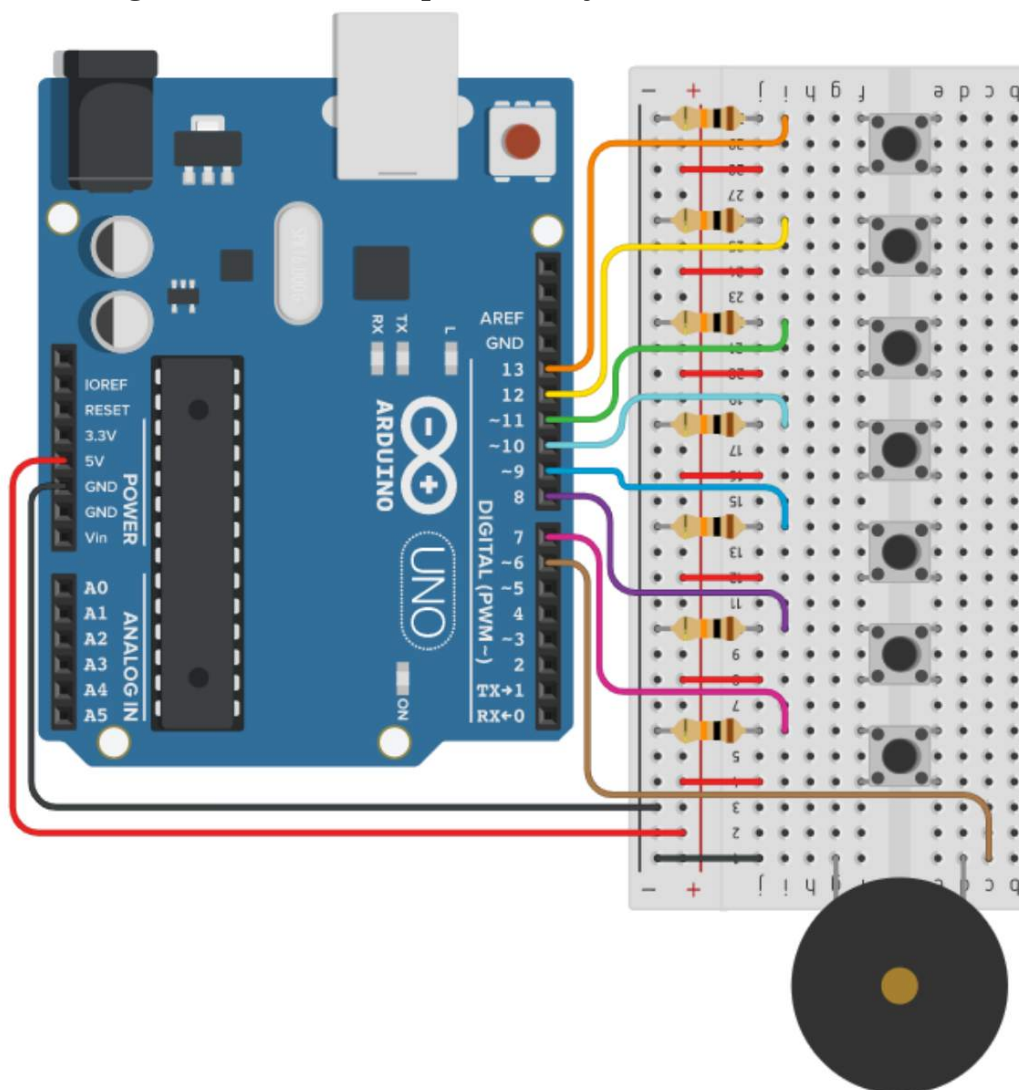
ESQUEMÁTICO DE LIGAÇÃO DOS COMPONENTES

Certifique-se que sua placa UNO esteja desligada e monte atentamente o circuito conforme a Figura 1 2 .

Ao montar seu circuito na protoboard preste atenção nos seguintes pontos:

- Você pode utilizar fios de cores diferentes ou furos diferentes na protoboard, mas deve assegurar que os componentes e fios estejam conectados na mesma ordem da Figura 1 2 ;
- O botão possui 4 terminais que são interligados aos pares. Quando pressionado, os 4 terminais são conectados gerando continuidade no circuito;
- O botão deve ser montado com dois terminais abaixo da cavidade central da protoboard e os outros dois acima;
- Conecte o push button com um resistor de 10 k Ω pull-down (resistor conectado no GND). Desta forma, quando o botão não estiver pressionado o Arduino receberá nível lógico baixo;
- Quando o botão estiver pressionado, a placa UNO receberá nível lógico alto (HIGH).
- O terminal positivo do buzzer deve ser conectado à porta digital 6 da placa UNO e o outro terminal deve ser ligado ao GND.

Figura 12 - Circuito para o Projeto Piano Eletrônico.



Assim que você tiver certeza de que tudo foi conectado corretamente, conecte a placa UNO ao seu computador.

ELABORANDO O CÓDIGO

Com o circuito montado, vamos a programação do piano. Para melhor compreensão o código será explicado passo a passo a seguir. Neste momento, observe o código abaixo e aproveite para analisar sua estrutura.

```
//Define os pinos dos botões
#define botao1 7
#define botao2 8
#define botao3 9
#define botao4 10
#define botao5 11
#define botao6 12
#define botao7 13

#define buzzer 6 //Pino do Buzzer

//Cada nota e sua frequência
#define C 262 //Nota Dó
#define D 294 //Nota Ré
#define E 330 //Nota Mi
#define F 349 //Nota Fá
#define G 392 //Nota Sol
#define A 440 //Nota Lá
#define B 493 //Nota Si

void setup () {
    //Configura os botões como entrada
    pinMode (botao1, INPUT );
    pinMode (botao2, INPUT );
    pinMode (botao3, INPUT );
    pinMode (botao4, INPUT );
    pinMode (botao5, INPUT );
    pinMode (botao6, INPUT );
    pinMode (botao7, INPUT );
    pinMode (buzzer, OUTPUT ); //Configura o buzzer como saída
}

void loop () {
    //Faz as leituras do status de cada botão e guarda na variável correspondente
    int status1 = digitalRead (botao1);
    int status2 = digitalRead (botao2);
    int status3 = digitalRead (botao3);
```

```
int status4 = digitalRead (botao4);
int status5 = digitalRead (botao5);
int status6 = digitalRead (botao6);
int status7 = digitalRead (botao7);

if (status1 == HIGH ) { //Se o botão 1 for pressionado
    tone (buzzer, C); //Toca nota Dó
    delay (200); //por 200 milissegundos
    noTone (buzzer); //Interrompe o som
}
if (status2 == HIGH ) { //Se o botão 2 for pressionado
    tone (buzzer, D); //Toca nota Ré
    delay (200); //por 200 milissegundos
    noTone (buzzer); //Interrompe o som
}
if (status3 == HIGH ) { //Se o botão 3 for pressionado
    tone (buzzer, E); //Toca nota Mi
    delay (200); //por 200 milissegundos
    noTone (buzzer); //Interrompe o som
}
if (status4 == HIGH ) { //Se o botão 4 for pressionado
    tone (buzzer, F); //Toca nota Fá
    delay (200); //por 200 milissegundos
    noTone (buzzer); //Interrompe o som
}
if (status5 == HIGH ) { //Se o botão 5 for pressionado
    tone (buzzer, G); //Toca nota Sol
    delay (200); //por 200 milissegundos
    noTone (buzzer); //Interrompe o som
}
if (status6 == HIGH ) { //Se o botão 6 for pressionado
    tone (buzzer, A); //Toca nota Lá
    delay (200); //por 200 milissegundos
    noTone (buzzer); //Interrompe o som
}
if (status7 == HIGH ) { //Se o botão 7 for pressionado
    tone (buzzer, B); //Toca nota Si
```

```
delay (200); //por 200 milissegundos
noTone (buzzer); //Interrompe o som
}
}
```

Para melhor compreensão do código acompanhe os seguintes passos:

1. Declarar as variáveis

Neste código declaramos 7 variáveis para os botões (botao1, botao2, botao3, botao4, botao5, botao6, botao 7), e uma variável para o buzzer (buzzer).

Em seguida, definimos as variáveis para armazenar a frequência das 7 notas musicais, conforme a Tabela a seguir.

Variável	Nota Musical	Frequência
C	Dó	262
D	Ré	294
E	Mi	330
F	Fá	349
G	Sol	392
A	Lá	440
B	Si	493

2. Configuração dos pinos de entrada e saída

Os pinos digitais em que os botões estão conectados foram configurados como entrada (INPUT) e o pino digital 6, conectado ao buzzer, foi configurado como saída (OUTPUT).

3. Leitura dos botões

Iniciamos a função loop () realizando a leitura dos botões. Os estados dos botões botao 1 , botao 2 , botao 3 , botao 4 , botao 5 , botao

6 e botao 7 são armazenados nas variáveis status 1 , status 2 , status 3 , status 4 , status 5 , status 6 e status 7 , respectivamente.

4. Verificação se o botão foi pressionado

Após a leitura, criamos uma série de estruturas de seleção usando o condicional `if` . Nelas, verificamos se os botões foram pressionados (`statusX == HIGH`) .

5. Tocar a nota correspondente

Sendo a condição verdadeira (botão pressionado), tocamos a nota correspondente a ele. Por exemplo, quando o botão 1 (botao 1) for pressionado, a nota Dó deverá ser tocada.

Para tocar a nota, usamos a função `tone` () passando como parâmetros a variável buzzer e a nota correspondente ao botão pressionado.

Sintaxe da função `tone` () : `tone` (pino, frequência);

Em seguida, inserimos um intervalo de 200 milissegundos.

Para interromper o toque da nota pelo buzzer, usamos a função `noTone` () e passamos como parâmetro a variável buzzer.

Sintaxe da função `noTone` () : `noTone` (pino);

TINKERCAD

Para melhor visualizar e/ou simular o circuito e a programação deste projeto, basta acessar o seguinte link:

www.blogdarobotica.com/tinkercad-piano .

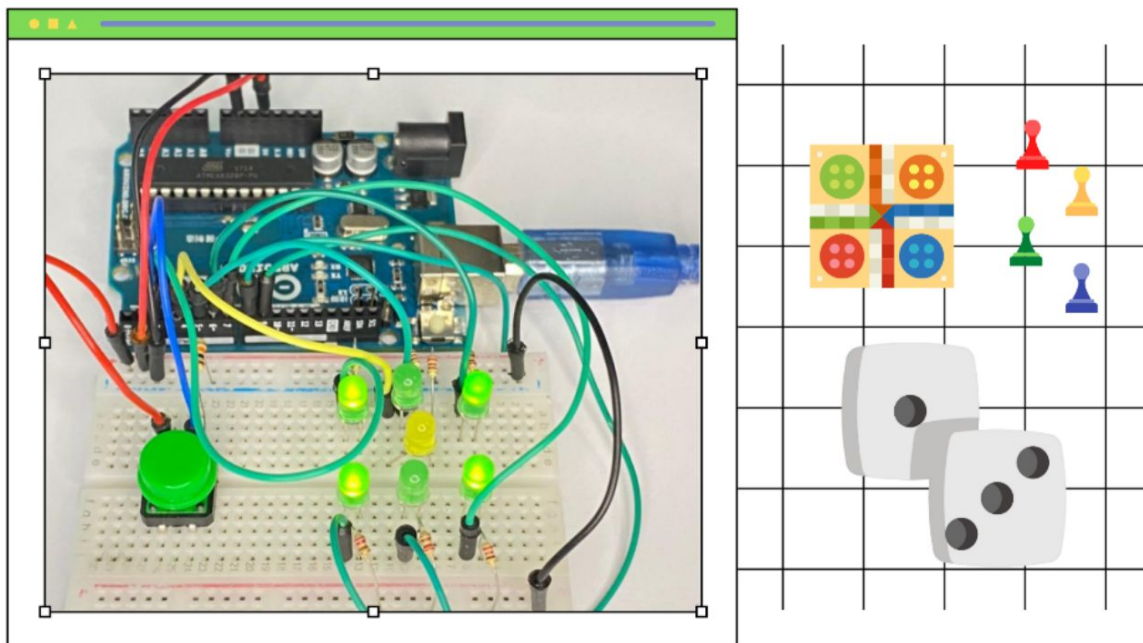
OceanofPDF.com

JOGO DE DADOS COM LEDS – DADO ELETRÔNICO

O jogo de dados é um dos mais conhecidos e antigos jogos do mundo, tão antigo quanto a civilização humana. Os dados aparecem retratados junto a uma espécie de jogo de tabuleiro em vasos pintados da Grécia antiga, sendo mencionados no Vedas (livro sagrado Hindu) e no Antigo Testamento da Bíblia.

Os dados são frequentemente usados em jogos de azar, adivinhações, RPG e jogos de tabuleiro quando algo aleatório precisa ser feito, como mover uma peça.

Existem inúmeros tipos de dados, mas o mais usado e clássico é o cubo (seis faces), gravado com números de 1 a 6. Neste projeto, construiremos um dado eletrônico usando o Arduino, LEDs e um botão.



Sempre que o jogador apertar o botão, os LEDs exibirão um valor aleatório de 1 a 6. Para criar os números aleatórios que serão sorteados no dado eletrônico usaremos as funções `randomSeed ( )` e `random ( )`.

A função `randomSeed ( )` será utilizada para iniciar o gerador de números aleatórios utilizando como entrada aleatória a leitura de um pino analógico que não está sendo utilizado. Por sua vez, a função `random ( )` será usada para gerar um número aleatório entre 1 e 6.

Você poderá usar o dado eletrônico para o jogo de dados ou para outros jogos de tabuleiro. Chame seus amigos e que a sorte seja lançada!

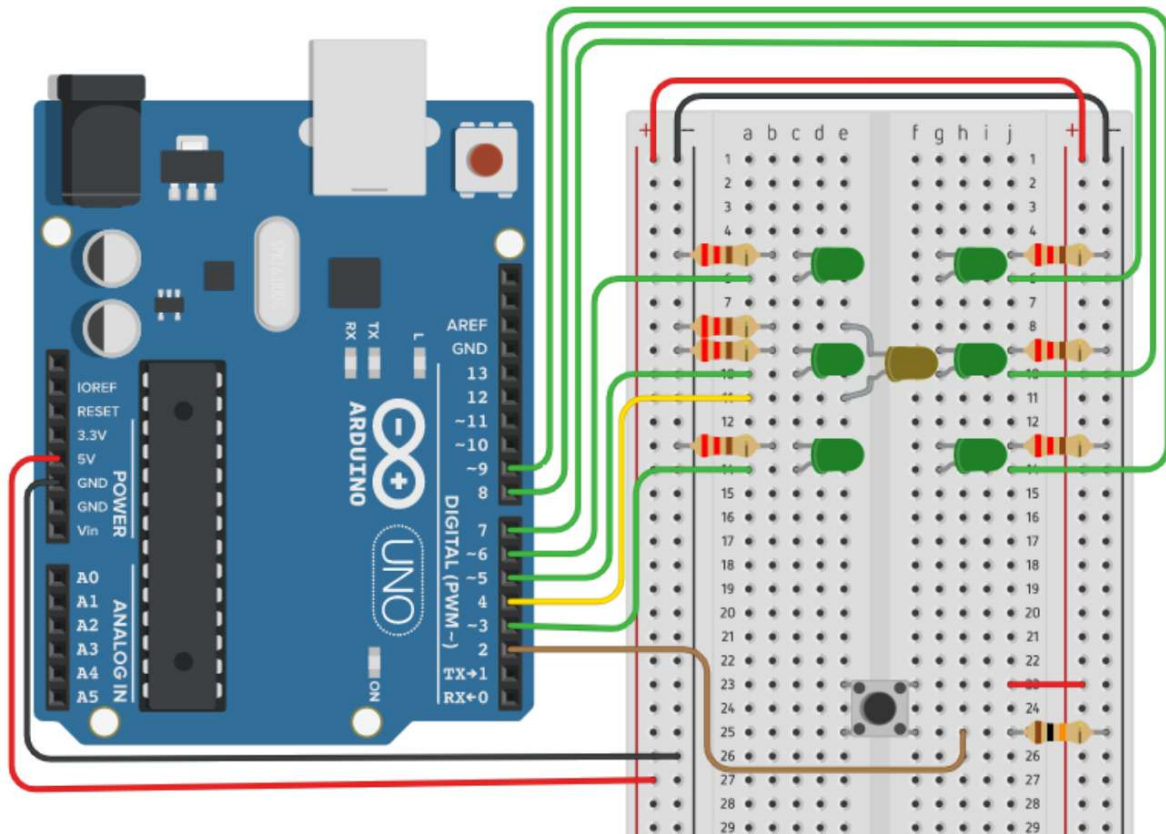
MATERIAIS NECESSÁRIOS

- 1 x [Placa UNO SMD compatível com Arduino UNO](#);
- 1 x [Cabo USB](#);
- 1 x [Protoboard](#);
- 7 x [LEDs difusos de 5 mm](#);
- 7 x [Resistores de 220 \$\Omega\$](#) ;
- 1 x [Resistores de 10 k \$\Omega\$](#) ;
- 1 x [Botão do tipo push button](#);
- [Jumpers macho-macho](#).

ESQUEMÁTICO DE LIGAÇÃO DOS COMPONENTES

Certifique-se que sua placa UNO esteja desligada e monte atentamente o circuito conforme a Figura 13 .

Figura 13 - Circuito para o Projeto Jogo de Dados – Dado Eletrônico.



Ao montar seu circuito na protoboard, preste atenção nos seguintes pontos:

- O terminal mais longo de cada LED deve ser conectado ao pino digital correspondente. O terminal mais curto é o cátodo (negativo) e deve ser conectado ao terra (GND);
- Em nosso projeto, utilizaremos resistores de $220\ \Omega$ para reduzir a tensão e a corrente de alimentação dos LEDs;
- O botão possui 4 terminais que são interligados aos pares, de modo que quando pressionado os 4 são conectados gerando continuidade no circuito;
- O botão deve ser montado com dois terminais abaixo da cavidade central da protoboard e os outros dois acima;

- Conecte o push button com um resistor de 10 k Ω pull-down (resistor conectado no GND). Desta forma, quando o botão estiver pressionado, a placa UNO retornará nível lógico alto (HIGH).

Assim que você tiver certeza de que tudo foi conectado corretamente, conecte a placa UNO ao seu computador.

ELABORANDO O CÓDIGO

Com o circuito montado, vamos a programação do jogo. Para melhor compreensão o código será explicado passo a passo a seguir. Neste momento, observe o código abaixo e aproveite para analisar sua estrutura.

```
//Pinos para os LEDs
#define ledPin1 6
#define ledPin2 5
#define ledPin3 3
#define ledPin4 4
#define ledPin5 7
#define ledPin6 8
# define ledPin7 9

#define botao 2 //Pino do botão
int leitura = 0; //Variável para guardar a leitura do status do botão
int sorteio; //Variável para armazenar o número aleatório sorteado

void setup () {
  //Inicializa a comunicação serial
  Serial . begin (9600);

  //Configura o botão como entrada
  pinMode (botao, INPUT );
```

```
//Configura os LEDs como saída
pinMode (ledPin1, OUTPUT );
pinMode (ledPin2, OUTPUT );
pinMode (ledPin3, OUTPUT );
pinMode (ledPin4, OUTPUT );
pinMode (ledPin5, OUTPUT );
pinMode (ledPin6, OUTPUT );
pinMode (ledPin7, OUTPUT );

//Inicializa o gerador de números aleatórios, começando em um ponto arbitrário
randomSeed ( analogRead (0));
}

void loop () {

    leitura = digitalRead (botao); //Realiza a leitura do status do botão e salva na variável leitura

    if (leitura == HIGH ) { //Se botão for pressionado
        sorteio = random (1, 7); //Sorteia número aleatório de 1 a 6
        Serial . println (sorteio); //Imprime o número sorteado no monitor serial

        if (sorteio == 1) { //Se o número sorteado for um
            digitalWrite (ledPin4, HIGH );
        }

        if (sorteio == 2) { //Se o número sorteado for dois
            digitalWrite (ledPin1, HIGH );
            digitalWrite (ledPin7, HIGH );
        }

        if (sorteio == 3) { //Se o número sorteado for três
            digitalWrite (ledPin1, HIGH );
            digitalWrite (ledPin4, HIGH );
            digitalWrite (ledPin7, HIGH );
        }

        if (sorteio == 4 ) { //Se o número sorteado for quatro
```

```
digitalWrite (ledPin1, HIGH );
digitalWrite (ledPin3, HIGH );
digitalWrite (ledPin5, HIGH );
digitalWrite (ledPin7, HIGH );
}

if (sorteio == 5) { //Se o número sorteado for cinco
digitalWrite (ledPin1, HIGH );
digitalWrite (ledPin3, HIGH );
digitalWrite (ledPin4, HIGH );
digitalWrite (ledPin5, HIGH );
digitalWrite (ledPin7, HIGH );
}

if (sorteio == 6) { //Se o número sorteado for seis
digitalWrite (ledPin1, HIGH );
digitalWrite (ledPin2, HIGH );
digitalWrite (ledPin3, HIGH );
digitalWrite (ledPin5, HIGH );
digitalWrite (ledPin6, HIGH );
digitalWrite (ledPin7, HIGH );
}
delay (3000);
apaga(); //chama a função para desligar os LEDs
}
}

//Função para apagar todos os LEDs
void apaga() {
digitalWrite (ledPin1, LOW );
digitalWrite (ledPin2, LOW );
digitalWrite (ledPin3, LOW );
digitalWrite (ledPin4, LOW );
digitalWrite (ledPin5, LOW );
digitalWrite (ledPin6, LOW );
digitalWrite (ledPin7, LOW );
}
```

Para melhor compreensão do código acompanhe os seguintes passos:

1. Declarar as variáveis

Neste código declaramos 7 variáveis para os LEDs (`ledPin1`, `ledPin2`, `ledPin3`, `ledPin4`, `ledPin5`, `ledPin6`, `ledPin7`), uma variável para o botão (`botao`), uma variável para leitura do botão (`leitura`) e outra para armazenar o número aleatório sorteado (`sorteio`).

2. Inicialização da comunicação serial e configuração das portas de entrada e saída

A comunicação serial foi inicializada por meio da instrução: `Serial.begin(9600)` ; .

O pino digital 2, em que o botão está conectado, foi configurado como entrada (`INPUT`) e os pinos de 3 a 9, conectados aos LEDs, foram configurados como saída (`OUTPUT`).

3. Inicialização do gerador de números aleatórios

O gerador de números aleatórios foi inicializado através da função `randomSeed ( analogRead (0))` ; que utiliza como entrada aleatória a leitura analógica do pino A0.

Sem a função `randomSeed ( analogRead (0))` ; , sempre que o Arduino fosse inicializado teríamos as mesmas sequências de lançamento de dado.

4. Realizar a leitura do botão

Iniciamos o loop realizando a leitura do botão e armazenamos o valor lido na variável `leitura` .

5. Verificação se o botão foi pressionado

Após a leitura, criamos uma estrutura de seleção usando o condicional `if` . Nela, verificamos se o botão foi pressionado (`leitura == HIGH`) .

6. Sorteio de um número aleatório entre 1 e 6

Sendo a condição verdadeira (botão pressionado), procedemos o sorteio de um número aleatório entre 1 e 6 para ser exibido no dado eletrônico.

Para realização do sorteio usamos a função `random()` ; que gera números aleatórios. Dentro dessa função incluímos o limite mínimo e máximo do valor aleatório (1, 7).

A função `random()` ; retorna um valor aleatório entre o limite mínimo e máximo – 1. Dessa forma, `random(1, 7)` ; retorna um valor aleatório entre 1 e 6.

7. Acionamento dos LEDs do dado eletrônico formando o número sorteado

Para acionar os LEDs do dado eletrônico na forma do número sorteado, usamos seis estruturas de seleção `if` . Dentro da condição do `if` , inserimos uma comparação de igualdade. Nela, comparamos se o valor sorteado foi igual a 1, 2, 3, 4, 5 ou 6.

Dentro de cada `if` , escrevemos os comandos para acionamento dos LEDs do dado eletrônico na forma do número sorteado. Por exemplo, se o número sorteado foi igual a 1 (`sorteio == 1`) o LED central do dado eletrônico deverá ser acionado (`digitalWrite(ledPin4, HIGH)` ;).

8. Tempo de acionamento dos LEDs do dado eletrônico formando o número sorteado

O dado eletrônico exibirá o número sorteado por 3 segundos e logo após terá todos os seus LEDs desligados novamente. Para isso, usamos a função `delay ( )` , passando como parâmetro 3000 milissegundos (3 segundos).

Para desligar todos os LEDs do dado eletrônico chamamos a função `apagar( )` .

9. Criar função apagar

Fora da função `loop ( )` , escrevemos a função `apagar()` . Nela colocamos as variáveis `ledPin1`, `ledPin2`, `ledPin3`, `ledPin4`, `ledPin5`, `ledPin6` e `ledPin7` em nível lógico baixo (`LOW`), desligando todos os LEDs do dado.

TINKERCAD

Para melhor visualizar e/ou simular o circuito e a programação deste projeto acesse o seguinte link: www.blogdarobotica.com/tinkercad-dado.

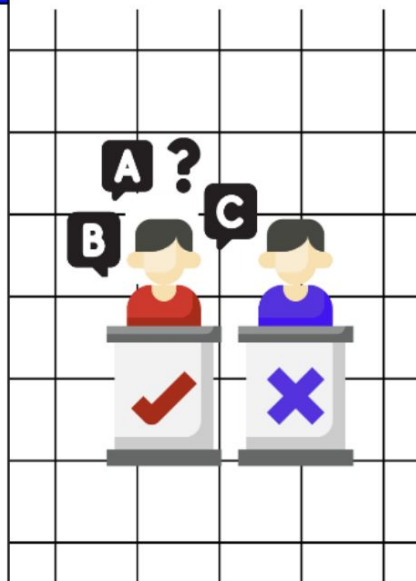
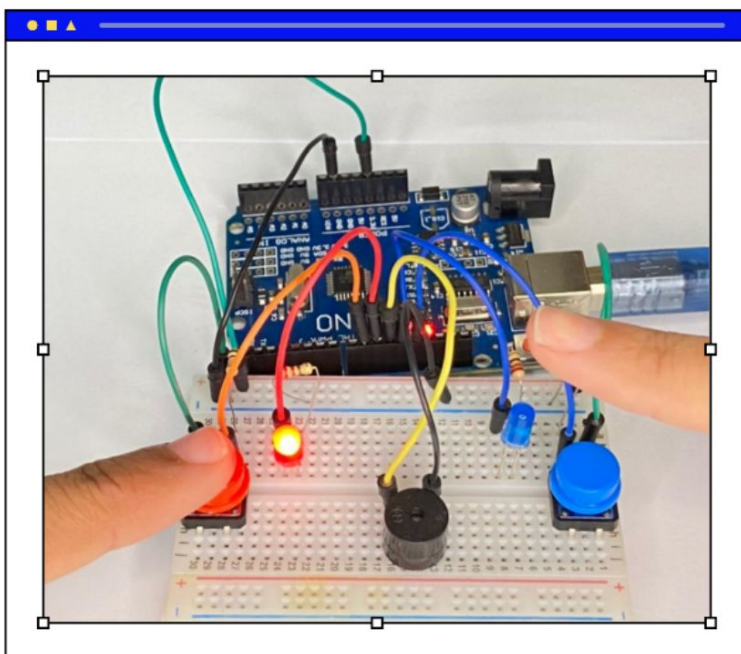
OceanofPDF.com

TORTA NA CARA – PASSA OU REPASSA

O Passa ou Repassa é um game show de perguntas e respostas. No programa, duas equipes devem cumprir diversas gincanas, como o Torta na Cara, para acumular pontos no placar geral.

No Torta na Cara, um competidor de cada equipe deve se posicionar um de frente para o outro, em meio a uma bancada equipada com dois botões e dois sinalizadores.

O apresentador faz a pergunta e o competidor deve apertar o botão assim que souber a resposta. O sinalizador do competidor que primeiro apertar o botão será ligado e ele deve responder à pergunta corretamente. Se a resposta estiver correta, o competidor somará pontos para a sua equipe e poderá acertar uma torta na cara do competidor adversário. Mas se estiver errada, ele que receberá uma torta na cara.



Neste projeto, simularemos a bancada do jogo Torta na Cara usando Arduino, LEDs, botões e um buzzer.

Escolha as perguntas, chame os amigos e divirta-se!

MATERIAIS NECESSÁRIOS

- 1 x [Placa UNO SMD compatível com Arduino UNO](#);
- 1 x [Cabo USB](#);
- 1 x [Protoboard](#);
- 2 x [Botões do tipo push button](#);
- 2 x [Resistores de 10 k \$\Omega\$](#) ;
- 2 x [Resistores de 220 \$\Omega\$](#) ;
- 2 x [LEDs difusos de 5 mm](#);
- 1 x [Buzzer ativo](#);
- [Jumpers macho-macho](#).

ESQUEMÁTICO DE LIGAÇÃO DOS COMPONENTES

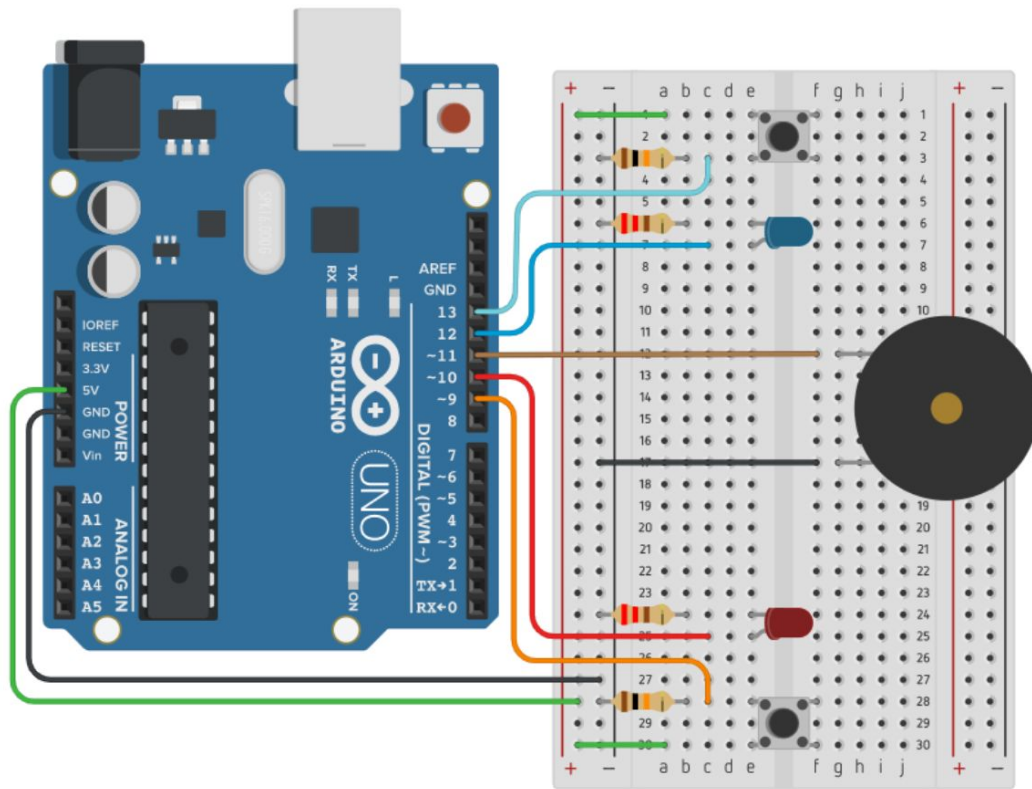
Certifique-se que sua placa UNO esteja desligada e monte atentamente o circuito conforme a Figura 1 4 .

Ao montar seu circuito na protoboard, preste atenção nos seguintes pontos:

- O terminal mais longo de cada LED deve ser conectado ao pino digital correspondente. O terminal mais curto é o cátodo (negativo) e deve ser conectado ao terra (GND);
- Em nosso projeto, utilizaremos resistores de 220 Ω para reduzir a tensão e a corrente de alimentação dos LEDs;
- O botão deve ser montado com dois terminais abaixo da cavidade central da protoboard e os outros dois acima;
- Conecte o push button com um resistor de 10 k Ω pull-down (resistor conectado no GND). Desta forma, quando o botão estiver pressionado, a placa UNO retornará nível lógico alto (HIGH).

- O terminal positivo do buzzer deve ser conectado à porta digital 11 da placa UNO e o outro terminal deve ser ligado ao GND.

Figura 14 - Circuito para o Projeto Torta na Cara - Passa ou Repassa.



Assim que você tiver certeza de que tudo foi conectado corretamente, conecte a placa UNO ao seu computador.

ELABORANDO O CÓDIGO

Com o circuito montado vamos a programação do jogo. Para melhor compreensão, o código será explicado passo a passo a seguir. Neste momento, observe o código abaixo e aproveite para analisar sua estrutura.

```
//Pino para os botões
#define botaoEsquerdo 13
#define botaoDireito 9
```

```

//Pinos para os LEDs (Azul e Vermelho)
#define ledEsquerdo 12
#define ledDireito 10

#define buzzerAtivo 11 //Pino em que o buzzer está ligado

void setup () {
    //Configura os pinos em que o buzzer e os LEDs estão conectados como saída
    pinMode (buzzerAtivo, OUTPUT );
    pinMode (ledEsquerdo, OUTPUT );
    pinMode (ledDireito, OUTPUT );

    //Configura os pinos em que os botões estão conectados como entrada
    pinMode (botaoEsquerdo, INPUT );
    pinMode (botaoDireito, INPUT );
}

void loop () {
    if ( digitalRead (botaoEsquerdo) == HIGH ) { //Se o botão esquerdo for pressionado, então:
        digitalWrite (ledEsquerdo, HIGH ); //Acende o led esquerdo
        efeitoSonoro(); //Chama a função responsável pelo efeito sonoro
        delay (5000); //Aguarda 5000 milissegundos (5 segundos)
        digitalWrite (ledEsquerdo, LOW ); //Apaga o led esquerdo

    } else if ( digitalRead (botaoDireito) == HIGH ) { //Se o botão direito for pressionado, então:
        digitalWrite (ledDireito, HIGH ); //Acende o led direito
        efeitoSonoro(); //Chama a função responsável pelo efeito sonoro
        delay (5000); //Aguarda 5000 milissegundos (5 segundos)
        digitalWrite (ledDireito, LOW ); //Apaga o led direito
    }
}

//Função responsável por fazer o efeito de 3 beeps quando um dos botões é acionado
void efeitoSonoro() {
    for ( int i = 0; i<3; i++){ //Para i = 0, enquanto i for menor que 3, realize o código e incremente
1 em i

```

```
digitalWrite (buzzerAtivo, HIGH ); //Ativa o buzzer
delay (100); //Aguarda 100 milissegundos
digitalWrite (buzzerAtivo, LOW ); //Desliga o buzzer
delay (100); //Aguarda 100 milissegundos
}
}
```

Para melhor compreensão do código acompanhe os seguintes passos:

1. Declarar as variáveis

Neste código, declaramos 2 variáveis para leitura dos botões (botaoEsquerdo e botaoDireito), duas variáveis para os LEDs (ledEsquerdo e ledDireito) e uma variável para o buzzer (buzzerAtivo).

2. Configuração das portas de entrada e saída

Os pinos digitais 10, 11 e 12, em que os LEDs e o buzzer estão conectados, foram configurados como saída (OUTPUT).

Os pinos digitais 13 e 9, em que os botões estão conectados, foram configurados como entrada (INPUT).

3. Verificação do botão pressionado

Iniciamos a função loop () , verificando qual botão foi pressionado. Para isso, usamos a estrutura de seleção condicional if .

Primeiramente, verificamos se o botão esquerdo foi pressionado (if (digitalWrite (botaoEsquerdo) == HIGH)). Sendo a condição satisfeita, o LED esquerdo (azul) será ligado e a função efeitoSonoro() será executada. Após 5 segundos, o LED esquerdo será desligado.

Se o botão esquerdo não tiver sido pressionado (botaoEsquerdo == LOW), verificamos se o botão direito foi pressionado (if (digitalWrite (botaoDireito) == HIGH)). Uma vez satisfeita a condição, o LED direito

(vermelho) será ligado e a função efeitoSonoro() será executada. Após 5 segundos, o LED direito será desligado.

4. Criar função efeitoSonoro()

Abaixo da função loop () , criamos a função efeitoSonoro() , responsável por fazer o efeito de 3 beeps no buzzer quando um dos botões for pressionado.

Para realizar a repetição dos beeps tocados pelo buzzer, usamos a estrutura de repetição **for** através da instrução: **for (int i = 0; i<3; i++) ;** .

Sintaxe: **for** (inicialização; condição; incremento);

Assim, o primeiro parâmetro incluso foi a variável **i** , usada como contador. O segundo parâmetro foi a condição a ser satisfeita para que as instruções do laço de repetição fossem executadas. Neste projeto, as instruções do **for** são realizadas enquanto **i** for menor que 3. O terceiro parâmetro foi o incremento (**i++**).

Para que o buzzer ativo toque o beep, usamos a instrução **digitalWrite** (buzzerAtivo, **HIGH**) ; . Ao contrário do buzzer passivo, cuja operação depende da frequência de onda enviada pelo microcontrolador, o buzzer ativo necessita apenas da alimentação para funcionar.

O buzzer ativo toca o beep por 100 milissegundos. Para parar o toque usamos a instrução **digitalWrite** (buzzerAtivo, **LOW**) ; .Então, o buzzer para por 100 milissegundos.

TINKERCAD

Para melhor visualizar e/ou simular o circuito e a programação deste projeto, basta acessar o seguinte link:

www.blogdarobotica.com/tinkercad-tortanacara .

OceanofPDF.com

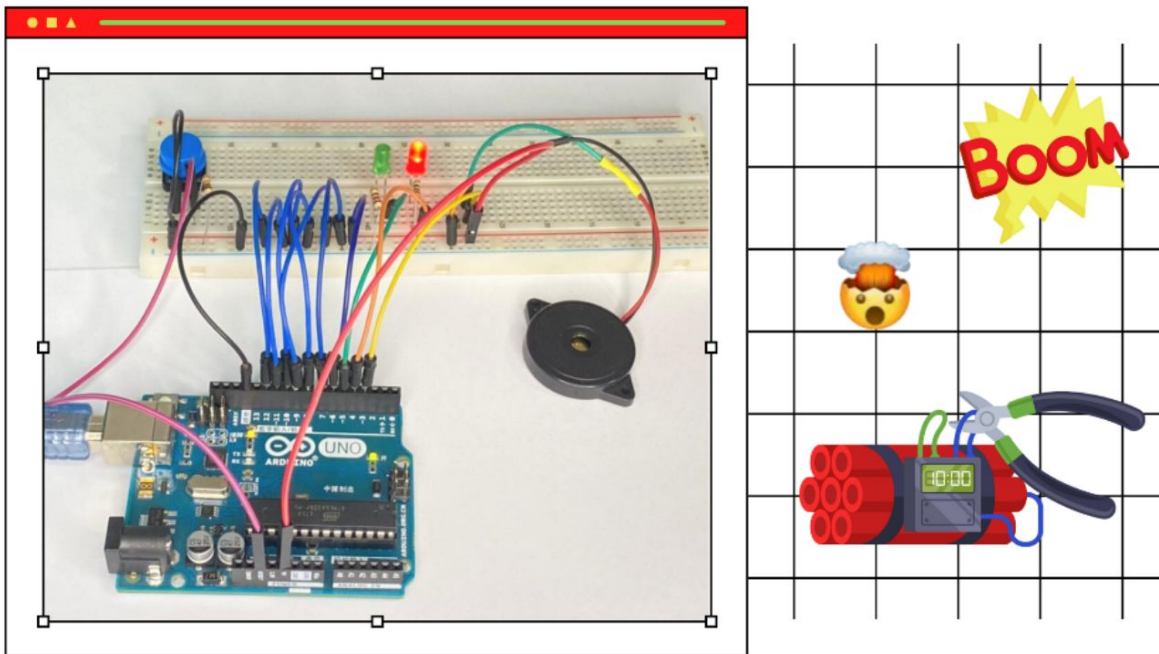
DESARME A BOMBA

Você e seu amigo estão presos em uma sala com uma bomba. A única forma de sair ileso é cortando o fio correto para desarmar a bomba. Vocês têm 30 segundos até que a bomba exploda!

Neste projeto, vamos criar o jogo Desarme a Bomba com Arduino. Trata-se de um jogo de sorte, você deve tentar desarmar a bomba em menos de 30 segundos, cortando um dos 7 fios conectados a ela.

Há apenas um fio que desarma a bomba, mas há um fio que explode a bomba instantaneamente, também. Tenha cuidado, se você cortar um dos outros cinco fios, seu tempo será reduzido pela metade.

Tenha sorte e ninguém explode!



MATERIAIS NECESSÁRIOS

- 1 x [Placa UNO SMD compatível com Arduino UNO](#);

- 1 x [Cabo USB](#) ;
- 1 x [Protoboard](#) ;
- 2 x [Resistores de 220 \$\Omega\$](#) ;
- 2 x [LEDs difusos de 5 mm](#) ;
- 1 x [Buzzer passivo](#) ;
- 1 x [Botões do tipo push button](#) ;
- 1 x [Resistores de 10 k \$\Omega\$](#) ;
- [Jumpers macho-macho](#) .

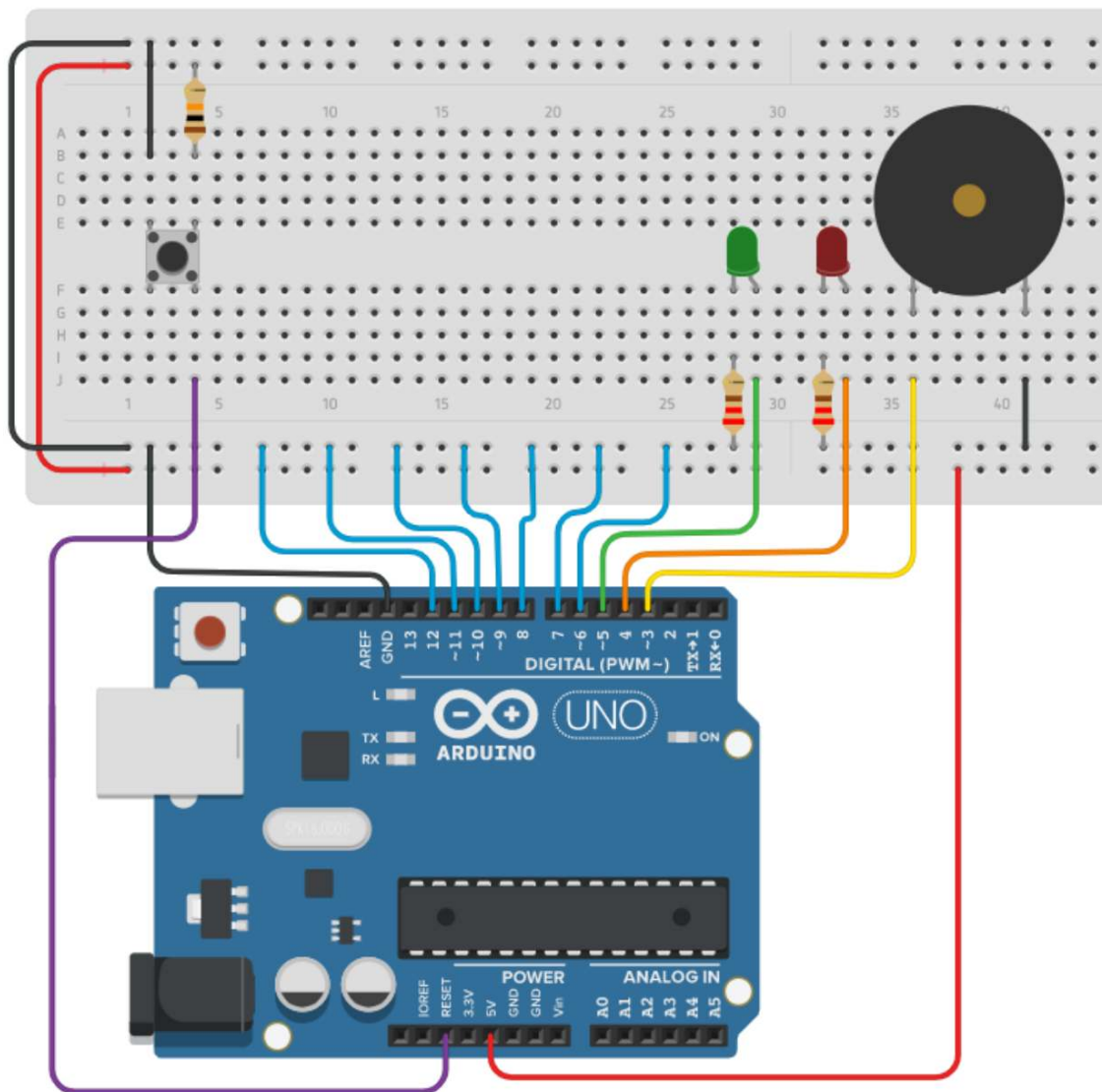
ESQUEMÁTICO DE LIGAÇÃO DOS COMPONENTES

Inicialmente, certifique-se que sua placa UNO esteja desligada. Monte atentamente o circuito conforme a Figura 15 .

Ao montar seu circuito na protoboard preste atenção nos seguintes pontos:

- O terminal positivo do buzzer deve ser conectado à porta digital 3 da placa UNO e o outro terminal deve ser ligado ao GND.
- O terminal mais longo de cada LED deve ser conectado ao pino digital correspondente. O terminal mais curto deve ser conectado ao terra (GND);
- Os jumpers conectados nas portas 6 a 12 do Arduino são correspondentes aos 7 fios conectados à bomba. Eles devem ser conectados ao terra (GND);
- Conecte o push button com um resistor de 10 k Ω pull-up (conectado no VCC) no porta Reset do Arduino. Desta forma, quando o botão for pressionado o Arduino reiniciará.

Figura 15 - Circuito para Projeto Desarme a Bomba.



ELABORANDO O CÓDIGO

Com o circuito montado, vamos a programação do jogo Desarme a Bomba. Para melhor compreensão o código será explicado passo a passo a seguir. Neste momento, observe o código abaixo e aproveite para analisar sua estrutura.

```
int fio[] = {6, 7, 8, 9, 10, 11, 12}; //Portas do Arduino correspondentes aos fios da bomba
int fioUsado[] = {0, 0, 0, 0, 0, 0, 0}; //Array para guardar o fio já retirado
```

```

int ledVerde = 5; //Correspondente ao LED verde
int ledVermelho = 4; //Correspondente ao LED vermelho
int buzzer = 3; //Correspondente ao buzzer
int pino; //Guarda o pino que desarma a bomba
int pinoExplode; //Guarda o pino que explode a bomba

int ledLigaDesliga = 0; //Variável utilizada para guardar o estado do LED (ligado/desligado)
int tempoMaximo = 1800; //Intervalo de tempo inicial da contagem para explosão da bomba
unsigned long millisAnterior = millis (); //variável que recebe o tempo atual em milissegundos

void setup () {

    Serial . begin (9600); //Inicializa a comunicação serial

    pinMode (ledVerde, OUTPUT ); //Define ledVerde como saída
    pinMode (ledVermelho , OUTPUT ); //Define ledVermelho como saída
    pinMode (buzzer, OUTPUT ); //Define buzzer como saída

    digitalWrite (ledVerde, LOW ); //Desliga o LED verde
    digitalWrite (ledVermelho, LOW ); //Desliga o LED vermelho

    for ( int i = 0; i < 9; i++) {
        pinMode (fio[i], INPUT_PULLUP ); //Define os fios da bomba como entrada e em modo
        PULL UP, ou seja, lê HIGH quando não colocamos em nível lógico LOW
    }
    randomSeed ( analogRead (0)); //Inicializa o gerador de número aleatórios, começando em um
    ponto arbitrário

    desarmeAbomba(); //Chama a função desarme a bomba
}

void loop () {} //A função loop está vazia, pois queremos que o jogo funcione uma única vez.
Para jogar novamente basta pressionar o botão conectado ao Reset.

//Função responsável pelo funcionamento do jogo
void desarmeAbomba() {

```

```

Serial . println ( "-*-*-* INICIO -*-*-*" ); //Imprime a mensagem no monitor serial

//Sorteio do fio que desarma a bomba
pino = random (0, 6); //Sorteia um número aleatório entre 0 e 6
Serial . print ( "Pino para desarmar a bomba -> " ); //Imprime a mensagem no monitor serial
Serial . println (fio[pino]); //Imprime no monitor serial o fio sorteado que desarma a bomba

//Sorteio do fio que explode a bomba
do {
    pinoExplode = random (0, 6); //Sorteia novamente um número aleatório entre 0 e 6
    Serial . print ( "Pino que explode a bomba -> " ); //Imprime a mensagem no monitor serial
    Serial . println (fio[pinoExplode]); //Imprime no monitor serial o fio sorteado para explodir a
bomba
} while (pinoExplode == pino); //verifica se o fio que explode a bomba é igual ao que desarma a
bomba. Se sim, repete o laço e sorteia um novo número

//Leitura e verificação dos pinos
do {
    for ( int i = 0; i < 7; i++) { //for que percorre todas as posições do array
        if ( digitalRead (fio[i]) == HIGH && fioUsado[i] == 0) { //Verifica se o pino foi retirado ou
se ele já foi retirado alguma vez
            fioUsado[i] = 1; //Define 1 na posição do vetor para gravar que o pino foi retirado

            if (pinoExplode == i) { //Verifica se o fio retirado é igual ao fio que explode a bomba
                Serial . println ( "Booooo!! A bomba explodiu -*-*-*" ); //Imprime a mensagem no
monitor serial
            }
            else {
                if (pino == i) { //Verifica se o fio retirado é igual ao fio que desarma a bomba
                    Serial . println ( "Ufa, essa foi por um fio. Voce conseguiu desarmar a bomba" );
//Imprime a mensagem no monitor serial
                } else { //Se o fio retirado não for igual ao fio que desarma ou explode a bomba
                    Serial . println ( "Pino errado, tenha cuidado!" ); //Imprime a mensagem no monitor serial
                    tempoMaximo = tempoMaximo / 2; //Reduz o tempo de jogo pela metade
                }
            }
        }
    }
}

```

```

    }
}
if (( millis () - millisAnterior) > tempoMaximo) { //Verifica se o tempo que passou é maior que
o intervalo de tempo determinado para explosão da bomba
    millisAnterior = millis (); //Atualiza para o tempo atual
    tempoMaximo = tempoMaximo - 50; //Reduz o intervalo de tempo em 50 milissegundos
    //Pisca LED verde
    if (ledLigaDesliga) {
        digitalWrite (ledVerde, LOW );
        ledLigaDesliga = 0;
    } else {
        ledLigaDesliga = 1;
        digitalWrite (ledVerde, HIGH );
    }
    tone (buzzer, 493, 100); //Beep para informar que o tempo está diminuindo
}

if (tempoMaximo <= 0) { //Se o tempo máximo for menor ou igual a zero, finaliza o jogo
    Serial . println ( "FIM ....."); //Imprime a mensagem Fim no monitor serial
    digitalWrite (ledVermelho, HIGH ); //O LED vermelho é acionado, indicando que a bomba
explodiu
    tone (buzzer, 493, 1000); //Beep que indica que a bomba explodiu
    delay (1000); //Intervalo para tocar o buzzer
    break ; //Se o tempo máximo for menor ou igual a zero para o laço de repetição
}

//Efeito do buzzer e LEDs quando o jogador consegue desarmar a bomba
if (fioUsado[pino] == 1) { //Verifica se o fio que desarma a bomba foi retirado
    tone (buzzer, 262, 600); delay (300); digitalWrite (ledVerde, LOW );
    tone (buzzer, 294, 400); delay (350); digitalWrite (ledVerde, HIGH );
    tone (buzzer, 330, 300); delay(350); digitalWrite (ledVerde, LOW );
    tone (buzzer, 349, 300); delay (350); digitalWrite (ledVerde, HIGH );
    tone (buzzer, 392, 300); delay (300); digitalWrite (ledVerde, LOW );
    tone (buzzer, 440, 300); delay (300); digitalWrite (ledVerde, HIGH );
    tone (buzzer, 493, 300); delay (200); digitalWrite (ledVerde, LOW );
    break ;
}

//Efeito do buzzer e LEDs quando o jogador explode a bomba

```

```
if (fioUsado[pinoExplode] == 1) { //Verifica se o fio que explode a bomba foi retirado
    digitalWrite (ledVermelho, HIGH );
    tone (buzzer, 493, 1000);
    delay (1000);
    break ;
}
} while (1);
}
```

Para melhor compreensão do código acompanhe os seguintes passos:

1. Declarar as variáveis

Iniciamos a programação declarando as variáveis correspondentes aos fios da bomba. Os fios conectados às portas 6, 7, 8, 9, 10, 11 e 12 foram declarados no array `fio[ ]`.

Um array é uma lista de variáveis que podem ser acessadas com um número índice. As variáveis que encontramos até agora tinham apenas um único valor, usando um array podemos criar uma variável com um conjunto de valores que podem ser acessados dando sua posição dentro da lista.

Em seguida, declaramos o array `fioUsado[ ]`, que será utilizado para armazenar qual fio da bomba já foi retirado ou cortado.

Logo após, declaramos as variáveis `ledVerd` e `ledVermelho` e `buzzer`, correspondentes aos pinos 5, 4 e 3, em que os LEDs e o buzzer estão conectados, respectivamente.

Declaramos também as variáveis `pin o`, para armazenar o fio que desarma a bomba; `pinoExplode` e , para armazenar o fio que explode a bomba; `ledLigaDesliga`, para guardar o estado do LED; `tempoMaximo`, para guardar o tempo até que a bomba exploda; e `millisAnterior`, para retornar o tempo passado desde que o jogo iniciou.

2. Inicialização da comunicação serial e configurações iniciais

A comunicação serial foi inicializada por meio da instrução: `Serial . begin (9600) ; .`

Os pinos de 4 e 5, conectados aos LEDs, e o pino 3, conectado ao buzzer, foram configurados como saída (`OUTPUT`).

Para iniciar o jogo com os LEDs desligados, os definimos como `LOW` .

Também na função `setup ( )` , definimos todas as portas em que os fios estão conectados como entrada pull-up. Fizemos isso, percorrendo os elementos do array e os configurando como entrada pull-up (`INPUT_PULLUP`) com o auxílio da estrutura de repetição `for` .

O Arduino possui resistores pull-up internos que podemos habilitar e usar em nossos projetos. Ao definir um pino como `INPUT_PULLUP` , garantimos que seu estado inicial seja igual a `HIGH` .

Os pinos correspondentes aos fios da bomba foram definidos com `INPUT_PULLUP` e conectados ao GND do Arduino. Assim, quando um fio for retirado ou cortado, seu estado mudará de `LOW` para `HIGH` .

3. Inicialização do gerador de números aleatórios

O gerador de números aleatórios foi inicializado através da função `randomSeed ( analogRead (0)) ;` que utiliza como entrada aleatória a leitura analógica do pino A0.

4. Chamar a função `desarmeAbomba()`

Ainda dentro da função `setup ( )` , chamamos a função `desarmeAbomba( )` , responsável pelo funcionamento do jogo. Assim, o jogo funcionará apenas uma vez.

Para que o jogo seja iniciado novamente, o jogador deverá pressionar o botão conectado ao Reset do Arduino. Por esta razão, a função

`loop()` encontra-se vazia.

5. Sorteio do fio que desarma a bomba

Iniciamos a função `desarmeAbomba()` sorteando um número aleatório entre 0 e 6. Para realização do sorteio usamos a função `random()` ; que gera números aleatórios. Dentro dessa função incluímos o limite mínimo e máximo do valor aleatório (0, 6).

Este número sorteado foi utilizado como índice para acessar o array que guarda os pinos do Arduino correspondente aos fios da bomba. Deste modo, se o número sorteado for igual ao número zero, o fio que desarma a bomba está conectado à porta digital 6, se o número sorteado for igual a 1, o fio que desarma a bomba está conectado à porta digital 7; e assim por diante.

6. Sorteio do fio que explode a bomba e verificar se este fio é diferente do fio que desarma a bomba

Para sortear o fio que explode a bomba também usamos a função `random()` ; . Dentro dessa função incluímos o limite mínimo e máximo do valor aleatório (0, 6).

Novamente, o número sorteado foi utilizado como índice para acessar o array que guarda os pinos do Arduino correspondente aos fios da bomba.

Por meio da estrutura de repetição `do ... while` , verificamos se o fio sorteado para explodir a bomba é igual ao fio sorteado para desarmá-la (`pinoExplode == pino`). Se sim, o sorteio será realizado novamente.

7. Leitura e verificação dos pinos

Com o auxílio da estrutura de repetição `for` percorremos todas as posições do array `fio[ ]`, realizando a sua leitura. Caso algum fio tenha sido retirado ou cortado, sua posição no array `fioUsado[ ]` será alterado para 1.

Em seguida, verificamos se o fio retirado é igual o fio que explode a bomba. Se sim, a mensagem "Booooo!! A bomba explodiu -*-*-*-*" será impressa no monitor serial. Senão, comparamos se ele é igual ao fio que desarma a bomba. Se sim, a mensagem "Ufa, essa foi por um fio. Você conseguiu desarmar a bomba" será mostrada no monitor serial. Se não, o monitor serial imprime a mensagem "Pino errado, tenha cuidado!" e o tempo para desarmar a bomba será reduzido pela metade.

8. Realizar a contagem do tempo usando a função `millis` () e decrementar o tempo

A função `millis` () retorna o número de milissegundos passados desde que a placa microcontroladora começou a executar o programa. Usamos essa função para realizar a contagem do tempo.

A variável `millisAnterior` foi utilizada para guardar o tempo anterior. Verificamos se `millis` () - `millisAnterior` é maior que o intervalo de tempo determinado para a explosão da bomba. Se sim, a variável `millisAnterior` é atualizada com o novo valor de `millis` () e o intervalo de tempo será reduzido em 50 milissegundos. Para sinalizar que o tempo está reduzindo o LED verde pisca e o buzzer emite o beep.

9. Verificar se o tempo de jogo acabou

Se o intervalo de tempo máximo para desarmar a bomba acabar (`tempoMaximo <= 0`), o acionamento do LED vermelho e o beep longo do buzzer sinalizarão que a bomba explodiu.

10. Verificar se o fio que desarma a bomba foi retirado

Usando a condicional **if** , verificamos se o fio retirado ou cortado foi igual ao pino que desarma a bomba (`fioUsado[pino] == 1`). Se sim, o LED verde continua piscando e o buzzer emite uma pequena melodia.

11. Verificar se o fio que explode a bomba foi retirado

Usando a condicional **if** , verificamos se o fio retirado ou cortado foi igual ao pino que explode a bomba (`fioUsado[pinoExplode] == 1`). Se sim, o LED vermelho é acionado e o buzzer emite um beep longo.

TINKERCAD

Para melhor visualizar o circuito deste projeto, basta se acessar o seguinte link : www.blogdarobotica.com/tinkercad-circuitobomba.

Para simular a programação do projeto, acesse o seguinte link: www.blogdarobotica.com/tinkercad-desarmeabomba

Observação : No Tinkercad o botão foi inserido apenas para que a simulação do código pudesse acontecer. Quando pressionado, botão representa o fio retirado ou cortado.

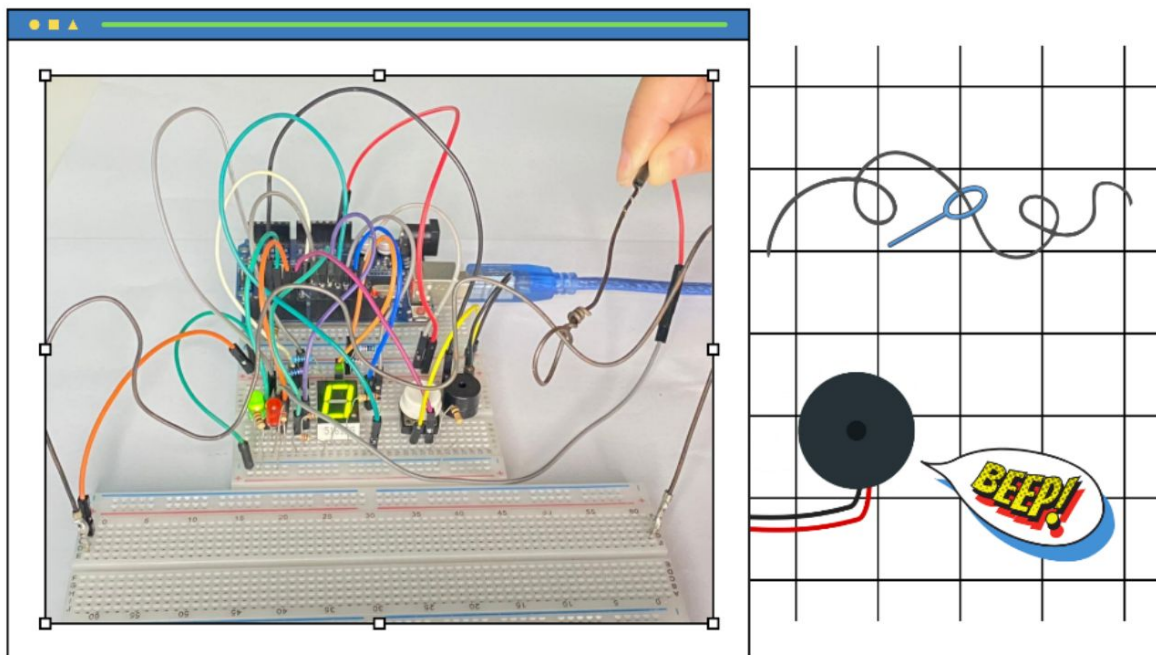
BUZZ WIRE COM CONTADOR

No jogo Buzz Wire, o jogador precisa guiar uma haste de metal ao longo de um circuito de arame curvado sem tocá-lo. Para ganhar, o jogador precisa ter a mão firme para completar o circuito.

Neste projeto, construiremos o jogo Buzz Wire usando Arduino, LEDs, display de 7 segmentos, um botão e um buzzer.

Se o jogador deixar a haste tocar no circuito de arame o buzzer emitirá um beep e o LED vermelho será acionado. O display de 7 segmentos será usado para contar quantas vezes o jogador tocou o circuito de arame.

Você tem a mão firme o suficiente para completar o circuito sem bater no fio e disparar o buzzer? Chame seus amigos e familiares para competir!



MATERIAIS NECESSÁRIOS

- 1 x [Placa UNO SMD compatível com Arduino UNO](#);
- 1 x [Cabo USB](#);
- 2 x [Protoboard](#);
- 1 x [Display de 7 segmentos ânodo comum](#);
- 2 x [LEDs difusos de 5 mm](#);
- 9 x [Resistores de 220 \$\Omega\$](#) ;
- 1 x [Botões do tipo push button](#);
- 2 x [Resistores de 10 k \$\Omega\$](#) ;
- 1 x [Buzzer passivo](#);
- [Jumpers macho-macho](#);
- Arame de metal ou fio desencapado de 0,8 mm.

CONSTRUÇÃO DA HASTE E DO CIRCUITO DE ARAME

Construa a haste (Figura 1 6) com o arame de metal ou o fio desencapado , deixando um gancho circular na ponta.

Para o circuito (Figura 1 7), molde o arame de metal ou o fio desencapado de acordo com sua preferência. Você pode fazer várias ondas e loops.

O nível de dificuldade do jogo é decidido na construção da haste e do circuito. Quanto menor for o gancho na ponta da haste ou quanto mais ondulado for o circuito, mais difícil será atravessar o circuito sem tocá-lo e disparar o buzzer.

Figura 16 – Haste de Arame.

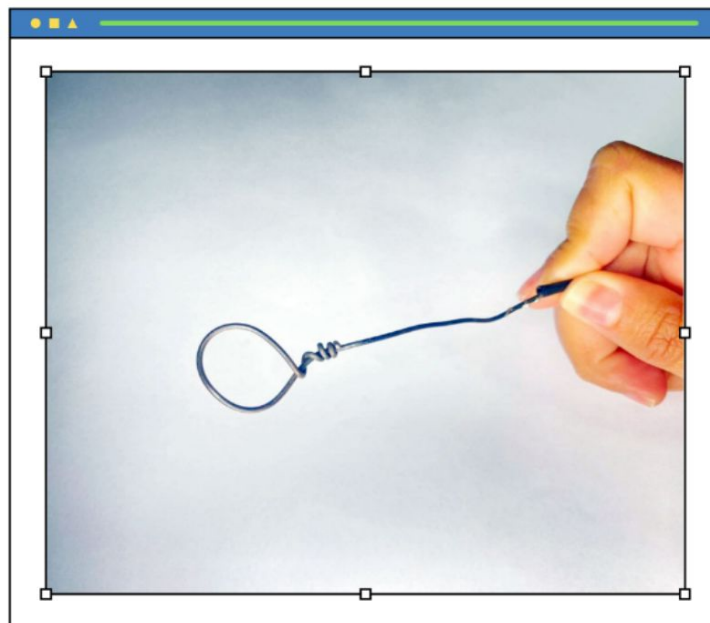
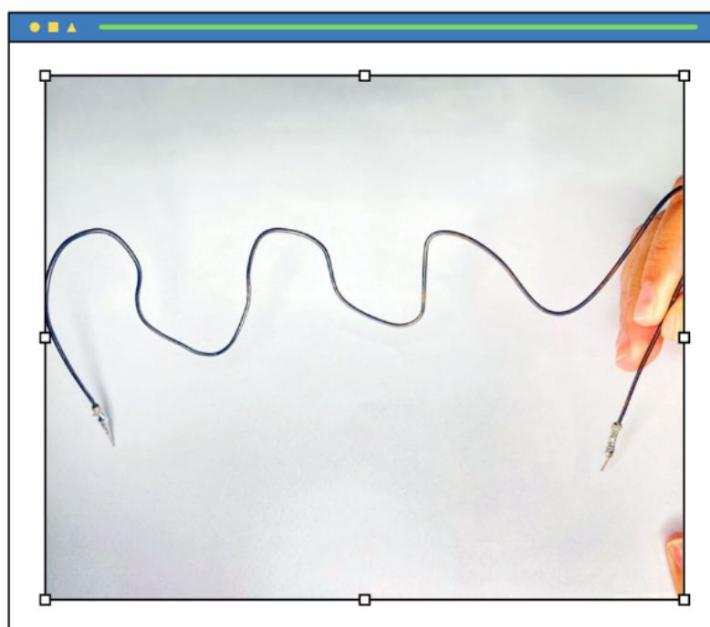


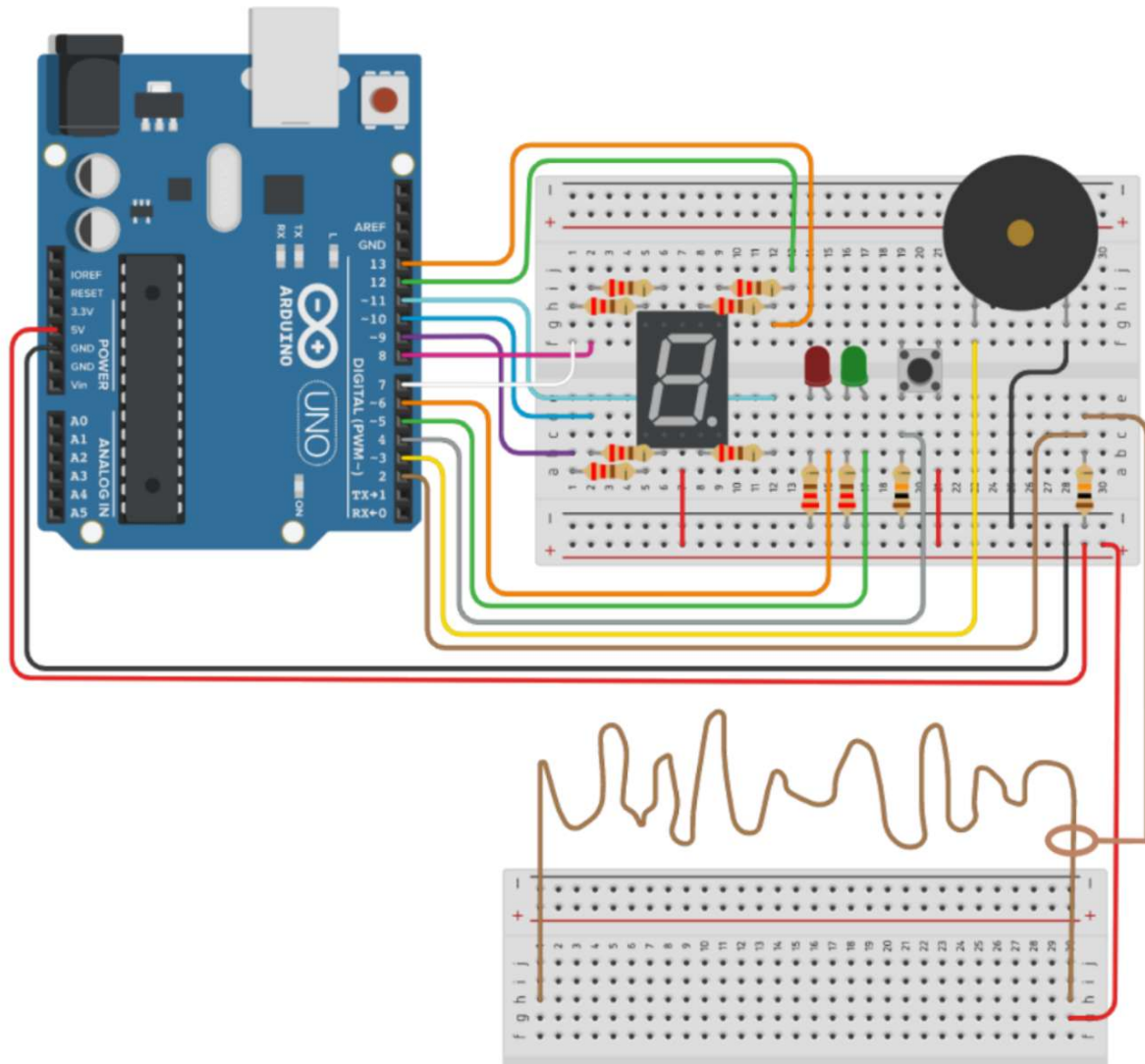
Figura 17 - Circuito de Arame.



ESQUEMÁTICO DE LIGAÇÃO DOS COMPONENTES

Inicialmente, certifique-se que sua placa UNO esteja desligada. Monte atentamente o circuito conforme a Figura 18 .

Figura 18 - Circuito para Projeto Buzz Wire.



Ao montar seu circuito na protoboard, preste atenção nos seguintes pontos:

- O display de 7 segmentos é formado por sete LEDs retangulares dispostos de forma que, por meio da combinação de LEDs ligados e desligados, possa ser mostrada informações alfanuméricas (números e letras);
- Os segmentos do display de 7 segmentos são nomeados de a a g , em sentido horário, a partir do segmento superior do componente;

- O display de 7 segmentos que estamos usando é do tipo ânodo comum, então um dos terminais comuns do mesmo deve ser conectado ao 5 V;
- Vamos conectar um resistor de $220\ \Omega$ em cada um dos segmentos do display;
- Os segmentos encontram-se conectados as portas digitais do Arduino na seguinte ordem: a = 13; b = 12; c = 11; d = 10; e = 9; f = 8; g = 7;
- Dois LEDs serão usados para sinalização. O LED conectado à porta digital 6 será ligado quando a haste tocar o arame do circuito. O LED conectado à porta 5 ficará ligado quando a haste não estiver tocando o arame;
- Em nosso projeto, utilizaremos resistores de $220\ \Omega$ para reduzir a tensão e a corrente de alimentação dos LEDs;
- Um push button será usado para reiniciar o jogo, voltando o contador do display para zero;
- Conecte o push button com um resistor de $10\ k\Omega$ pull-down (resistor conectado no GND). Desta forma, quando o botão estiver pressionado a placa UNO retornará nível lógico alto (HIGH).
- O pino digital 4 será responsável pela leitura do botão;
- O terminal positivo do buzzer deve ser conectado à porta digital 3 da placa UNO e o outro terminal deve ser ligado ao GND;
- A porta digital 2 da placa UNO deve ser conectada a haste de metal e a um resistor de $10\ k\ \Omega$ pull-down (resistor conectado no GND). Desta forma, quando a haste não estiver tocando o arame do circuito o pino digital 2 do Arduino, receberá nível lógico baixo (LOW);

- Em outra protoboard encaixe o circuito de arame e conecte-o a alimentação de 5V (VCC);
- Quando a haste de metal tocar o circuito de arame o pino digital 2 receberá nível alto (HIGH).

ELABORANDO O CÓDIGO

Com a haste e circuito de arame construídos e circuito montado, vamos a programação do jogo. Para melhor compreensão, o código será explicado passo a passo a seguir. Neste momento, observe o código abaixo e aproveite para analisar sua estrutura.

```
//Pinos conectados aos LEDs do display
#define a 13 //Correspondente ao LED a
#define b 12 //Correspondente ao LED b
#define c 11 //Correspondente ao LED c
#define d 10 //Correspondente ao LED d
#define e 9 //Correspondente ao LED e
#define f 8 //Correspondente ao LED f
#define g 7 //Correspondente ao LED g

//Pinos conectados aos LEDs
#define ledVermelho 6 //Correspondente ao LED vermelho
#define ledVerde 5 //Correspondente LED verde

#define botao 4 //Correspondente ao botão
#define buzzer 3 // Correspondente ao buzzer
#define haste 2 //Correspondente a haste de metal

int botaoLeitura = 0; //Armazena a leitura do pino conectado ao botão

int leitura = 0; //Armazena a leitura do pino conectado a haste
```

```

int ultleitura = 0; //Armazena a última leitura do pino conectado a haste
int contador = 0; //Correspondente ao contador

void setup () {

    Serial . begin (9600); //Inicia a comunicação serial

    //Configura os pinos conectados aos LEDs do display, os LEDs e o buzzer como saída
    pinMode (a, OUTPUT ); //Define a como saída
    pinMode (b, OUTPUT ); //Define b como saída
    pinMode (c, OUTPUT ); //Define c como saída
    pinMode (d, OUTPUT ); //Define d como saída
    pinMode (e, OUTPUT ); //Define e como saída
    pinMode (f, OUTPUT ); //Define f como saída
    pinMode (g, OUTPUT ); //Define g como saída
    pinMode (ledVerde, OUTPUT ); //Define ledVerde como saída
    pinMode (ledVermelho, OUTPUT ); //Define ledVermelho como saída
    pinMode (buzzer, OUTPUT ); //Define buzzer como saída

    //Configura os pinos conectados ao botão e a haste como entrada
    pinMode (botao, INPUT ); //Define botao como entrada
    pinMode (haste, INPUT ); //Define haste como entrada
}

void loop () {
    digitalWrite (ledVerde, HIGH ); //Liga o LED verde
    digitalWrite (ledVermelho, LOW ); //Desliga o LED vermelho

    botaoLeitura = digitalRead (botao); //Guarda a leitura do botao na variável botaoLeitura
    if (botaoLeitura == HIGH ) { //Se o botão for apertado
        contador = 0; //Zera o contador
    }

    leitura = digitalRead (haste); // Lê o estado de buttonPin e armazena em leitura
    Serial . println (leitura); //Imprime na serial o conteúdo de leitura

```

```
if (leitura != ultleitura) { // Compara a leitura de haste com a leitura anterior
  if (leitura == HIGH ) { //Se leitura for igual a HIGH
    digitalWrite (ledVerde, LOW ); //Desliga o LED verde
    digitalWrite (ledVermelho, HIGH ); //Liga o LED vermelho
    tone (buzzer, 1500); //Liga o buzzer com uma frequência de 1500 Hz.
    delay (500); //Intervalo de 500 milissegundos
    noTone (buzzer); //Desliga o buzzer.
    contador++; //Incrementa contador em 1
  }
}
ultleitura = leitura; //Atribui a ultleitura o conteúdo de leitura
```

```
switch (contador) {
  case 0:
    zero(); //Executa a função zero
    break ;
  case 1:
    um(); //Executa a função um
    break ;
  case 2:
    dois(); //Executa a função dois
    break ;
  case 3:
    tres(); //Executa a função três
    break ;
  case 4:
    quatro(); //Executa a função quatro
    break ;
  case 5:
    cinco(); //Executa a função cinco
    break ;
  case 6:
    seis(); //Executa a função seis
    break ;
  case 7:
    sete(); //Executa a função sete
    break ;
}
```

```

    case 8:
        oito(); //Executa a função oito
        break ;
    case 9:
        nove(); //Executa a função nove
        break ;
}

if (contador >= 10) { //Se contador for maior ou igual a 10
    contador = 0; //Retorna contador a zero
}
}

//Funções para escrever os números no display de 7 segmentos
//Função para escrever o nº zero
void zero() {
    digitalWrite (a, 0); //coloca a em nível baixo
    digitalWrite (b, 0); //coloca b em nível baixo
    digitalWrite (c, 0); //coloca c em nível baixo
    digitalWrite (d, 0); //coloca d em nível baixo
    digitalWrite (e, 0); //coloca e em nível baixo
    digitalWrite (f, 0); //coloca f em nível baixo
    digitalWrite (g, 1); //coloca g em nível alto
    delay (100);
}

//Função para escrever o nº um
void um() {
    digitalWrite (a, 1);
    digitalWrite (b, 0);
    digitalWrite (c, 0);
    digitalWrite (d, 1);
    digitalWrite (e, 1);
    digitalWrite (f, 1);
    digitalWrite (g, 1);
    delay (100);
}

//Função para escrever o nº dois
void dois() {

```

```
digitalWrite (a, 0);
digitalWrite (b, 0);
digitalWrite (c, 1);
digitalWrite (d, 0);
digitalWrite (e, 0);
digitalWrite (f, 1);
digitalWrite (g, 0);
delay (100);
}
//Função para escrever o nº três
void tres() {
digitalWrite (a, 0);
digitalWrite (b, 0);
digitalWrite (c, 0);
digitalWrite (d, 0);
digitalWrite (e, 1);
digitalWrite (f, 1);
digitalWrite (g, 0);
delay (100);
}
//Função para escrever o nº quatro
void quatro() {
digitalWrite (a, 1);
digitalWrite (b, 0);
digitalWrite (c, 0);
digitalWrite (d, 1);
digitalWrite (e, 1);
digitalWrite (f, 0);
digitalWrite (g, 0);
delay (100);
}
//Função para escrever o nº cinco
void cinco() {
digitalWrite (a, 0);
digitalWrite (b, 1);
digitalWrite (c, 0);
digitalWrite (d, 0);
```

```
digitalWrite (e, 1);
digitalWrite (f, 0);
digitalWrite (g, 0);
delay (100);
}
//Função para escrever o nº seis
void seis() {
digitalWrite (a, 0);
digitalWrite (b, 1);
digitalWrite (c, 0);
digitalWrite (d, 0);
digitalWrite (e, 0);
digitalWrite (f, 0);
digitalWrite (g, 0);
delay (100);
}
//Função para escrever o nº sete
void sete() {
digitalWrite (a, 0);
digitalWrite (b, 0);
digitalWrite (c, 0);
digitalWrite (d, 1);
digitalWrite (e, 1);
digitalWrite (f, 1);
digitalWrite (g, 1);
delay (100);
}
//Função para escrever o nº oito
void oito() {
digitalWrite (a, 0);
digitalWrite (b, 0);
digitalWrite (c, 0);
digitalWrite (d, 0);
digitalWrite (e, 0);
digitalWrite (f, 0);
digitalWrite (g, 0);
delay (100);
```

```
}  
//Função para escrever o nº nove  
void nove() {  
    digitalWrite (a, 0);  
    digitalWrite (b, 0);  
    digitalWrite (c, 0);  
    digitalWrite (d, 0);  
    digitalWrite (e, 1);  
    digitalWrite (f, 0);  
    digitalWrite (g, 0);  
    delay (100);  
}
```

Para melhor compreensão do código acompanhe os seguintes passos:

1. Declarar as variáveis

Iniciamos a programação declarando as variáveis correspondentes aos LEDs do display de 7 segmentos. Conforme o esquemático elétrico, atribuiremos a porta 13 a variável **a** ; a porta 12 a variável **b** ; a porta 11 a variável **c** ; a porta 10 a variável **d** ; a porta 9 a variável **e** ; a porta 8 a variável **f** ; a porta 7 a variável **g** .

Em seguida, declaramos 2 variáveis para os LEDs (ledVermelho e ledVerde), uma variável para o botão (botao), uma variável para o buzzer (buzzer) e outra para a haste de metal (haste).

Além destas, declaramos também uma variável para armazenar a leitura do botão (botaoLeitura), uma para armazenar a leitura do pino conectado a haste (leitura), uma para guardar a última leitura do pino conectado a haste (ultleitura) e uma correspondente ao contador (contador).

2. Inicialização da comunicação serial e configuração das portas de entrada e saída

A comunicação serial foi inicializada por meio da instrução: `Serial . begin (9600) ;` .

Configuramos como saída (`OUTPUT`) as variáveis correspondentes aos LEDs do display de 7 segmentos, LEDs e buzzer.

As variáveis `botao` e `haste` foram definidas como entradas (`INPUT`).

3. Definir LED verde como ligado e LED vermelho como desligado

No jogo, os LEDs serão usados como sinalização. O LED verde ficará ligado se a haste não estiver tocando o arame. O LED vermelho será ligado quando a haste tocar o arame do circuito.

Desse modo, iniciamos a função `loop` () colocando a variável `ledVerde` em nível alto (`HIGH`), ligando o LED verde, e a variável `ledVermelho` em nível baixo (`LOW`), desligando o LED vermelho.

4. Realizar a leitura do botão e verificar se ele foi pressionado

Em seguida, realizamos a leitura do botão, variável `botao` , e atribuímos o resultado da leitura a variável `botaoLeitura` . Isso foi feito através da instrução `botaoLeitura = digitalRead (botao) ;` .

Em seguida, verificamos se o botão foi pressionado (`if ( botaoLeitura == HIGH )`). Sendo a condição satisfeita, a variável `contador` será zerada (`contador = 0 ;`), reiniciando o jogo.

5. Realizar a leitura do pino conectado a haste e imprimir no monitor serial

De mesmo modo, vamos realizar a leitura do pino conectado a haste, variável `haste` , e atribuir o resultado da leitura a variável `leitura` .

Incluimos a instrução `Serial.println(leitura)`; para que seja exibido seu valor no monitor serial. Com isso, podemos testar de fato se o Arduino recebe nível lógico baixo (zero) quando a haste não está tocando o circuito de arame e se esse valor muda para 1 quando ela toca o circuito.

6. Comparar o valor de leitura com o valor de ultleitura

Precisamos saber quantas vezes a haste tocou no circuito de arame e quantas mudanças de estado isto ocasionou. Para fazer essa detecção, precisamos comparar o estado de leitura do pino conectado a haste com a sua leitura anterior. Fizemos isso através das seguintes instruções:

```
if (leitura != ultleitura) { // Compara a leitura de haste com a
leitura anterior
    if (leitura == HIGH ) { //Se leitura for igual a HIGH
        digitalWrite (ledVerde, LOW ); //Desliga o LED verde
        digitalWrite (ledVermelho, HIGH ); //Liga o LED vermelho
        tone (buzzer, 1500); //Liga o buzzer com uma frequência de
1500 Hz.
        delay (500); //Intervalo de 500 milissegundos
        noTone (buzzer); //Desliga o buzzer.
        contador++; //Incrementa contador em 1
    }
}
```

7. Verificar o valor de contador

Utilizaremos a estrutura de controle de fluxo de seleção `switch ... case` para comparar o valor armazenado na variável `contador` aos números especificados nos comandos `case`. Quando o valor armazenado for igual ao caractere especificado no `case`, a função para exibição do número correspondente deve ser executada.

8. Comparar se o valor de contador é maior ou igual a 10

Como estamos utilizando apenas um display de 7 segmentos, apenas podemos exibir números de 0 a 9. Desta forma precisamos limitar o valor de contador. Sempre que contador assumir um valor maior ou igual a 10, seu valor será zerado.

9. Criar as funções para escrever os números no display

Para que os números sejam exibidos no display de 7 segmentos, criamos uma função para cada número. Cada função contém a combinação de segmentos que devem ser acionados (colocados em nível baixo) para exibir o número desejado. Na Tabela estão listados os segmentos que devem ser acionados para formação dos números de 0 a 9.

Tabela 7 - Número e sequência de segmentos.

Número	Sequência
0	a, b, c, d, e, f
1	b, c
2	a, b, d, e, g
3	a, b, c, d, g
4	b, c, f, g
5	a, c, d, f, g
6	a, c, d, e, f, g
7	a, b, c
8	a, b, c, d, e, f, g
9	a, b, c, f, g

Por exemplo, a função zero() contém todas as instruções para acionamento dos segmentos do display exibindo o número zero. Conforme a tabela, precisamos colocar em nível baixo as variáveis **a**, **b**, **c**, **d**, **e**, e **f** e colocar em nível alto a variável **g**. A função zero() foi escrita da seguinte forma:

```
//Função para escrever o nº zero
void zero() {
  digitalWrite (a, 0); //coloca a em nível baixo
  digitalWrite (b, 0); //coloca b em nível baixo
  digitalWrite (c, 0); //coloca c em nível baixo
  digitalWrite (d, 0); //coloca d em nível baixo
  digitalWrite (e, 0); //coloca e em nível baixo
  digitalWrite (f, 0); //coloca f em nível baixo
  digitalWrite (g, 1); //coloca g em nível alto
  delay (100);
}
```

TINKERCAD

Para melhor visualizar e/ou simular o circuito e a programação deste projeto, acesse o seguinte link: www.blogdarobotica.com/tinkercad-buzzwire.

Observação : No Tinkercad o botão foi inserido apenas para que a simulação do código pudesse acontecer. Neste caso, o botão pressionado simula o toque da haste no circuito de arame.

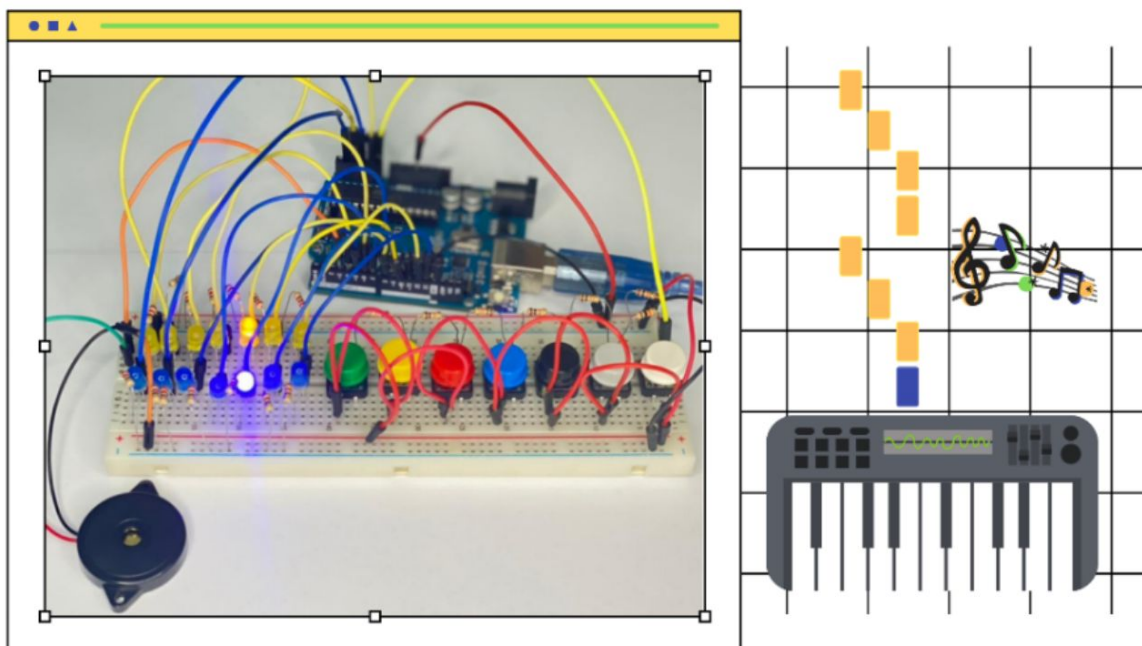
OceanofPDF.com

PIANO HERO

O Piano Hero é inspirado no Guitar Hero, popular jogo de vídeo game. No Guitar Hero, o jogador deve pressionar os botões do controle de acordo com as notas coloridas que aparecem na tela. Substituiremos a tela por LEDs, os botões do controle por botões push button e o vídeo game pelo Arduino.

No Piano Hero, cada um dos sete botões representa uma nota musical. Os LEDs azuis representam qual nota deverá ser tocada. O jogador deve pressionar o botão da nota sinalizada pelo LED azul. Por exemplo, se o LED azul ligado é o primeiro à esquerda, o jogador deverá tocar o primeiro botão à esquerda (nota Dó). Os LEDs amarelos indicam a próxima nota que será sinalizada pelos LEDs azuis.

Nesta versão, o Piano Hero com Arduino traz a música infantil Cai, Cai, Balão.



MATERIAIS NECESSÁRIOS

- 1 x [Placa UNO SMD compatível com Arduino UNO](#);
- 1 x [Cabo USB](#);
- 2 x [Protoboard](#);
- 14 x [LEDs difusos de 5 mm](#);
- 14 x [Resistores de 220 \$\Omega\$](#) ;
- 7 x [Botões do tipo push button](#);
- 7 x [Resistores de 10 k \$\Omega\$](#) ;
- 1 x [Buzzer passivo](#);
- [Jumpers macho-macho](#);

ESQUEMÁTICO DE LIGAÇÃO DOS COMPONENTES

Certifique-se que sua placa UNO esteja desligada e monte atentamente o circuito conforme a Figura 19.

Ao montar seu circuito na protoboard, preste atenção nos seguintes pontos:

- O terminal mais curto do LED é o cátodo (negativo) e deve ser conectado ao terra (GND). O terminal mais longo de cada LED deve ser conectado ao pino correspondente. Observe que alguns LEDs serão conectados aos pinos analógicos.
- Em nosso projeto, utilizaremos resistores de 220 Ω para reduzir a tensão e a corrente de alimentação dos LEDs;
- O botão deve ser montado com dois terminais abaixo da cavidade central da protoboard e os outros dois acima;
- Para ler os botões usaremos uma única entrada analógica do Arduino, utilizando como estratégia um divisor de tensão. Assim, quando pressionado cada botão terá em sua saída níveis de tensão diferentes.

- Figura 19 - Circuito para Projeto Piano Hero.

ELABORANDO O CÓDIGO

O uso de apenas uma porta analógica do Arduino para realizar a leitura de botões é bastante útil. Como a placa Uno possui apenas 14 portas digitais, em projetos maiores, como o Piano Hero, podemos ficar sem pinos digitais.

O pino analógico do nosso Arduino é conectado a um conversor analógico-digital (ADC), com resolução de dez bits. Assim, o pino

analógico retorna um valor numérico de 0 a 1023, que se refere a faixa de tensão lida (entre 0 e 5 V).

O divisor de tensão utilizado no circuito dos botões permite que cada botão retorne um nível de tensão diferente. Deste modo, para encontrar o valor de referência dos botões, vamos apenas ler o valor analógico para cada botão pressionado, conforme o código abaixo.

```
int botao = A0; //Correspondente a leitura dos botões

void setup ( ) {
  Serial . begin (9600); //Inicia a comunicação Serial
  pinMode (botao, INPUT ); //Define botao como entrada
}

void loop () {
  Serial . println ( analogRead (botao)); //Realiza a leitura
analógica do botão e imprime na serial
}
```

Após transferir este código para o seu Arduino, abra o monitor serial e pressione os botões em sequência. Inicie pelo primeiro botão à esquerda, ele representa a nota Dó, o segundo a nota Ré, e assim por diante.

Anote os valores mostrados no monitor serial com suas correspondentes notas, pois vamos utilizá-los no código final do projeto Piano Hero.

Em nosso teste, obtemos os seguintes resultados:

Nota Musical	Valor analógico lido
Dó	145
Ré	169
Mi	203
Fá	254
Sol	340
Lá	508

Os seus resultados podem ser diferentes, por conta dos valores das resistências.

De posse desses resultados, podemos elaborar o código para o projeto Piano Hero. Para melhor compreensão, o código será explicado passo a passo a seguir. Neste momento, observe o código abaixo e aproveite para analisar sua estrutura.

```
int ledAmarelo[] = {A1, A3, 4, 6, 8, 10, 12}; //LEDs da fileira
amarela(nota anterior)
int ledAzul[] = {A2, A4, 5, 7, 9, 11, 13}; //LEDs da fileira azul
(nota a ser tocada)
int botao = A0; //Correspondente a leitura dos botões
int buzzer = 3; //Correspondente ao buzzer
int leitura = 0; //Armazena a leitura do pino A0 conecta aos
botões

int quantidade_Notas = 30; //Quantidade de notas totais da música
cai, cai balão

//Matriz que guarda a sequência das notas da música cai, cai balão
String musica[] = { "SOL" ,
                    "SOL" ,
                    "FA" ,
                    "MI" ,
                    "SOL" ,
                    "SOL" ,
                    "FA" ,
                    "MI" ,
                    "SOL" ,
                    "LA" ,
```

```
"SOL" ,  
"FA" ,  
"MI" ,  
"RE" ,  
"RE" ,  
"MI" ,  
"FA" ,  
"RE" ,  
"MI" ,  
"FA" ,  
"RE" ,  
"MI" ,  
"FA" ,  
"SOL" ,  
"LA" ,  
"SOL" ,  
"FA" ,  
"MI" ,  
"RE" ,  
"DO"  
};
```

```
void setup () {  
  pinMode (botao, INPUT ); //Define botao como entrada  
  pinMode (buzzer, OUTPUT ); //Define buzzer como saída  
  
  //Define os LEDs da fileira amarela e azul como saída  
  for ( int i = 0; i < 7; i++) {  
    pinMode (ledAmarelo[i], OUTPUT );  
    pinMode (ledAzul[i], OUTPUT );  
  }  
  //Desliga todos os LEDs da fileira amarela e azul  
  for ( int i = 0; i < 7; i++) {  
    digitalWrite (ledAmarelo[i], LOW );  
    digitalWrite (ledAzul[i], LOW );  
  }  
}
```

```

void loop () {
    //Laço de repetição para percorrer todas as notas do array de
    notas
    for ( int j = 0; j < quantidade_Notas; j++){ //Repete as
    instruções enquanto j for menor que 30

    // Liga LED azul e toca a nota correspondente...
    if (musica[j]. equals ( "DO" )) { //Condição que verifica se a
    nota da matriz é um Dó
        digitalWrite (ledAzul[0], HIGH ); //Se sim, liga o LED azul
        correspondente a nota Dó
        do {
            leitura = analogRead (botao); //Realiza a leitura do canal
            analógico conectado aos botões
            if (leitura == 145) { //Verifica se o valor lido foi igual ao
            número correspondente do botão Dó
                break ; //Se sim, para o laço de repetição
            }
        } while (1); //Enquanto não for pressionado o botão, o laço de
        repetição continua
        tone (buzzer, 262, 600); //Toca a nota Dó
        delay (400); //Intervalo necessário para tocar a nota Dó
    }

    if (musica[j]. equals ( "RE" )) {
        digitalWrite (ledAzul[1], HIGH );
        do {
            leitura = analogRead (botao);
            if (leitura == 169) {
                break ;
            }
        } while (1);
        tone (buzzer, 294, 300);
        delay (300);
    }
}

```

```
if (musica[j]. equals ( "MI" )) {  
    digitalWrite (ledAzul[2], HIGH );  
    do {  
        leitura = analogRead (botao);  
        if (leitura == 203) {  
            break ;  
        }  
    } while (1);  
    tone (buzzer, 330, 300);  
    delay (300);  
}
```

```
if (musica[j]. equals ( "FA" )) {  
    digitalWrite (ledAzul[3], HIGH );  
    do {  
        leitura = analogRead (botao);  
        if (leitura == 254) {  
            break ;  
        }  
    } while (1);  
    tone (buzzer, 349, 300);  
    delay (300);  
}
```

```
if (musica[j]. equals ( "SOL" )) {  
    digitalWrite (ledAzul[4], HIGH );  
    do {  
        leitura = analogRead (botao);  
        if (leitura == 340) {  
            break ;  
        }  
    } while (1);  
    tone (buzzer, 392, 300);  
    delay (300);  
}
```

```
if (musica[j]. equals ( "LA" )) {
```

```
digitalWrite (ledAzul[5], HIGH );  
do {  
  leitura = analogRead (botao);  
  if (leitura == 508) {  
    break ;  
  }  
} while (1);  
tone (buzzer, 440, 300);  
delay (300);  
}
```

```
if (musica[j]. equals ( "SI" )) {  
  digitalWrite (ledAzul[6], HIGH );  
  do {  
    leitura = analogRead (botao);  
    if (leitura == 1020) {  
      break ;  
    }  
  } while (1);  
  tone (buzzer, 493, 300);  
  delay (300);  
}
```

// Liga LED Amarelo...

```
if ( j < quantidade_Notas - 1) { //Condição responsável por  
acionar o LED amarelo até a penúltima nota
```

```
  if (musica[j + 1]. equals ( "DO" )) { //Condição que verifica  
se a próxima nota da matriz é um Dó
```

```
    digitalWrite (ledAmarelo[0], HIGH ); //Se sim, liga o LED  
amarelo correspondente a nota Dó
```

```
  }
```

```
  if (musica[j + 1]. equals ( "RE" )) {  
    digitalWrite (ledAmarelo[1], HIGH );  
  }
```

```
  if (musica[j + 1]. equals ( "MI" )) {  
    digitalWrite (ledAmarelo[2], HIGH );
```

```

    }
    if (musica[j + 1]. equals ( "FA" )) {
        digitalWrite (ledAmarelo[3], HIGH );
    }
    if (musica[j + 1]. equals ( "SOL" )) {
        digitalWrite (ledAmarelo[4], HIGH );
    }
    if (musica[j + 1]. equals ( "LA" )) {
        digitalWrite (ledAmarelo[5], HIGH );
    }
    if (musica[j + 1]. equals ( "SI" )) {
        digitalWrite (ledAmarelo[6], HIGH );
    }
}

//Desliga todos os LEDs da fileira amarela e azul
for ( int i = 0; i < 7; i++) {
    digitalWrite (ledAmarelo[i], LOW );
    digitalWrite (ledAzul[i], LOW );
}
}
}

```

Para melhor compreensão do código acompanhe os seguintes passos:

1. Declarar as variáveis

Iniciamos a programação declarando as variáveis correspondentes aos LEDs. Os LEDs da fileira amarela (portas A1, A3, 4, 6, 8, 10 e 12) foram declarados no array `ledAmarelo[ ]` e os da fileira azul (portas A2, A4, 5, 7, 11 e 13) foram declarados no array `ledAzul[ ]`.

Em seguida, declaramos as variáveis `bota o` e `buzzer`, correspondentes aos pinos A0 e digital 3, em que os botões e o buzzer estão conectados, respectivamente.

Declaramos também a variável `leitura` para armazenar a leitura do pino A0 e a variável `quantidade_Notas` para guardar a quantidade de notas totais da música caí, caí balão.

No array `musica[ ]` escrevemos todas as notas em sequência para tocar a música caí, caí balão.

2. Configurações iniciais do projeto

Na função `setup ( )`, realizamos as configurações iniciais do nosso projeto. Inicialmente, definimos as portas de entrada e saída. A variável `bota_o` foi definida como entrada e a variável `buzzer` foi configurada como saída.

As portas correspondentes aos LEDs azuis e amarelos também foram declaradas como saída. Fizemos isso percorrendo os elementos do array e os configurando como saída (`OUTPUT`), com o auxílio da estrutura de repetição `for`.

Ainda na função `setup ( )`, vamos também desligar todos os LEDs. Fizemos isso percorrendo os elementos do array e os configurando como desligado (`LOW`), com o auxílio da estrutura de repetição `for`.

3. Percorrer todas as notas música caí, caí balão guardadas no array

Para que os LEDs azuis indiquem qual a nota deve ser tocada e os LEDs amarelos sinalizem a próxima nota a ser tocada, vamos percorrer todas as posições do array `musica[ ]`. Para isso, usamos a estrutura de repetição `for`, em que definimos a variável inteira `j = 0` na inicialização, testamos se `j` é um número menor que 30 (armazenado na variável `quantidade_Notas`) na condição, e somamos `j++` no incremento.

4. Verificar nota a ser tocada, acionar o LED azul correspondente a nota a ser tocada e realizar a leitura do botão

Para verificar a nota a ser tocada, acionar o LED azul correspondente e ler o botão usamos 7 estruturas de seleção `if`, uma para cada nota musical.

Por meio da instrução `if (musica[j]. equals ( "DO" ) )`, verificamos se a string da matriz `musica[ ]` na posição `j` é igual a nota musical Dó. A função `equals ( )` verifica se duas strings são iguais.

Se a condição for verdadeira (string de `musica[ ]` igual a Dó), o LED azul correspondente a nota Dó será acionado, ou seja, o primeiro LED à esquerda e o programa entra na estrutura de repetição `do ... while`. Dentro da estrutura `do ... while`, realizamos a leitura do botão e comparamos se o valor lido é igual ao valor analógico lido do botão correspondente à nota Dó (`if (leitura == 145 )`).

OBSERVAÇÃO : Substitua 145 pelo valor analógico obtido ao pressionar o botão correspondente a nota Dó, que encontramos no código de teste dos botões (valor de referência).

Quando o jogador pressiona o botão correspondente a nota Dó, o programa sai do laço de repetição e o buzzer toca a nota Dó. No entanto, se ele pressionar o botão errado nenhum som será emitido.

Repetimos as mesmas instruções para as outras notas musicais.

5. Verificar próxima nota a ser tocada e acionar o LED amarelo correspondente a nota

De mesmo modo, para verificar a próxima nota a ser tocada e acionar o LED amarelo correspondente, usamos 7 estruturas de seleção `if`, uma para cada nota musical.

Através da instrução `if (musica[j+1]. equals ( "DO" ) )` , verificamos se a string da matriz `musica[ ]` na posição `j+ 1` é igual a nota musical Dó. Em outras palavras, verificamos se a próxima posição do array `musica[ ]` é igual a Dó.

Se a condição for verdadeira (string de `musica[ ]` igual a Dó), o LED amarelo correspondente a nota Dó será acionado, ou seja, o primeiro LED à esquerda.

Repetimos as mesmas instruções para as outras notas musicais.

6. Desligar todos os LEDs para reiniciar o jogo

Quando a variável `j` assumir um valor maior que a variável `quantidade_Notas` , o programa sai do primeiro laço de repetição `for` , indicando que todas as notas da música cá, cá balão foram tocadas.

Para finalizar o código desligamos todos os LEDs azuis e amarelos. Fizemos isso percorrendo os elementos dos arrays e os configurando como desligado (`LOW`), com o auxílio da estrutura de repetição `for` .

TINKERCAD

Para melhor visualizar e/ou simular o circuito e as programações deste projeto, basta acessar os seguintes links:

Teste dos botões: www.blogdarobotica.com/tinkercad-botao-pianohero

Piano Hero completo: www.blogdarobotica.com/tinkercad-pianohero

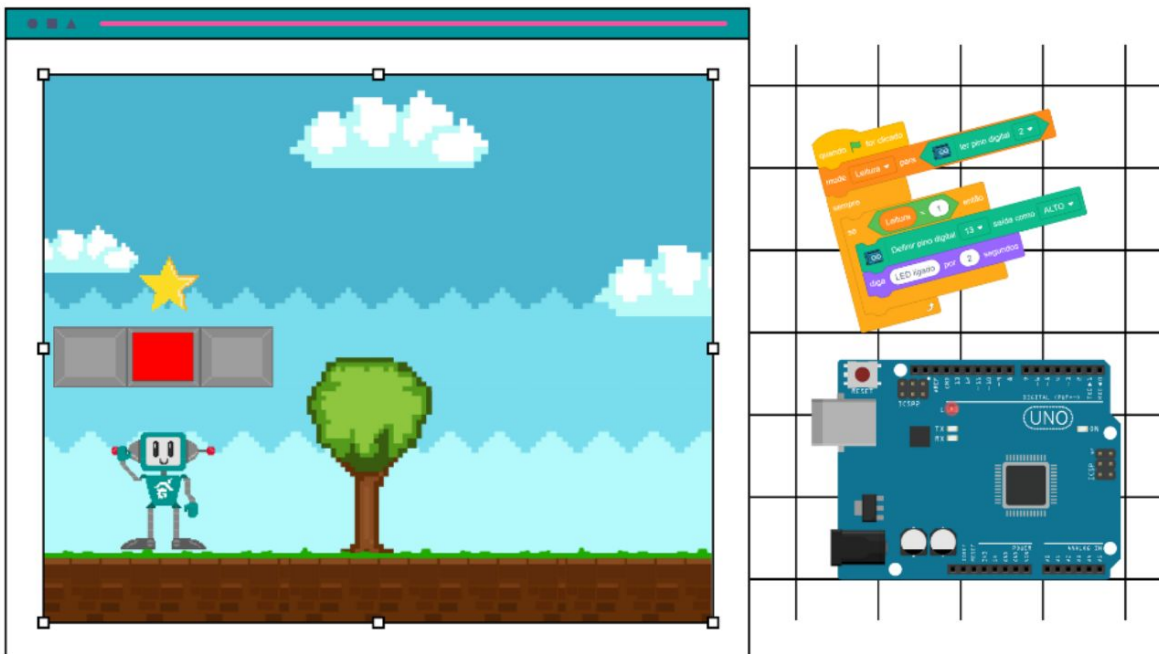
SUPER CDRBOT

Você já pensou em desenvolver seu próprio jogo de vídeo game? Se a resposta foi sim, prepare-se, pois esse projeto foi feito para você. Vamos criar o Super CDRbot.

O Super CDRbot é um jogo inspirado no Super Mario World, famoso jogo de plataforma dos anos 90 produzido pela Nintendo. O CDRbot é o nosso mascote, um robzinho simpático e alegre!

O CDRbot deve saltar os obstáculos até que o tijolo surpresa apareça no cenário. Ao bater nesse tijolo, uma estrela surge piscando. Para vencer, o CDRbot deve tocar na estrela.

Vamos controlar o salto do CDRbot com botão no Arduino. Com o LED flash rápido, vamos indicar no mundo físico que o jogador ganhou.



Para desenvolvê-lo, vamos utilizar o Pictoblox, software de programação em blocos baseado em Scratch, que permite a interação com Arduino.

O interessante aqui é introduzir a possibilidade de unir o hardware com o software, o que torna o aprendizado muito mais prazeroso. Divirta-se!

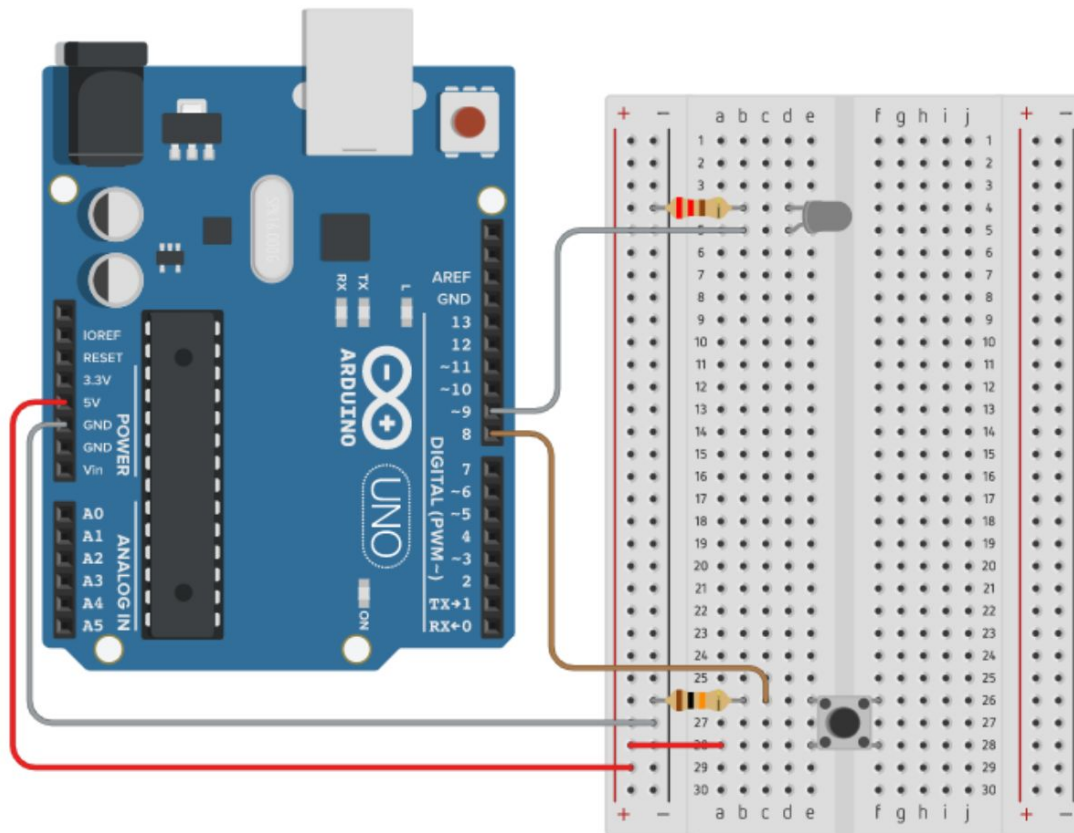
MATERIAIS NECESSÁRIOS

- 1 x [Placa UNO SMD compatível com Arduino UNO](#);
- 1 x [Cabo USB](#);
- 1 x [Protoboard](#);
- 1 x [LED flash rápido](#);
- 1 x [Botões do tipo push button](#);
- 1 x [Resistores de 220 \$\Omega\$](#) ;
- 1 x [Resistores de 10 k \$\Omega\$](#) ;
- [Jumpers macho-macho](#);

ESQUEMÁTICO DE LIGAÇÃO DOS COMPONENTES

Certifique-se que sua placa UNO esteja desligada e monte atentamente o circuito conforme a Figura 20 .

Figura 20 - Circuito para o Projeto Super CDRbot



Ao montar seu circuito na protoboard, preste atenção nos seguintes pontos:

- O botão encontra-se conectado ao pino digital 8 da placa UNO. Esse botão será responsável por movimentar o CDRbot. Quando o botão for pressionado, o CDRbot deve pular;
- O LED flash rápido encontra-se conectado ao pino digital 9 do Arduino. Quando o CDRbot conseguir saltar todos os obstáculos, um tijolo surpresa surge no cenário. Ao bater nesse tijolo, uma estrela surge piscando, indicando que o jogador venceu o jogo. Com o LED flash rápido, vamos indicar no mundo físico que o jogador ganhou.

TINKERCAD

Para melhor visualizar o circuito deste projeto, acesse o seguinte link:

www.blogdarobotica.com/tinkercad-CDRbot

INSTALE O PICTOBLOX

O Pictoblox é um programa de computador gratuito que permite a programação de jogos, animações, histórias e músicas.

A parte gráfica do nosso jogo será desenvolvida utilizando programação em blocos no Pictoblox. Portanto, será necessário que você instale este software em seu computador.

Em nosso blog, você encontra um guia de instalação do Pictoblox. Acesse:

www.blogdarobotica.com/guia-de-instalacao-do-pictoblox

PROGRAMAÇÃO NO PICTOBLOX

Dividimos a explicação da programação no Pictoblox em etapas com o intuito de facilitar o entendimento.

Etapas 1: Download dos atores e palco do projeto

Como o cenário e os atores (CDRbot, Obstáculos, Tijolo Surpresa, Estrela e outros) que vamos usar em nosso projeto não se encontram

disponíveis na biblioteca do Pictoblox, precisamos fazer o download. Baixe o arquivo do seguinte link:

www.blogdarobotica.com/pictoblox-CDRbot

Este é um arquivo com extensão .sb 3 , compatível com o Pictoblox e o Scratch 3. Após o download, abra o arquivo com o Pictoblox.

Ao abrir o arquivo no Pictoblox, você verá uma tela semelhante à Figura 2 1 e perceberá que o cenário e os atores já se encontram configurados no projeto.

No Pictoblox, o palco é a área onde o ator executa as ações de acordo com seu código. Você pode alterar o palco usando imagens, chamamos essas imagens de cenários.

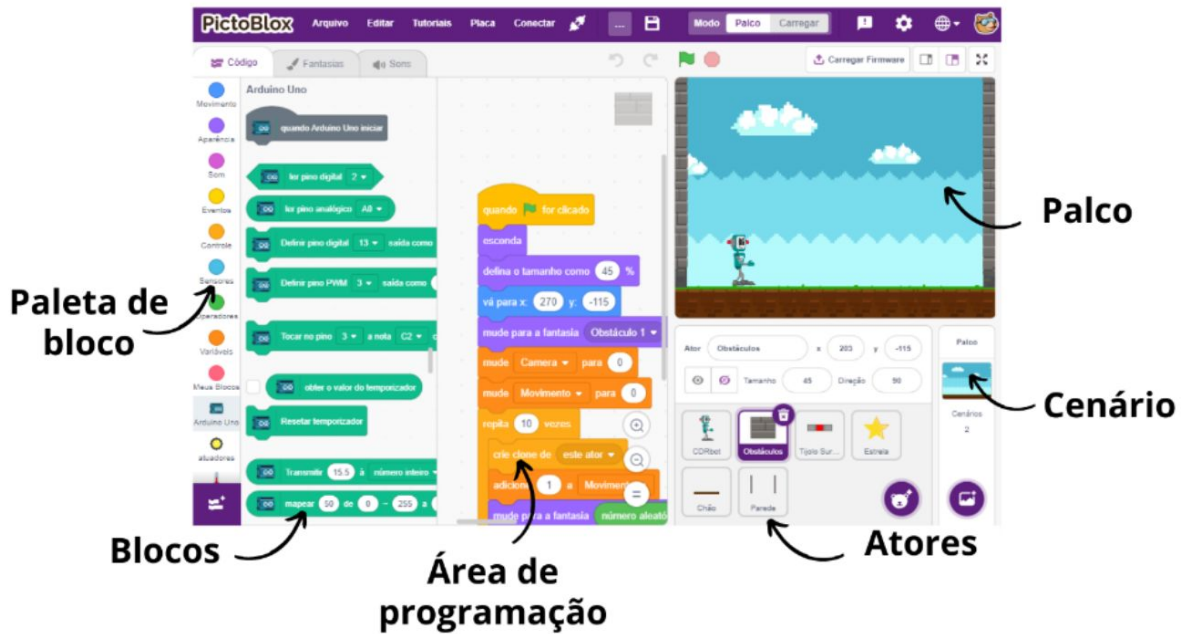
Um ator pode ser um personagem ou um objeto capaz compreender e realizar as ações escritas no código.

A área de programação é onde agrupamos um conjunto de blocos, interligados entre si, para realizar uma ação.

Os blocos são peças de encaixar usadas para criar os códigos no Pictoblox. Os blocos se conectam como um quebra-cabeça.

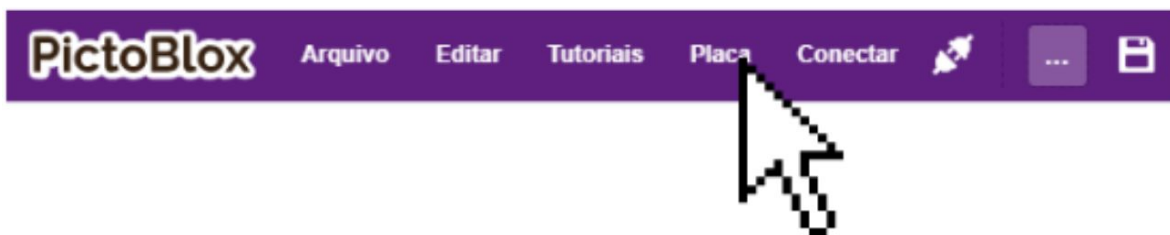
Os blocos estão localizados na paleta de blocos . Existem várias paletas diferentes, as principais são: movimento, aparência, som, eventos, controle, sensores, operadores, variáveis e meus blocos.

Figura 21 - Elementos do Pictoblox.



Etapa 2: Preparar as configurações do Pictoblox para receber e enviar informações para o Arduino

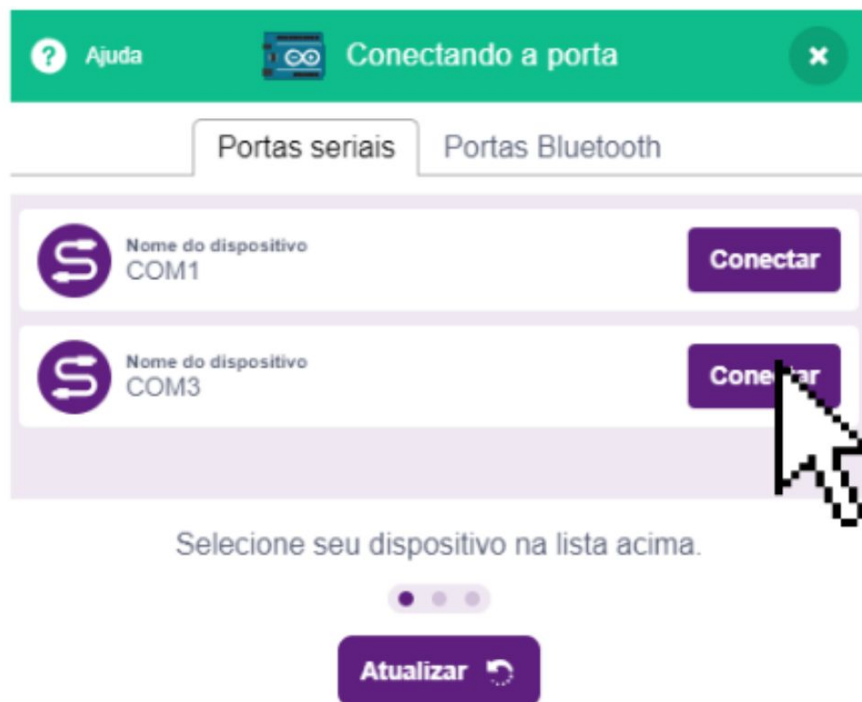
A primeira tarefa que devemos fazer antes de iniciar a programação usando o Pictoblox é configurar a versão do Arduino que será utilizada. Para isso, clique em Placa.



Em seguida, a tela Selecionar Placa será exibida. Vamos clicar em Arduino Uno para selecioná-lo.



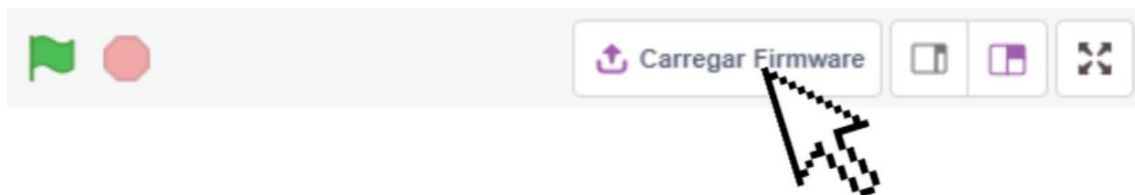
Logo após, vamos selecionar a Porta Serial em que o Arduino Uno foi reconhecido pelo computador. Em nosso exemplo, o Arduino foi reconhecido pela COM 3 . Clique em Conectar .



Caso seu Arduino não esteja sendo localizado, verifique a ligação com o cabo USB e clique em Atualizar .

Etapa 3: Carregar firmware para o Arduino

Clique na opção Carregar Firmware , localizado no canto superior direito do Pictoblox.



Etapa 4: Codificação do jogo

Após as configurações, é hora de programar! Siga as instruções para criar o jogo Super CDRbot.

A) Obstáculos

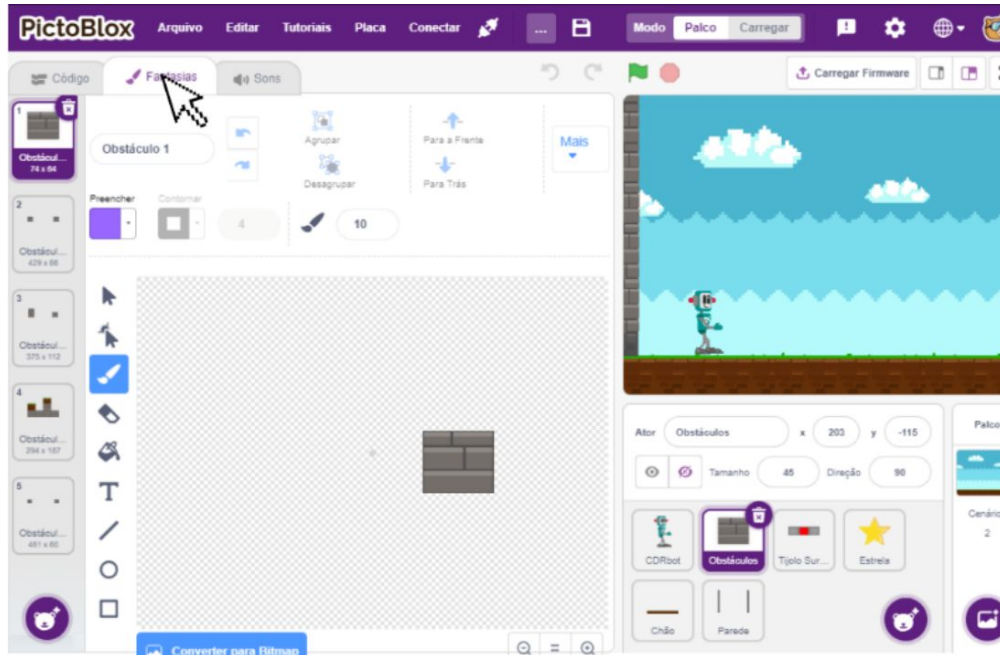


Vamos iniciar programando o ator Obstáculos. Para isso, clique na miniatura do ator para acessar seu editor de código.

O ator Obstáculos é composto por 5 fantasias (Figura 2.2). Você pode visualizá-las clicando em *Fantasias* . Cada fantasia equivale a um

obstáculo diferente que o CDRbot terá que saltar.

Figura 22 - Fantasia do ator Obstáculos.



Iniciamos o código do ator Obstáculos escondendo-o e definindo seu tamanho, posição e fantasia inicial.

Em seguida, vamos criar duas variáveis: *Camera* e *Movimento* . Para isso, clique na opção *Variáveis* localizada na paleta de blocos. Logo após, clique em *Criar variáveis* . Inicialize as variáveis com o valor zerado, ou seja, mude *Camera* e *Movimento* para zero.

Inclua o bloco “ *Repita*” . Em seu espaço em branco, inclua o número 10. Este trecho do código será responsável por criar 10 clones do ator Obstáculos. Para que os clones possuam fantasias diferentes de forma aleatória, usaremos o bloco “ *número aleatório entre... e ...*” .

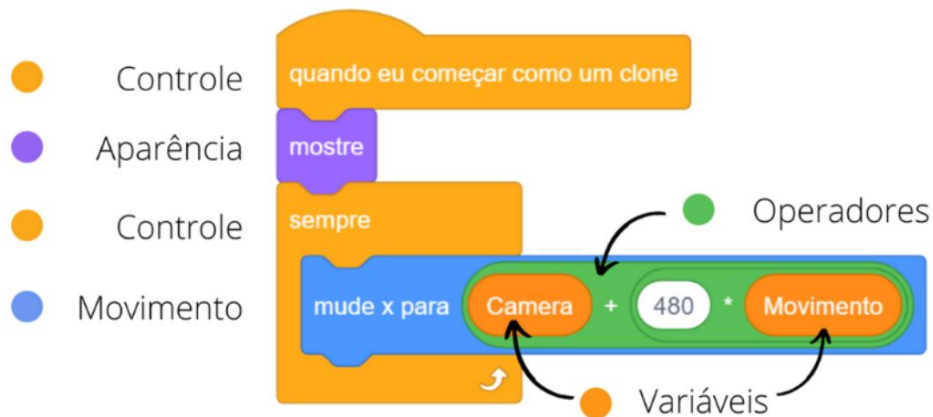
O primeiro código para o ator Obstáculos ficará da seguinte maneira:



Agora, vamos criar outros dois códigos que serão responsáveis pela movimentação dos Obstáculos no cenário.

Começamos o segundo código incluindo o bloco “*Quando eu começar como clone*”, localizado na paleta Controle. Em seguida, vamos exibir o clone criado e fazer com que o clone acompanhe a movimentação da câmera. Para isso, precisamos alterar sua posição x para o resultado da seguinte equação: $\text{Posição x} = \text{Camera} + 480 \times \text{Movimento}$. Em que, 480 é a largura total do palco.

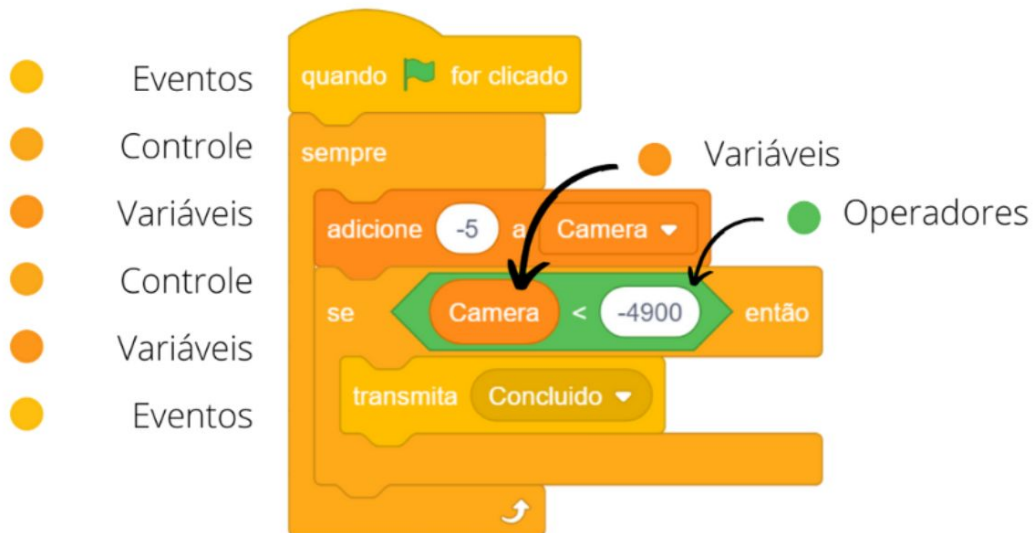
O segundo código do ator Obstáculos ficará da seguinte maneira:



No terceiro código, vamos decrementar a variável *Camera* para que os Obstáculos se movam da direita para a esquerda. Vamos incluir também uma estrutura condicional *Se...Então* para verificar o fim do jogo.

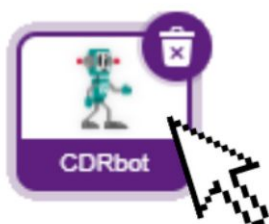
Se o valor da variável *Camera* for menor que -4900, a mensagem *Concluido* será transmitida.

O terceiro código do ator Obstáculos ficará da seguinte forma:





Para testar se os códigos do ator Obstáculos estão funcionando corretamente, clique na bandeira verde. Os Obstáculos se movem da direita para a esquerda no palco? Suas fantasias são alteradas de modo aleatório? Se sim, está tudo correto. Caso contrário, revise a programação.



B) CDRbot

Clique na miniatura do nosso robozinho (CDRbot) para acessar seu editor de código.

Iniciaremos o código do CDRbot criando uma animação que simule que ele está caminhando pelo cenário do jogo. Uma animação é feita de muitas imagens estáticas, que são reproduzidas uma após a outra. Quando isso é feito muito rápido, simula que os personagens estão realmente se movendo

O Super CDRbot será um jogo de rolagem lateral, também conhecido como side-scroller, no qual o cenário e o personagem são vistos na lateral. Por isso, nosso personagem ficará de lado enquanto caminha.

Quando o jogo for finalizado (Camera < -4900), o CDRbot ficará de frente, seus olhos ficarão piscando e seu braço movimentará. O código para criar esses efeitos de animações ficará da seguinte forma:

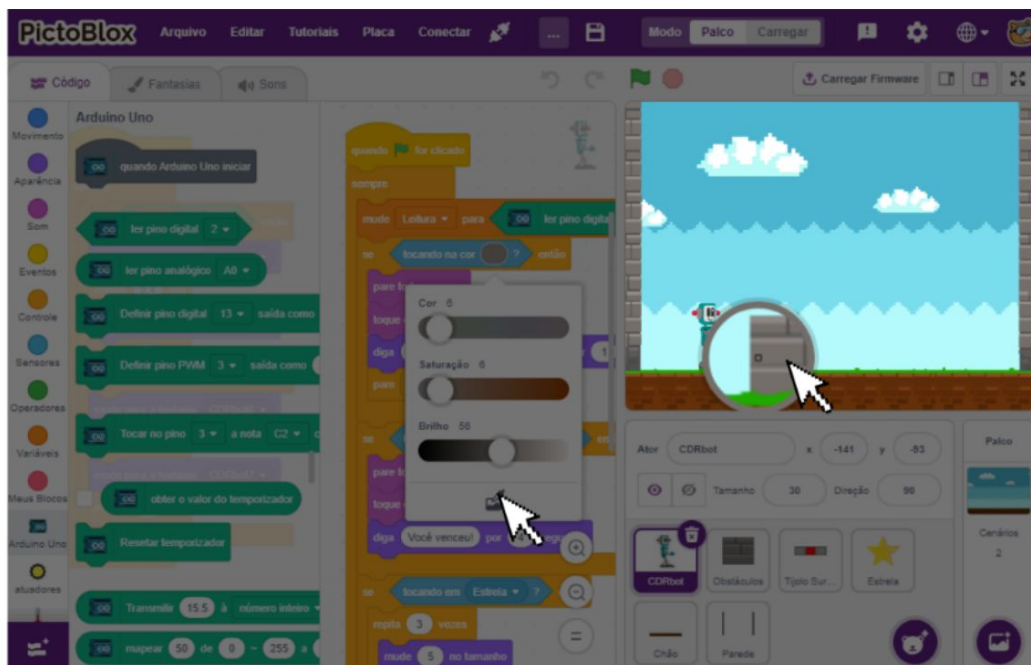


O segundo código do CDRbot será responsável por ler o pino digital 8 do Arduino e por verificar se o robô tocou os Obstáculos, o Tijolo Surpresa ou a Estrela. Estas quatro tarefas deverão ser executadas durante todo o jogo. Por isso, iniciamos este código incluindo o bloco “S *empre*”.

Em seguida, vamos criar a variável Leitura, que ficará responsável por armazenar o valor lido da porta digital 8 do Arduino, em que o botão está conectado.

Com auxílio da estrutura condicional *Se...Então* , vamos verificar se o CDRbot tocou os Obstáculos. Como existem Obstáculos com plataformas com gramado que o CDRbot pode tocar, vamos analisar se ele tocou na cor cinza. Use o Seletor de cor (Figura 23) para copiar a cor.

Figura 23 - Seletor de cor no Pictoblox.



Caso a condição seja verdadeira (CDRbot tocou na cor cinza), o som *Lose* deverá ser tocado e a mensagem “Game Over! Tente novamente.” será dita pelo CDRbot por 1 segundo. Em seguida, o bloco *Pare todos* encerra todas as ações do jogo.

Também com o auxílio da condicional *Se...Então*, vamos verificar se o CDRbot tocou o Tijolo Surpresa. Quando isso ocorrer, o som *Win* será tocado e a mensagem “Você venceu” será dita pelo CDRbot por 4 segundos.

A última verificação que vamos fazer no segundo código do CDRbot é se ele tocou a Estrela. Caso a condição seja verdadeira, o tamanho e a cor do CDRbot serão alterados 3 vezes. Após isso, a cor deve ser definida como zero novamente e todas as ações do jogo devem parar.

Assim, o segundo código do Ator CDRbot ficará da seguinte maneira:

Eventos

quando  for clicado

Controle

sempre

Variáveis

mude Leitura para  ler pino digital 8

Arduino Uno

Controle

se tocando na cor  ? então

Sons

pare todos os sons

Sensores

Sons

toque o som Lose

Aparência

diga Game Over! Tente novamente. por 1 segundos

Controle

pare todos

Controle

se tocando em Tijolo Surpresa ? então

Sons

pare todos os sons

Sensores

Sons

toque o som Win

Aparência

diga Você venceu! por 4 segundos

Controle

se tocando em Estrela ? então

Controle

repita 3 vezes

Sensores

Aparência

mude 5 no tamanho

Aparência

mude 25 ao efeito cor

Controle

espere 0.5 seg

Movimentos

mude y para -75

Aparência

defina o efeito cor como 0

Controle

pare todos

No terceiro código do CDRbot, vamos definir seu tamanho e posição inicial. Em seguida, vamos criar as variáveis *Pular* e *Pulou* .

A variável *Pular* será usada para determinar o incremento e decremento no eixo y, que deve ocorrer para que o CDRbot pule e volte para o chão.

Por sua vez, a variável *Pulou* será usada para verificar o estado em que o CDRbot se encontra (pulando ou não pulando).

Para evitar que CDRbot pule várias vezes seguidas ou que ele fique flutuando, vamos criar o bloco *Verificar pulo* . Para criar este bloco, clique na opção Meus Blocos na paleta de blocos. Em seguida, clique em Criar um Bloco.

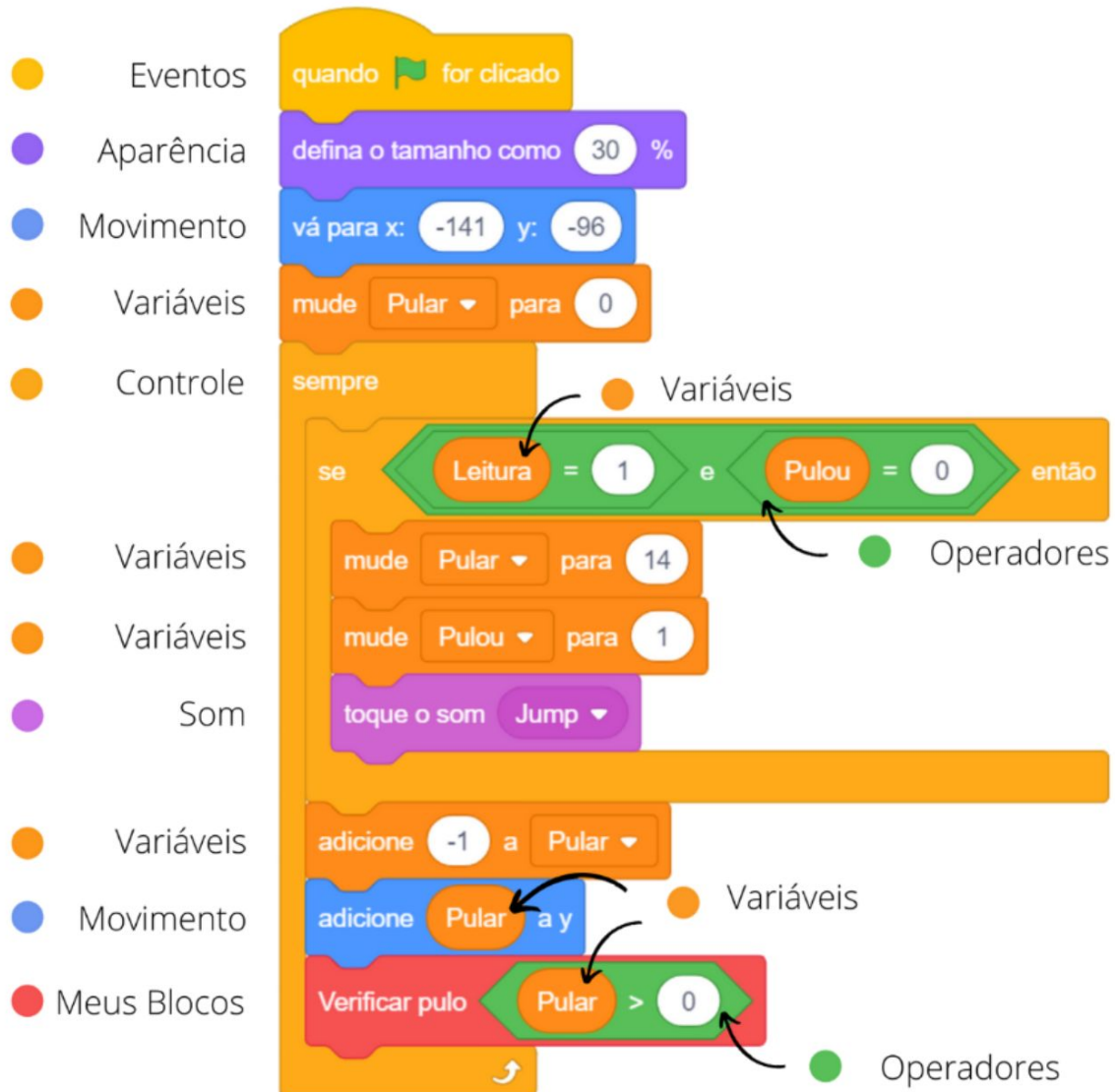
Em seguida, inclua *Verificar pulo* como nome do bloco e adicione uma entrada booleana com nome *alto* . Marque a opção *Executar sem atualização da tela* e clique em Ok, conform e Figura 2 4 .

Figura 24 - Criar um bloco no Pictoblox.



O bloco *Verificar pulo* será chamado sempre que a variável Pulou for maior que zero.

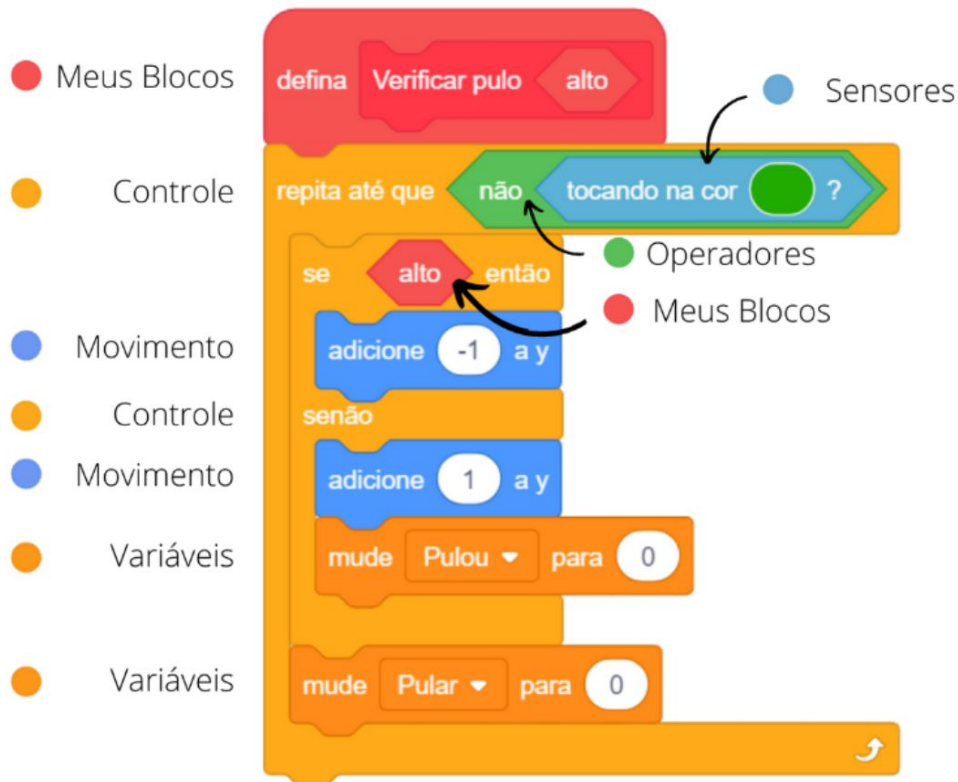
O terceiro código do CDRbot ficará da seguinte forma:



Agora precisamos escrever o código do bloco *Verificar Pulo*. Vamos iniciar este código incluindo o bloco de repetição controlado por lógica “*Repita até que*”. O código será repetido até que o CDRbot não esteja tocando no gramado (verde), ou seja, sempre que ele estiver no alto os blocos dentro do laço serão repetidos.

Em seguida, vamos verificar o valor de alto. Se o valor de alto for verdadeiro, significa que o CDRbot está subindo. Se o valor de alto for falso, o CDRbot está descendo do pulo e a variável *Pulou* deverá ser alterada para 0.

Ao final, o código do bloco *Verificar Pulo* ficará da seguinte forma:



Para testar se os códigos do CDRbot estão funcionando corretamente, clique na bandeira verde. O CDRbot pula quando o botão é pressionado? A animação do CDRbot simula que ele está andando?

C) Tijolo surpresa



Clique na miniatura do ator Tijolo Surpresa para acessar seu editor de código.

Para o tijolo surpresa, construiremos dois códigos. O primeiro define as suas configurações iniciais.



O segundo código será executado apenas quando o CDRbot tiver saltado os 10 blocos e a mensagem *Concluido* ter sido transmitida.

Quando Tijolo Surpresa receber a mensagem *Concluido* , o ator deverá ser mostrado no palco com sua fantasia *Tijolo 1* .

Após isso, vamos verificar se o CDRbot tocou o ator Tijolo Surpresa. Se sim, a fantasia de Tijolo Surpresa será alterada e a mensagem Estrela deverá ser transmitida.

O segundo código para do ator Tijolo Surpresa pode ser observado a seguir:



D) Estrela



Clique na miniatura ator Estrela para acessar seu editor de código.

Quando o jogo for iniciado, a Estrela deverá estar escondida. Portanto, em seu primeiro código vamos apenas realizar as suas configurações iniciais, definindo sua posição no palco e escondendo-a.

Em seguida, vamos criar a variável *posição*, que deve ser igualada a -6. Essa variável será utilizada para movimentação da Estrela em seu segundo código.

Também em seu primeiro código, vamos definir o pino digital 9 como baixo. Assim, o LED flash rápido ficará desligado ao início do jogo.

Seguindo estas ideias, o primeiro código da Estrela ficará do seguinte modo:



O segundo código da Estrela será executado somente quando o CDRbot tocar no Tijolo Surpresa.

Neste segundo código, escreveremos as instruções necessárias para que a Estrela apareça piscando e se mova no palco, bem como para o acionamento do LED flash rápido. Essas ações indicam que o jogador venceu o jogo e serão executadas apenas quando o ator Estrela receber a mensagem *Estrela*.

Para mover a Estrela no palco, incrementamos e decrementamos a variável *posição*.

O segundo código para a Estrela pode ser observado a seguir:



E) Chão



Clique na miniatura do ator Chão para acessar seu editor de código.

O código do ator Chão será apenas para configuração de sua posição no jogo e pode ser observado a seguir:

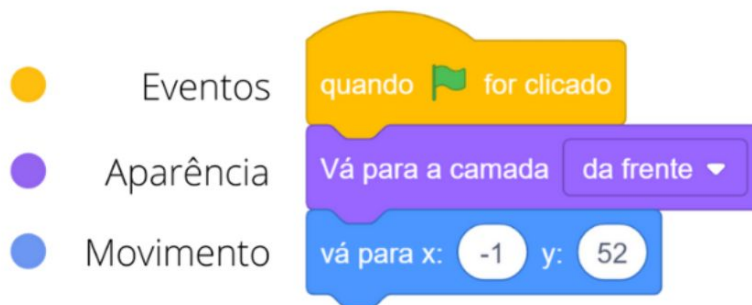


F) Parede



Clique na miniatura do ator Parede para acessar seu editor de código.

O código do ator Parede também definirá apenas as suas configurações no jogo e pode ser observado a seguir:



Etapa 4: Execução do projeto

Clique na bandeira verde para executar o código e testar o jogo completo.

- Os obstáculos estão se movimentando?
- Os obstáculos surgem com fantasias aleatórias?
- Quando o botão é pressionado, o CDRbot pula?
- Quando o CDRbot toca o obstáculo, a mensagem “Game Over! Tente novamente” é exibida e o som Lose é reproduzido?
- Quando o CDRbot salta 10 obstáculos, o tijolo surpresa aparece no palco?
- O que acontece quando o CDRbot toca no tijolo surpresa? A estrela aparece piscando no palco?
- O LED flash rápido é ligado quando a estrela surge no palco?

Etapa 5: Salve o Projeto

Assim que finalizar o projeto, não se esqueça de salvá-lo. Assim, você poderá acessá-lo novamente quando desejar.

DÚVIDAS?

Ficou com alguma dúvida? Para melhor visualização e simulação, você pode baixar o Jogo Super CDRbot em nosso site. Acesse:

www.blogdarobotica.com/pictoblox-SuperCDRbot

OceanofPDF.com

5.

KITS CASA DA ROBÓTICA

KIT INICIANTE



Kit Construyendo o Jogo da Memória e + 20 projetos:
Ideal para você aprender sobre o projeto Arduino e obter conhecimentos em eletrônica e programação.

KIT AUTOMAÇÃO

KIT Automatize seu Quarto:
Ideal para você aprender sobre o projeto Arduino e obter conhecimentos em Smart Home/Casas Inteligentes e Internet das Coisas - IoT.



WWW.CASADAROBOTICA.COM



CONSIDERAÇÕES FINAIS

Chegamos ao final do nosso e-book Criando Jogos com Arduino – Passo a Passo .

Caso tenha alguma dúvida entre em contato conosco, afinal “a dúvida é o princípio da sabedoria” (Aristóteles).

Caso você tenha alguma sugestão, por favor, entre em contato conosco. Sua opinião é muito importante para nós.

contato@casadarobotica.com

Acompanhe as novidades em nossas redes sociais:

Facebook: @casadaroboticaoficial

Instagram: @casadarobotica

Grupos para compartilhamento de informações, dúvidas e novidades:

Telegram: <http://www.blogdarobotica.com/GrupoTelegram>

Whatsapp: <http://www.blogdarobotica.com/GrupoWhatsapp>

Até a próxima,

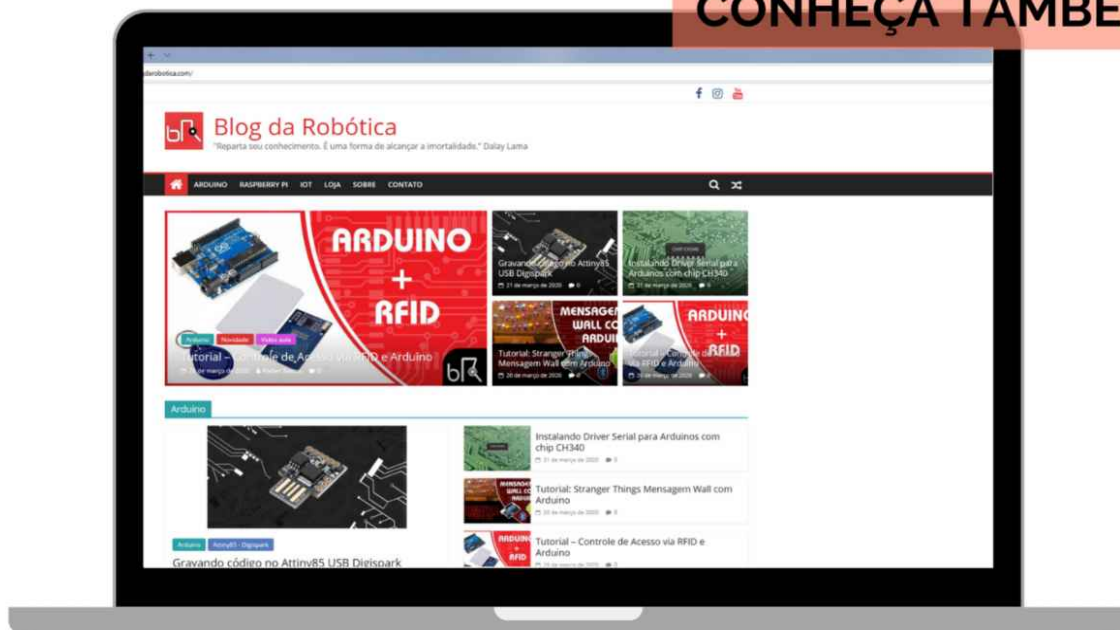
Equipe Casa da Robótica

OceanofPDF.com

blog da ROBOTICA

#ELETRÔNICA #MAKER #Tecnologia #PROGRAMAÇÃO
#CIÊNCIAS #ROBÓTICA #INOVAÇÃO

CONHEÇA TAMBÉM



blogda**robotica**.com



REFERÊNCIAS

ARDUINO. Documentação de Referência da Linguagem Arduino .

Disponível em: < <https://www.arduino.cc/reference/pt/> >. Acesso em: 29 jul. 2020.

ARDUINO. Downloads Arduino IDE . Disponível em: <

<https://www.arduino.cc/en/software> >. Acesso em: 29 jul. 2020.

MONK, Simon. Programação com Arduino : Começando com Sketches. 2. ed. Bookman, 2017. 180 p.

PASSA OU REPASSA . In: WIKIPÉDIA : enciclopédia livre. Wikimedia, 2021. Disponível em: < https://pt.wikipedia.org/wiki/Passa_ou_Repassa >. Acesso em: 29 jul. 2020.

REDAÇÃO MUNDO ESTRANHO. Qual é a origem do jogo de dados ? . Super Interessante. Disponível em: < <https://super.abril.com.br/mundo-estranho/qual-e-a-origem-do-jogo-de-dados/> >. Acesso em: 29 jul. 2020.

SCHMIDT, J. O jogo mais antigo: A história dos dados . Blog Bitcasino. Disponível em: < <https://bitcasino.io/pt/blog/tipshackstricks/history-of-dice> >. Acesso em: 29 jul. 2020.

STEMPEDIA . Getting Started with PictoBlock . Disponível em: < <https://thestempedia.com/tutorials/getting-started-pictoblox> >. Acesso em: 29 jul. 2020.

OceanofPDF.com

SOBRE NÓS

Carol Correia Viana

Bacharel em Engenharia Elétrica com ênfase em Eletrônica, mestra em Engenharia Industrial e especialista em Docência com ênfase em Educação Inclusiva.

Atua no setor de Desenvolvimento de Produtos na Casa da Robótica, participando ativamente das etapas de planejamento, produção e revisão dos kits de Robótica Educacional. Criadora de conteúdo no Blog da Robótica. Fanática por livros, Star Wars e projetos *Maker* .

Kleber Rocha Bastos

Bacharel em Engenharia de Computação. Um dos criadores do Blog da Robótica. Faz parte da equipe Casa da Robótica desde 2017. Apaixonado por tecnologias e pelas “loucuras” que é possível criar usando microcontroladores e impressoras 3D.

Ex-BBB, mas isso é uma longa história... que poderá ser contada posteriormente.

Luan Silva Santana

Bacharel em Engenharia de Computação e mestre em Mecatrônica. Sócio fundador da Casa da Robótica. Atua na gestão estratégica da empresa e no desenvolvimento de produtos.

Aficionado por robótica, programação e eletrônica.

Wellington Assunção Azevedo

Bacharel em Engenharia de Computação e mestre em Mecatrônica. Sócio fundador do Blog da Robótica e da Casa da Robótica, onde atua na direção de negócios, planejamento e desenvolvimento da empresa.

Apaixonado por criar robôs e participar de competições.

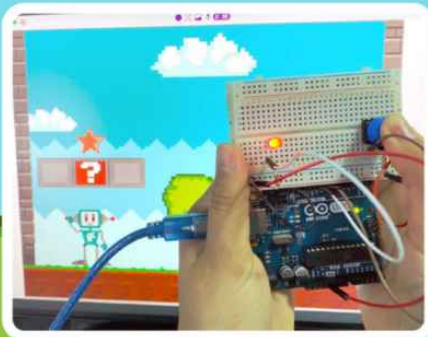
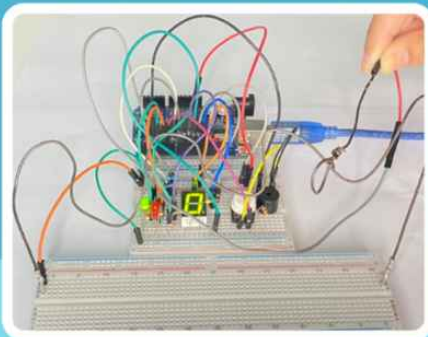
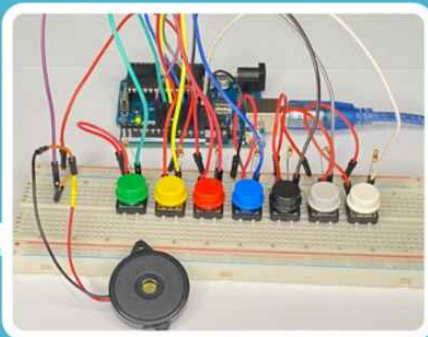
Carlos Dyorgenes Santana

Bacharel em Administração de Empresas e em Ciências Contábeis e especialista em Análise e Viabilidade de Projetos Financeiros. Atua como administrador, contador e Maker na Casa da Robótica.

Para Carlos é uma alegria imensurável compartilhar conhecimentos, sobretudo com pessoas entusiasmadas em aprender programação, eletrônica e robótica.

Anielle Barbosa Menezes

Graduanda em Engenharia Elétrica e aprendiz de sistemas microcontrolados. Atua como estagiária na Casa da Robótica, estudando e auxiliando no desenvolvimento de projetos diversos com variadas placas microcontroladoras, dispositivos e componentes eletrônicos.

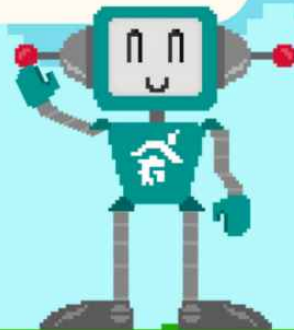


É com grande alegria que celebramos nosso sexto aniversário e, para comemorar este momento especial, convidamos você para se divertir com a gente na criação de jogos com Arduino.

Este e-book foi carinhosamente preparado para você, que assim como nós, é apaixonado por eletrônica, programação, games e, é claro, Arduino.

Neste e-book elaboramos sete jogos com Arduino para você recriar.

Venha aprender e se divertir com a gente!



REALIZAÇÃO:



casadarobotica.com



blogdarobotica.com

OceanofPDF.com