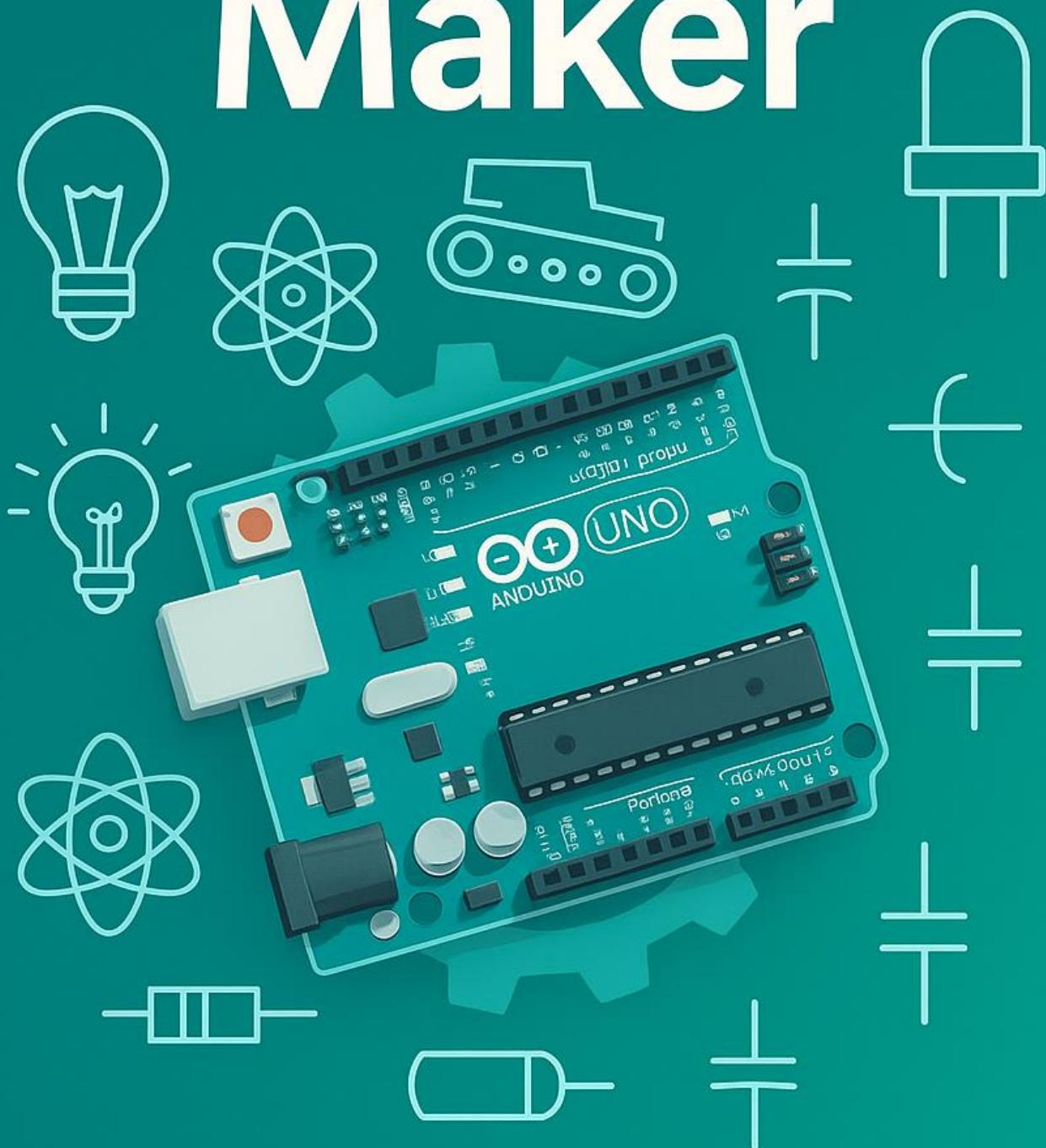


Robótica & Programação Maker



Prof. Sandro Mesquita

ROBÓTICA E PROGRAMAÇÃO MAKER: PROJETO JOGO DA MEMÓRIA

Uma abordagem de ensino orientado a projetos desde os conceitos básicos até a programação funcional para iniciantes, desenvolvedores, makers e professores.

Por

Sandro Costa Mesquita

Ficha Catalográfica

(Elaborada conforme a ABNT NBR 6023:2018)

Robótica e Programação Maker: Projeto Jogo da Memória

Uma abordagem de ensino orientado a projetos desde os conceitos básicos até a programação funcional para iniciantes, desenvolvedores, makers e professores.

Autor: Sandro Costa Mesquita

Local de Publicação: Fortaleza – CE

Ano de Publicação: 2025

Número de Páginas: [Total de Páginas]

Assuntos:

1. Programação em Python
2. Estruturas de Dados
3. Modularização e Funções
4. Programação Funcional
5. Ciência da Computação

Classificação Decimal de Dewey (CDD): 629.89

Classificação Decimal Universal (CDU): 004.932.2:373.3/.5

Créditos

Título: Robótica e Programação Maker: Projeto Jogo da Memória

Autor: Mesquita, Sandro Costa

Ano de Publicação: 2025

Edição: 1^a edição

Editora: [Nome da Editora ou "Publicação Independente"]

Design da Capa: [Nome do designer, se houver]

Revisão Técnica: [Nome do revisor, se houver]

Bibliotecário Responsável pela Ficha Catalográfica: [Nome do Bibliotecário e CRB]

Contato do Autor: contato@profsandromesquita.com

Formação: Graduado em Mecatrônica Industrial, Engenheiro de Software, Especialista em Automação Industrial, MBA em Engenharia de Petróleo, Mestre em Engenharia de Software, Doutorando em Bioinformática (Fiocruz Ceará).

Experiência: 15 anos ensinando robótica educacional. Coordenador e professor de Engenharia da Computação, coordenador de pós-graduação em Inteligência Artificial. Cofundador da Comunidade Arduino Ceará e presidente do ITIA (Instituto de Tecnologia e Inteligência Artificial)

.

Contato: Instagram [@profsandromesquita](https://www.instagram.com/profsandromesquita), Site profsandromesquita.com, WhatsApp (85) 98818-2453.

Sumário

ROBÓTICA E PROGRAMAÇÃO MAKER:	1
PROJETO JOGO DA MEMÓRIA	1
Ficha Catalográfica	2
Créditos	2
Introdução	6
UNIDADE 01: Sobre o eBook e sua Aplicação Pedagógica.....	9
1.1 Proposta do eBook	9
1.2. Importância da Robótica e Programação para o Futuro	13
Exercícios – Unidade 01	15
UNIDADE 02: Introdução ao Arduino	17
2.1. O que é o Arduino?	17
2.2. Conhecendo a placa Arduino Uno (pinos e funções).....	19
2.3. Instalando o Arduino IDE	23
2.4. Escrevendo e Executando o Primeiro Código: “Blink” (Piscar LED)	26
2.5. Conceitos Básicos de Programação Aplicados.....	28
2.6. Arduino e o Jogo da Memória: visão adiante	33
Exercícios – Unidade 02	35
UNIDADE 03: Eletrônica Aplicada ao Projeto	37
3.1. Noções Básicas de Eletricidade: corrente, tensão e resistência.....	37
3.2. Resistores.....	40
3.3. LEDs	41
3.4. Push-Buttons (Botões de pressão).....	43
3.5. Potenciômetro	45
3.6. Display LCD com módulo I2C	46
3.7. Buzzer	48
3.8. Bateria de 9V.....	49
3.9. Protoboard (Placa de ensaio).....	51
3.10. Lei de Ohm e Divisor de Tensão (revisão rápida com aplicação)	52
3.11. Dimensionamento Correto de Componentes	53
Exercícios – Unidade 03	55
UNIDADE 04: Montagem do Projeto	57
4.1 Preparando a Protoboard e Alimentação.....	58
4.2 Montando os LEDs com resistores	59
4.3 Montando os Botões	60

4.4 Conectando o Buzzer	63
4.5 Conectando o Display LCD (com I2C)	63
4.6 Conectando o Potenciômetro:.....	65
4.7 Conectando a Alimentação Externa (Bateria 9V).....	65
4.8 Análise da Imagem do Circuito Montado	66
4.9 Recomendações de Segurança e Organização	67
4.10 Dicas Práticas para Crianças Montarem	68
Exercícios – Unidade 04	70
UNIDADE 05: Programação do Jogo da Memória	72
5.1 Visão Geral do Código	72
5.2 Código Completo	74
5.4 Entendendo e Adaptando o Código	99
5.5 Carregando e Testando o Código	101
5.6 Expandindo e Melhorando	102
Exercícios – Unidade 05	104
UNIDADE 06: Funcionamento e Manual do Usuário	106
6.1 Manual de Uso do Jogo da Memória	106
6.3 Situações Comuns e Soluções	109
6.4 Sugestões de Melhoria ou Expansão do Projeto.....	111
6.5 Encerrando o Projeto.....	113
Exercícios – Unidade 06	114
CONCLUSÃO	116

Introdução

Olá! Seja bem-vindo ao **eBook de Robótica com Arduino**, um guia educacional prático e divertido para iniciantes. Sou o professor **Sandro Mesquita**, professor e entusiasta de tecnologia. Tenho 15 anos de experiência ensinando robótica para crianças e adolescentes, e elaborei este material para compartilhar um pouco desse conhecimento com você. O objetivo deste eBook é **ensinar robótica e programação de maneira simples e interativa**, usando a plataforma Arduino e um projeto lúdico: um jogo da memória eletrônico (estilo “Genius”).

Quem sou eu? Sou graduado em Mecatrônica Industrial e Engenharia de Software, Especialista em Automação Industrial, possuo MBA em Engenharia de Petróleo, Mestre em Engenharia de Software e doutorando em Bioinformática na Fiocruz Ceará, até a data da escrita desse eBook. Atuo como professor universitário e coordenador dos cursos de Engenharia de Software e Engenharia da Computação, também coordeno a pós graduação Inteligência Artificial. Desde 2008, venho trabalhando com robótica educacional, ajudando jovens a descobrir o prazer de aprender construindo seus próprios dispositivos tecnológicos. Acredito que a robótica é uma forma fantástica de aprender **resolução de problemas, criatividade e trabalho em equipe**, competências essenciais para o século XXI.

Competências desenvolvidas: Este eBook foi elaborado levando em conta as **Diretrizes Curriculares Nacionais e a Base Nacional Comum Curricular (BNCC)**, que destacam a importância de desenvolver diversas habilidades nos alunos. A robótica educacional é uma atividade **interdisciplinar e transversal**, capaz de englobar várias competências gerais da educação básica, como raciocínio lógico, pensamento científico, cultura digital, comunicação, cooperação e protagonismo. Por exemplo, a robótica **estimula o raciocínio lógico e a capacidade de resolver problemas** ao desafiar o aluno a montar circuitos e programar sequências.

Também promove a **colaboração e o engajamento** quando os estudantes trabalham juntos para construir e testar suas invenções. Outra vantagem é fortalecer a aprendizagem integrada de **ciências, tecnologia, engenharia, artes e matemática (princípios STEAM)**, conectando conceitos dessas áreas de forma prática e interativa.

Vale ressaltar que **programação e robótica** vêm ganhando espaço oficial no currículo. De acordo com a legislação educacional recente, o ensino de computação, programação e robótica é um diferencial no Ensino Médio. Isso reflete a visão de que formar crianças e jovens fluentes em tecnologia é tão importante quanto outros conhecimentos tradicionais. A própria BNCC, em sua 5ª competência geral, incentiva os alunos a “**compreender, utilizar e criar tecnologias digitais de forma crítica e ética (...) para resolver problemas e exercer protagonismo e autoria**”. Ou seja, espera-se que o estudante não seja apenas usuário, mas também **criador de tecnologia**, e projetos de robótica são uma ótima maneira de alcançar isso.

Interdisciplinaridade: A robótica combina conhecimentos de várias disciplinas escolares. Por exemplo:

- **Matemática:** ao programar, usamos lógica, sequências, contagem e eventualmente cálculos de tempo, ângulos ou até estatística básica (como percentuais de acertos no jogo).
- **Ciências/Física:** para montar circuitos entendemos conceitos de eletricidade, energia, tensão e corrente. Aplicamos a **Lei de Ohm ($U = R \cdot I$)** para calcular resistências necessárias nos circuitos. Também exploramos sensores e atuadores, que envolvem princípios físicos.
- **Engenharia:** a própria montagem do protótipo envolve pensamento de engenharia como planejar, construir, testar, identificar falhas e melhorar (ciclo de projeto).
- **Artes:** a criatividade é fundamental na robótica. Podemos escolher cores de LEDs, personalizar a aparência do projeto, criar caixas/enfeites para o jogo da memória, etc. Isso integra aspectos artísticos e de design.
- **Língua Portuguesa:** ao aprender robótica, também praticamos leitura (de tutoriais, diagramas) e escrita/descrição de projetos. Os estudantes podem documentar o processo, escrever um diário de bordo ou apresentar seu projeto, treinando comunicação.
- **Inglês:** grande parte do material de robótica (exemplos de código, termos técnicos, manuais) está em inglês. Durante o aprendizado, acabam aparecendo palavras em inglês (como *display*, *sensor*, *software*, etc.), o que incentiva o aluno a aprender novos termos nesse idioma.

Metodologia adotada: A abordagem deste eBook é **mão na massa**, ou seja, você vai aprender fazendo! Aqui, cada unidade combina teoria e prática, sempre relacionada ao projeto do jogo da memória. Essa metodologia está alinhada com a ideia de **construcionismo** do educador Seymour Papert, que defendia que os alunos aprendem melhor quando estão ativamente engajados em construir algo significativo para eles. Não se preocupe: vamos guiá-lo passo a passo. Primeiro, apresentamos conceitos básicos (por exemplo, o que é Arduino, o que são LEDs, etc.) e em seguida aplicamos construindo partes do projeto. Você será estimulado a pensar, experimentar e até errar, porque é normal errar ao montar e programar, e é assim que aprendemos! Conforme Papert destacou, quando o aluno tem oportunidade de construir algo de seu interesse, “a elaboração do conhecimento acontece de fato”.

A linguagem utilizada é **acessível para faixas etárias de 8 a 15 anos**, evitando termos excessivamente técnicos e explicando todos os conceitos passo a passo. Para os leitores mais novos, algumas partes podem exigir o apoio de um adulto ou professor (especialmente nas etapas de montagem elétrica, por segurança). Já os leitores mais experientes podem avançar mais rápido em certos trechos. Sinta-se à vontade para ir no seu ritmo.

Importância da robótica e programação na formação: Aprender robótica desde cedo traz inúmeros benefícios. Desenvolve o *pensamento lógico* e a criatividade, pois ao programar um robô (ou jogo), precisamos planejar sequências lógicas de

ações e, ao mesmo tempo, encontrar soluções criativas para os desafios que surgem. Além disso, projetos assim **aumentam a curiosidade sobre como a tecnologia funciona** – em vez de serem apenas consumidores passivos de jogos e aparelhos, os alunos tornam-se inventores, questionando e explorando os mecanismos por trás das coisas. A robótica também pode melhorar o desempenho em outras matérias, pois muitos conceitos abstratos tornam-se concretos: por exemplo, conteúdos de matemática e física ficam mais fáceis de entender quando aplicados em um robô de verdade.

Por fim, aprender programação e robótica prepara as crianças e adolescentes para um futuro cada vez mais tecnológico. Mesmo que nem todos sigam carreira na área de TI ou engenharia, as habilidades desenvolvidas – raciocínio estruturado, resolução de problemas, adaptabilidade, trabalho em equipe – serão úteis em qualquer profissão. E quem sabe? Talvez este eBook desperte em você uma paixão por tecnologia que pode orientar seus estudos e carreira no futuro.

Nas próximas unidades, vamos então mergulhar no universo do Arduino e da eletrônica básica, montar o circuito do jogo da memória passo a passo, aprender a programar o jogo e, ao final, você terá em mãos (e funcionando!) um mini jogo eletrônico feito por você mesmo. Vamos começar essa aventura tecnológica?

UNIDADE 01: Sobre o eBook e sua Aplicação Pedagógica

Nesta primeira unidade, vamos entender **a proposta pedagógica** deste eBook e como aproveitá-lo ao máximo. Você descobrirá quais **competências** serão desenvolvidas ao longo do projeto, quais **disciplinas escolares** se relacionam com os conteúdos abordados, qual **metodologia de ensino** utilizamos e por que aprender robótica e programação desde cedo é importante para seu futuro.

1.1 Proposta do eBook

A proposta deste eBook é guiá-lo na construção de um **projeto completo de robótica educacional**: o jogo da memória eletrônico com Arduino. Mas não se trata apenas de montar um circuito e escrever código; nosso foco é na **aprendizagem durante o processo**. Ao seguir as instruções, você vai:

- **Aprender conceitos de eletrônica básica** (o que são e para que servem resistores, LEDs, etc.).
- **Entender noções de programação** (como o Arduino executa instruções, uso de variáveis, estruturas como condições e loops).
- **Desenvolver habilidades práticas** como manusear componentes, montar um protótipo em protoboard e resolver problemas de ligação.
- **Estimular o pensamento lógico e crítico**, pois a cada etapa você será instigado a refletir sobre o que está acontecendo – por exemplo, por que precisamos de um resistor em série com o LED? Como o Arduino "sabe" que apertamos um botão?
- **Criar algo útil e divertido** – no final, você terá construído um jogo que realmente funciona, podendo desafiar amigos e familiares, o que traz senso de realização e motivação.

Este eBook pode ser utilizado de forma **autônoma pelo aluno** ou integrado em atividades escolares sob orientação de um professor. Professores podem usá-lo como material de apoio em oficinas de robótica, clubes de programação ou até em aulas regulares de ciências e matemática, dada sua natureza interdisciplinar.

Competências Desenvolvidas

Ao longo do projeto, diversas **competências e habilidades** serão trabalhadas. Vamos destacar as principais:

- **Raciocínio Lógico e Resolução de Problemas:** A programação do Arduino e a montagem do circuito exigem planejar passos lógicos e identificar soluções quando algo não funciona. Você aprenderá a “pensar como um computador”, de forma estruturada, para criar a sequência correta de comandos. Segundo especialistas, montar robôs e programar códigos incentiva o aluno a pensar de forma estruturada, favorecendo o raciocínio lógico e a habilidade de resolver problemas.
- **Criatividade e Inovação:** Embora seguimos um projeto específico, há espaço para personalização e ideias próprias (por exemplo, escolher as cores dos LEDs, inventar novas regras para o jogo, decorar seu dispositivo).

A robótica educacional convida a inovar e experimentar — se algo pode ser melhorado, você é encorajado a tentar!

- **Coordenação Motora e Atenção aos Detalhes:** Montar componentes pequenos em uma protoboard, manusear fios, conectar tudo nos pinos corretos — isso desenvolve sua coordenação motora fina e também sua capacidade de concentração e atenção, pois um único pino errado pode impedir o projeto de funcionar.
- **Trabalho em Equipe e Comunicação:** Se você estiver fazendo este projeto em grupo (por exemplo, com colegas na escola), aprenderá a dividir tarefas, trocar ideias e comunicar descobertas. Montar um robô em equipe é uma ótima oportunidade de colaboração: ao compartilhar informações, testando juntos e dialogando para resolver dificuldades, os alunos desenvolvem a criatividade e a **colaboração**.
- **Curiosidade Científica:** Você vai começar a olhar aparelhos do dia a dia e se perguntar “como isso funciona?”. Ao ver um LED piscando ou um som tocando no buzzer, você compreenderá conceitos que estão presentes em celulares, computadores, semáforos e tantos outros dispositivos, despertando sua curiosidade pelo funcionamento das tecnologias ao redor.
- **Autonomia e Persistência:** Projetos de robótica ensinam a aprender com tentativa e erro. Se algo não der certo de primeira, tudo bem — você analisa o problema e tenta novamente, aprendendo muito no processo. Isso desenvolve a persistência e a autonomia para buscar soluções. Ao finalizar o projeto, você ganhará confiança para enfrentar novos desafios, pois sabe que é capaz de aprender e superar obstáculos.

Disciplinas Relacionadas

Como vimos na Introdução, este projeto dialoga com várias disciplinas escolares:

- **Matemática:** aparece nos cálculos simples (ex.: determinar o valor do resistor usando a fórmula da Lei de Ohm), na lógica dos algoritmos e até na noção de sequência do jogo (progressão do nível de dificuldade). Programar também envolve compreender conceitos matemáticos como variáveis e eventualmente funções.
- **Ciências e Física:** a base do circuito eletrônico é física pura — corrente elétrica, tensão (voltagem), resistência, energia (pilha/bateria). Vamos abordar as grandezas elétricas, unidades de medida (Volt, Ohm, Ampère) e conceitos como circuito **série e paralelo**, divisor de tensão, etc. Tudo isso está dentro do conteúdo de Ciências/Física do ensino fundamental.
- **Tecnologia e Informática:** naturalmente, mexer com Arduino é entrar no mundo da computação física. O Arduino é um microcontrolador, parente dos computadores, então ao programá-lo estamos praticando princípios de Ciência da Computação (como lógica booleana nos condicionais, laços de repetição, depuração de código básico, etc.). Hoje em dia fala-se muito em **cultura digital e computacional**, e projetos assim inserem essas competências de forma lúdica no cotidiano escolar.
- **Engenharia/Robótica:** a disciplina de robótica (quando houver na escola) seria diretamente relacionada. Mas mesmo sem uma matéria específica, consideraremos parte de *engenharia maker*: projeto, construção e teste. A

BNCC enfatiza a **aprendizagem “mão na massa” em áreas STEAM** (Ciência, Tecnologia, Engenharia, Artes e Matemática), e esse projeto exemplifica bem essa integração de áreas.

Figura 1 - Educação Steam



- **Artes:** sim, também podemos relacionar! O produto final – um jogo da memória eletrônico – pode ser embelezado pelos alunos. Sugere-se que, após a parte eletrônica pronta, os alunos confeccionem uma caixa ou painel para o jogo, desenhem símbolos ou personagens para cada botão/LED, enfim, usem criatividade artística para deixar o projeto com a cara deles. Dessa forma, trabalhamos expressão artística e design, mostrando que tecnologia e arte podem andar juntas.

Metodologia Adotada

A metodologia adotada neste eBook é **inspirada em projetos**. Isso significa que todo o conteúdo teórico é imediatamente aplicado em algo concreto. Em vez de aprender teoria de eletrônica de forma abstrata, você aprenderá porque precisa usar um resistor ao realmente **colocar a mão no resistor para proteger um LED**. Em vez de estudar programação lendo slides, você escreverá código para fazer algo divertido acontecer no mundo real (um LED piscar, um som tocar, etc.). Essa abordagem segue a máxima: “*Aprender Fazendo*”.

Figura 2 - Cultura Maker e Educação Steam



Cada unidade combina explicações didáticas com **atividades práticas**:

- **Passo a Passo:** Todo procedimento de montagem ou programação é descrito passo a passo, numerado, para facilitar o acompanhamento. Você é orientado o tempo inteiro, então não tenha medo se for iniciante.
- **Explicações Simples:** Após cada ação prática, há uma explicação do *porquê* daquilo. Exemplo: você conectou um fio em certo pino – em seguida explicamos o que aquele pino faz e por que a ligação é assim. Isso ajuda a conectar a prática com a teoria.
- **Ilustrações e Diagramas:** Sempre que possível, o texto inclui imagens ou diagramas para auxiliar a visualização. Por exemplo, teremos o diagrama do Arduino Uno com seus pinos, e a imagem do circuito completo montado. Assim, você pode conferir se montou corretamente comparando com a figura.
- **Perguntas de Fixação:** Ao final de cada unidade, há uma lista de **10 questões** relacionadas ao conteúdo aprendido. Essas perguntas servem para revisar os conceitos e estimular você a pensar sobre o que foi feito. Você pode tentar responder sozinho e depois discutir com um professor ou colega. As questões variam entre identificar funções de componentes, aplicar conceitos (cálculos simples), e refletir sobre aplicações.

Um ponto importante da metodologia é incentivar a **descoberta guiada**. Por exemplo, podemos propor um pequeno desafio: “O que você acha que acontece se inverter tal componente?” ou “Tente modificar tal linha do código e observe o resultado.” Essas explorações controladas ajudam a construir autonomia e um entendimento mais profundo, ao invés de apenas seguir receitas prontas. Sinta-se livre para experimentar (sempre tomando os cuidados de segurança indicados) – a robótica é um campo de experimentação!

Por fim, a metodologia também se baseia na **contextualização**. Vamos sempre relacionar o que estamos montando com situações reais ou conhecimentos prévios. Se falarmos de um buzzer (dispositivo sonoro), vamos lembrar de campainhas, alarmes etc. Se usarmos um sensor ou botão, falaremos de exemplos

cotidianos (como um interruptor de luz). Isso torna o aprendizado mais significativo, pois você percebe onde aquele conhecimento se encaixa no mundo.

1.2. Importância da Robótica e Programação para o Futuro

Vivemos em uma época em que a tecnologia está presente em praticamente todos os aspectos de nossas vidas. Celulares, computadores, Internet, eletrodomésticos inteligentes, carros com piloto automático – a lista é enorme. Saber *programar* essas máquinas e entender seus princípios de funcionamento deixou de ser algo restrito a cientistas e engenheiros e passou a ser uma **habilidade básica** desejável para todos.

Figura 3 - Importância da Robótica e Programação na Educação



Aprender robótica e programação ainda criança/adolescente oferece vantagens como:

- **Pensamento Computacional:** Mesmo que você não siga carreira em programação, aprender a programar um Arduino desenvolve sua capacidade de decompor problemas complexos em partes menores, reconhecer padrões, criar algoritmos (sequências de passos) e pensar de forma sistemática. Esse tipo de pensamento é útil para solucionar problemas em qualquer área do conhecimento.
- **Oportunidades de Carreira:** O mercado de trabalho do futuro (e do presente!) busca profissionais familiarizados com tecnologia. Profissões tradicionais estão incorporando cada vez mais automação e análise de dados. Ao aprender programação e robótica, você já sai na frente adquirindo essa “fluência digital”. Sem contar nas carreiras diretamente ligadas, como ciência da computação, engenharia de software, mecatrônica, inteligência artificial – áreas em crescimento que carecem de talentos.
- **Protagonismo e Inovação:** Quem sabe programar e entende de robótica pode criar suas próprias soluções para problemas. Em vez de ficar limitado ao que já existe no mercado, você pode inventar. Se na sua escola há algum desafio (por exemplo, melhorar a rega das plantas do jardim, ou automatizar

alguma tarefa), você tem a capacidade de pensar em soluções tecnológicas. Isso faz de você um **protagonista**, não apenas um consumidor passivo de tecnologia.

- **Entendimento do Mundo Atual:** Saber o básico de como um computador “pensa” ou como um circuito funciona ajuda você a compreender melhor o mundo moderno. Você passa a ter um olhar mais crítico sobre as tecnologias – por exemplo, entendendo conceitos de sensores, poderá imaginar como funcionam portas automáticas, alarmes de incêndio, termostatos etc. Ao saber programar, entenderá melhor os aplicativos e sistemas que usa diariamente. Essa **alfabetização tecnológica** é importante para exercer plenamente a cidadania na era digital.
- **Desenvolvimento Socioemocional:** Pode não parecer óbvio à primeira vista, mas robótica também trabalha aspectos como paciência, persistência, cooperação e autoconfiança. Projetos tecnológicos frequentemente exigem tentar várias vezes, lidar com frustrações (quando algo dá errado) e buscar ajuda quando necessário. Ao superar esses desafios, o jovem desenvolve **resiliência** (aprender com erros e não desistir facilmente) e também **ética** (por exemplo, aprender a usar o conhecimento para coisas positivas, ter responsabilidade ao lidar com ferramentas e equipamentos, etc.).

Em resumo, robótica e programação formam uma combinação poderosa de aprendizado interdisciplinar que prepara você tanto no âmbito **cognitivo** (pensamento lógico-científico) quanto **socioemocional** (trabalho em equipe, perseverança). Conforme descrevemos, há pelo menos sete ótimos motivos para incluir robótica na rotina de estudos! Neste eBook, esperamos contribuir para que você desenvolva essas habilidades de maneira **divertida e significativa**.

Agora que entendemos o **porquê** e o **como** deste eBook, vamos colocar mãos à obra! A seguir, na Unidade 02, faremos sua primeira imersão no mundo Arduino: explicaremos o que é o Arduino Uno, como prepará-lo e escrever seu primeiro programinha. Pronto para acender seu primeiro LED? Então vamos lá!

Exercícios – Unidade 01

1. **Competências:** Cite **duas competências** ou habilidades que você irá desenvolver estudando robótica educacional. Por que elas são importantes?

2. **Interdisciplinaridade:** A robótica envolve várias disciplinas. Como conteúdos de **Matemática** e **Física** aparecem no projeto do jogo da memória? Dê um exemplo de cada.

3. **Aprender Fazendo:** Explique com suas palavras o que significa metodologia “**mão na massa**”. Você prefere aprender assim, de forma prática? Justifique.

4. **Colaboração:** Por que trabalhar em equipe pode ajudar em projetos de robótica? Descreva uma situação do projeto em que duas pessoas poderiam colaborar.

5. **Resolução de Problemas:** Se algo no circuito ou no código não funciona de primeira, o que você deve fazer? (a) Desistir imediatamente; (b) Tentar identificar o erro e corrigir, possivelmente pedindo ajuda; (c) Jogar o Arduino fora.

6. **STEAM:** A sigla STEAM resume áreas importantes integradas na robótica. O que significam as letras **A** e **M** em **STEAM** e como elas podem aparecer em um projeto de robótica?

7. **Seymour Papert:** Quem foi Seymour Papert e qual ideia dele influenciou a forma como aprendemos robótica? (Dica: Pesquise sobre “Construcionismo” se necessário.)

8. **Tecnologia na Escola:** Segundo a BNCC, os alunos devem não só usar, mas criar **tecnologias** de forma ética e responsável. Você consegue pensar em um exemplo de tecnologia que você gostaria de criar ou melhorar no futuro?

9. **Motivação:** Liste um motivo que te deixa empolgado(a) para aprender robótica e programação. Como você acha que essa aprendizagem pode ser útil fora da sala de aula?

10. **Futuro e Carreira:** Verdadeiro ou Falso – “Aprender programação e robótica só é útil para quem quer ser engenheiro ou cientista; para outras carreiras não faz diferença.” Explique sua resposta.

UNIDADE 02: Introdução ao Arduino

Bem-vindo à sua primeira incursão no mundo do **Arduino**! Nesta unidade, você vai descobrir o que exatamente é esse tal de Arduino de que tanto falamos. Vamos aprender sobre a **placa Arduino Uno**, seus componentes e pinos principais, e preparar seu computador com o **Arduino IDE** (ambiente de programação) para escrever nosso primeiro código. Ao final da unidade, você fará o clássico experimento “Olá, Mundo” do Arduino: o piscar de um LED. Além disso, explicaremos conceitos básicos de programação (como variáveis e loops) que usaremos no projeto do jogo da memória.

2.1. O que é o Arduino?

Arduino é uma plataforma de prototipagem eletrônica **de código aberto** muito popular. Isso significa que é um conjunto de hardware (placas eletrônicas) e software (programa de computador para codificar) de uso livre, projetado para ser fácil para iniciantes, mas também poderoso para usuários avançados. Em outras palavras, **o Arduino é um mini-computador programável** que podemos conectar a diversos componentes eletrônicos para criar projetos interativos.

Figura 4 - Arduino Uno Original Italia



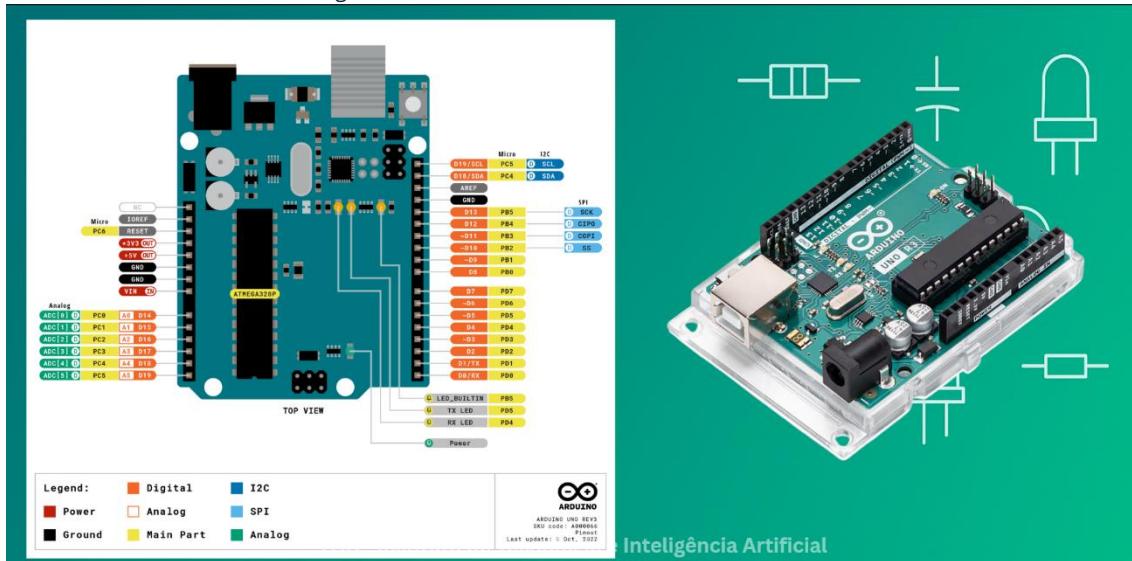
Uma explicação famosa define Arduino assim: “*Arduino é uma plataforma eletrônica open-source baseada em hardware e software fáceis de usar. As placas Arduino conseguem ler entradas – por exemplo, luz em um sensor, o pressionar de um botão – e convertê-las em saídas – acionar um motor, acender um LED, publicar algo online*”. Ou seja, ele serve de cérebro para nossos projetos, **lendo informações do mundo real e tomando decisões** conforme programado para então controlar dispositivos.

O Arduino foi criado na Itália em meados dos anos 2000, com o objetivo de oferecer uma ferramenta de prototipagem rápida para estudantes de design sem

background em eletrônica. Deu tão certo que se espalhou pelo mundo inteiro. Hoje em dia, existem vários **modelos de placas Arduino**, mas a mais clássica (e a que usamos neste eBook) é a **Arduino Uno**.

A placa Arduino Uno é pequena (do tamanho aproximado de um baralho de cartas) e relativamente barata. Ela possui um microcontrolador ATmega328P, que é como o “chip cérebro” da placa. Essa placa vem com diversas **portas de entrada e saída** que podemos usar. Segundo documentação, o Arduino Uno **possui 14 pinos digitais e 6 entradas analógicas**, além de uma porta USB para comunicação com o computador e alimentação, um conector de energia (jack DC) e alguns componentes auxiliares. Tudo isso torna o Uno uma placa versátil para inúmeros projetos – desde robôs móveis até sistemas de irrigação automatizada.

Figura 5 - Placa e Datasheet do Arduino Uno Rev 3



Em resumo, podemos imaginar o Arduino como um **controle central programável**: você cria um programa (chamado *sketch*) no computador, envia para a placa, e então a placa passa a executar esse programa indefinidamente, interagindo com o que você conectou a ela (sensores, LEDs, motores, etc.). A grande vantagem do Arduino é justamente sua facilidade: a linguagem de programação é simplificada (baseada em C/C++), a IDE (ferramenta de programação) é amigável, e existe uma enorme comunidade na Internet com exemplos e bibliotecas prontas para praticamente tudo.

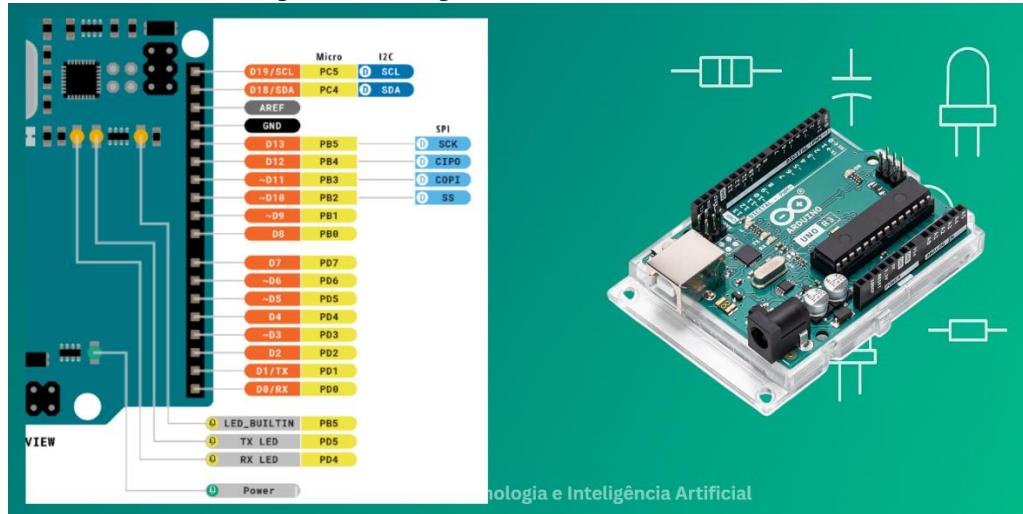
Nesta unidade vamos montar um primeiro circuito simples e aprender a programar o Arduino. Mas antes, precisamos conhecer melhor os **pinos e componentes do Arduino Uno** para saber onde ligar cada coisa com segurança.

2.2. Conhecendo a placa Arduino Uno (pinos e funções)

Pegue sua placa Arduino Uno (ou similar) em mãos, ou observe uma imagem dela. Você vai notar vários pinos (os “furinhos” com contatos metálicos) nas bordas da placa, geralmente com números e letras ao lado. Esses pinos são as conexões que usaremos para ligar LEDs, botões, etc. Vamos entender as categorias de pinos do Arduino Uno e suas funções básicas:

- **Pinos Digitais:** São marcados com números de 0 a 13 na placa. Eles podem ser usados tanto como entradas como saídas digitais. *Digital* significa que só têm dois estados possíveis: nível ALTO (HIGH = 5 Volts no Uno) ou nível BAIXO (LOW = 0 Volt). Por exemplo, se configurarmos o pino 13 como saída e o colocarmos em HIGH, ele fornecerá 5V e pode acender um LED; se colocarmos em LOW, fornece 0V (GND) e apaga o LED. Como entrada, um pino digital pode ler se em sua conexão está chegando 5V (interpretado como valor “1”) ou 0V (“0”). No Arduino Uno, os pinos digitais 0 e 1 têm função especial (RX e TX, comunicação serial), então costumamos evitar usá-los para sensores/atuadores comuns para não conflitar com a comunicação USB. Os pinos ~3, ~5, ~6, ~9, ~10 e ~11 têm um ~ ao lado do número, indicando que são capazes de **PWM** (Pulse-Width Modulation), um recurso que permite gerar saídas analógicas “falsas” a partir de um digital (usado para, por exemplo, controlar intensidade de LED ou velocidade de motor). Mas não entraremos nesse detalhe agora. Importante: cada pino digital do Uno suporta correntes de no máximo 20 mA de forma recomendada (40 mA é o absoluto máximo), por isso sempre precisamos de resistores em LEDs para não exceder essa corrente.

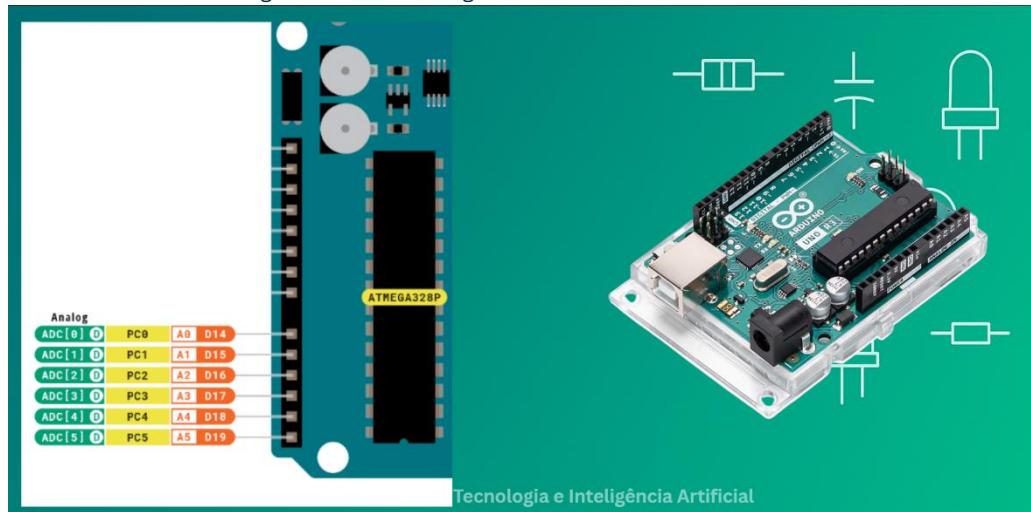
Figura 6 - Pinos digitais Arduino UNO REV 3



- **Pinos Analógicos:** No Uno, ficam separados e numerados como A0, A1, ..., A5. Eles funcionam principalmente como entradas analógicas, ou seja, podem ler um sinal que varia entre 0 e 5V e traduzir isso para um valor numérico. O Arduino Uno tem um conversor ADC de 10 bits, o que significa que 0V é lido como 0 e 5V como 1023 ($2^{10} - 1$), e valores intermediários variam proporcionalmente. Esses pinos são úteis para sensores que dão

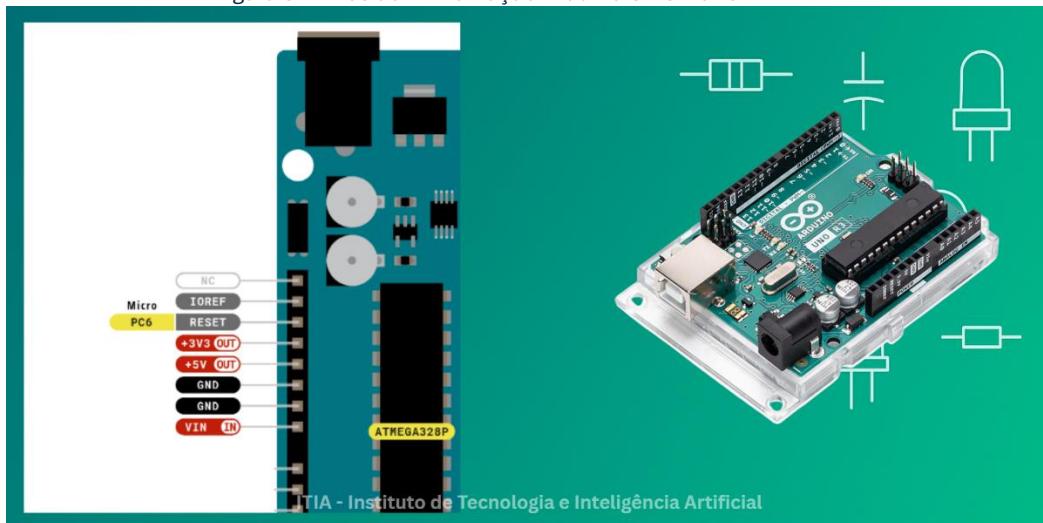
saída variável (por exemplo, um sensor de luminosidade, um potenciômetro, etc.). No nosso projeto, curiosamente, vamos usar alguns desses pinos analógicos como entradas digitais para botões – sim, eles podem fazer isso também! Os pinos A0 a A5 podem ser usados como digitais (no código você pode referenciá-los como 14, 15, etc., ou simplesmente usar “A0” e configurar como digital). Alguns modelos mais novos de Arduino têm A6 e A7, mas no Uno padrão só até A5. Vale notar: se um pino analógico não estiver ligado a nada (entrada em aberto), sua leitura “flutua” (ruído aleatório) – vamos ver depois que isso pode até servir para gerar números aleatórios.

Figura 7 - Pinos Analógicos Arduino Uno REV 3



- Pinos de Alimentação (Power):** Normalmente agrupados em um canto da placa, incluem o pino **5V**, **3.3V**, **GND** (terra) e outros. O pino 5V é a saída regulada de 5 volts – alimenta nossos componentes eletrônicos. O pino 3.3V fornece 3,3 volts (alguns sensores/modulos operam nessa tensão). Os **pinos GND** (ground) são terra comum – há vários GND na placa e todos estão interconectados internamente, use qualquer um para fechar o circuito dos componentes. Há também o pino **VIN**, que pode ser usado para alimentar a placa com uma tensão externa (6-12V) se não estiver usando o jack de energia. E o pino **IREF** (não muito usado em projetos básicos) fornece a referência de tensão de operação (5V no Uno) para shields (placas de expansão). Importante: sempre que conectar sensores ou outros módulos, lembre de conectá-los ao GND comum do Arduino, senão o circuito não fecha! A placa também tem um conector preto redondo (jack DC) onde você pode plugar, por exemplo, uma bateria de 9V ou uma fonte; essa entrada passa por um regulador para 5V. Via USB, a placa recebe 5V diretamente do computador. Nunca alimente o Arduino por vários pontos ao mesmo tempo (tipo USB e jack e VIN simultâneos) para evitar problemas.

Figura 8 - Pinos de Alimentação Arduino UNO Rev 3



- **Porta USB:** Usada para **programar a placa** (carregar o código) e também para alimentá-la durante os testes. Quando conectamos o cabo USB ao computador, o Arduino aparece como uma porta serial. Pelo USB, além de enviar o programa, podemos abrir um *Monitor Serial* (ferramenta do IDE) para enviar/receber mensagens de texto do Arduino – útil para depuração. No projeto do jogo da memória não vamos precisar do monitor serial, mas é bom saber que ele existe. Se o Arduino estiver sendo alimentado pela tomada ou bateria e desconectado do PC, obviamente o USB não estará ativo para comunicação.

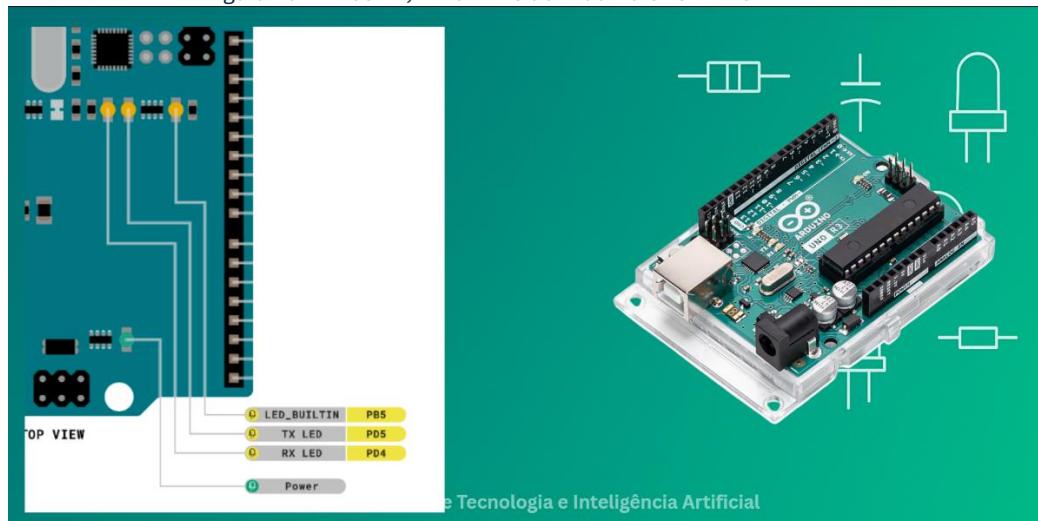
Figura 9 - Porta USB para transferência de código



- **LEDs da placa:** A placa Arduino Uno já tem alguns LEDs indicativos soldados nela. Um deles é o LED “L” (de “Led”, às vezes identificado) conectado internamente ao **pino digital 13**. Isso significa que se fazermos digitalWrite(13, HIGH), esse LED na placa acende (mesmo sem conectar nada externo). Isso é ótimo para testes iniciais, como o nosso primeiro código de piscar LED. Existem também LEDs indicadores de **Power (ON)** – acende quando a placa está energizada – e **RX** e **TX** que piscam quando há

comunicação serial (envio e recebimento de dados pelo USB, por exemplo durante upload de código).

Figura 10 - Pinos RX, TX e LEDs do Arduino Uno REV 3



- Outros componentes da placa:** O Arduino Uno possui ainda um botão de **RESET** (que reinicia o programa do início quando pressionado), e conectores como o **ICSP** (6 pinos usados para programar o microcontrolador via protocolo SPI, ou para alguns módulos como leitor de cartão SD – não entraremos nisso aqui). Há também um cristal oscilador de 16 MHz que serve de “relógio” para o Arduino, garantindo que ele execute instruções no tempo correto.

Para visualizar melhor tudo isso, veja abaixo um **diagrama dos pinos do Arduino Uno**:

Figura 11 - Pinagem completa da Placa Arduino Uno REV 3

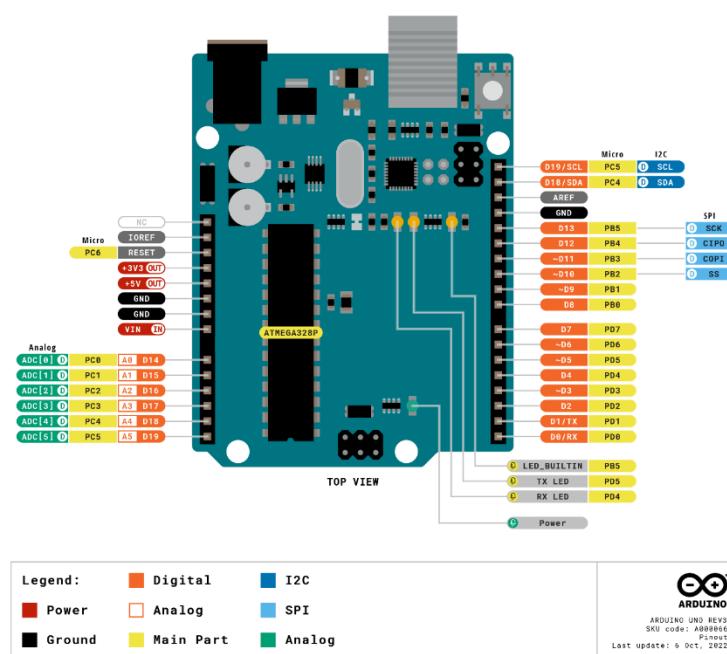


Diagrama de pinagem do Arduino Uno R3, mostrando os pinos digitais (em azul à direita, 0 a 13), pinos analógicos (em verde à esquerda, A0 a A5), pinos de alimentação (em vermelho, 5V, 3.3V, GND, VIN) e demais conexões do microcontrolador ATmega328P.

Como podemos ver na imagem acima, cada pino tem uma “função por trás” no microcontrolador, mas felizmente não precisamos nos preocupar com os nomes técnicos do chip – usamos as marcações da placa (que já são simples, como 13, 12, A0, etc.). Sempre que for ligar um componente, certifique-se de **ler o número ao lado do pino** e seguir as instruções do projeto para não conectar errado. Ao longo do eBook, sempre indicaremos algo como “conecte o LED verde ao pino 8” – isso significa conectar o positivo do LED ao orifício marcado com “8” na seção Digital.

Recapitulando:

- Para **acender/apagar algo** (ex.: LED, buzzer), usamos **pinos digitais configurados como saída**.
- Para **ler estado LIGADO/DESLIGADO** (ex.: um botão pressionado ou não), usamos **pinos digitais como entrada** (muitas vezes com resistores de *pull-up* ou *pull-down* para garantir leitura estável, assunto que abordaremos mais adiante).
- Para **ler um valor variável** (ex.: posição de um potenciômetro, intensidade de luz de um sensor LDR), usamos **pinos analógicos como entrada** e obtemos um número de 0 a 1023 proporcional ao sinal.
- Todos os dispositivos externos precisam compartilhar o **GND comum** da placa, e podem requerer alimentação 5V ou 3.3V se forem ativos (no nosso projeto, usaremos apenas 5V e GND para componentes simples).
- Podemos alimentar o Arduino tanto pelo USB quanto por uma fonte externa (bateria) conforme necessidade – mas enquanto programamos, ele ficará conectado ao computador via USB de qualquer forma.

Agora que sabemos o que é e do que é feita a plaquinha Arduino Uno, vamos aprender a usá-la! O próximo passo é instalar em seu computador o **Arduino IDE**, que é o programa onde escreveremos os códigos (sketches) e enviaremos para a placa.

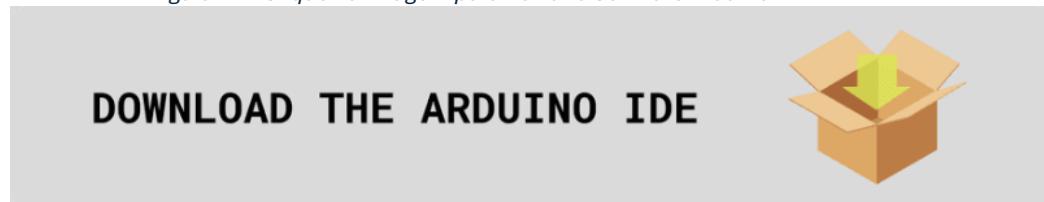
2.3. Instalando o Arduino IDE

O **Arduino IDE (Integrated Development Environment)** é o ambiente de programação do Arduino. Nele, escrevemos o código, verificamos/compilamos e fazemos o upload para a placa. A IDE é gratuita e está disponível para Windows, Mac e Linux.

Passos para instalar:

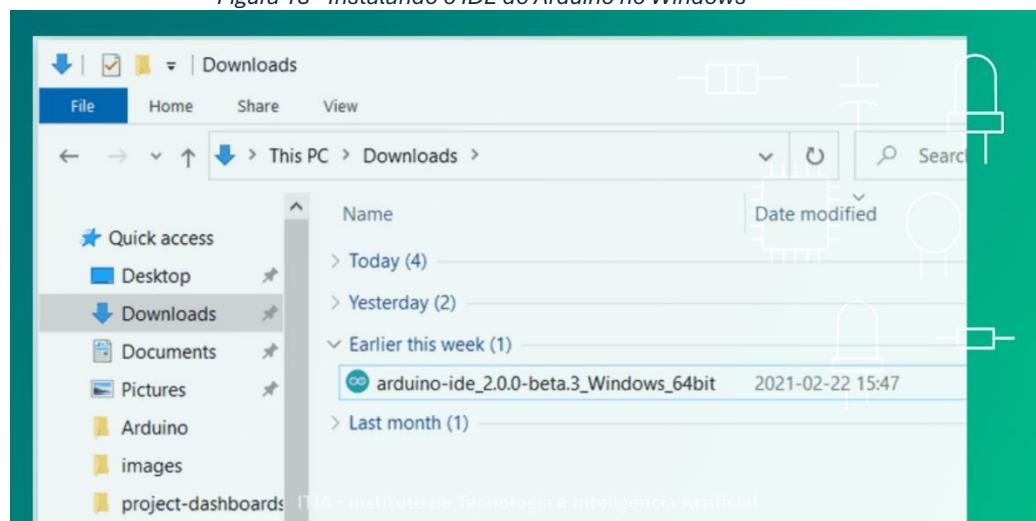
1. **Baixar o Arduino IDE:** Acesse o site oficial arduino.cc e baixe a versão apropriada para seu sistema operacional. Há versões instaláveis e uma versão portátil (ZIP). Baixe a versão mais recente estável.

Figura 12 - Clique na Imagem para Baixar o Software Arduino IDE



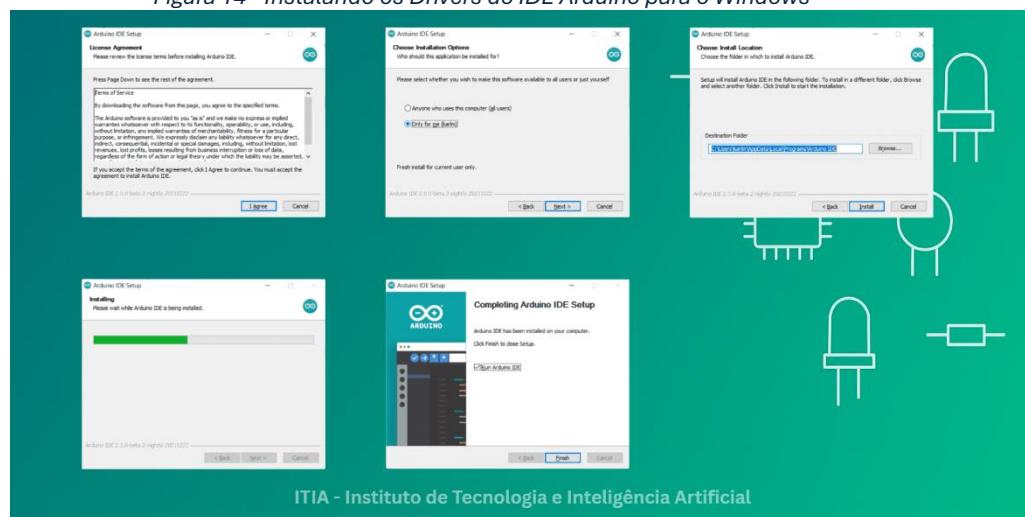
- Instalar:** No Windows, execute o arquivo .exe baixado e siga as instruções (geralmente basta concordar e avançar). No Mac, arraste o aplicativo para a pasta Aplicativos. No Linux, pode ser necessário descompactar e rodar um script de instalação ou instalar via gerenciador de pacotes dependendo da distribuição.

Figura 13 - Instalando o IDE do Arduino no Windows



- Drivers (Windows):** Durante a instalação no Windows, permita a instalação dos drivers (eles permitem que o PC se comunique com o Arduino pela USB). No Mac e Linux, geralmente não é necessário driver extra, pois o sistema já reconhece como porta serial.

Figura 14 - Instalando os Drivers do IDE Arduino para o Windows



4. **Abrir a IDE:** Após instalado, abra o Arduino IDE. Ele deve exibir uma janela de edição em branco ou talvez um exemplo padrão.

Figura 15 - Abrindo pela primeira vez o IDE do Arduino



Primeira configuração:

- Selecionar placa e porta:** Conecte seu Arduino Uno ao computador via cabo USB. No IDE, vá em Ferramentas > Placa > Arduino AVR Boards > Arduino Uno (se não estiver selecionado automaticamente). Ainda em Ferramentas, vá em Porta e selecione a porta que corresponde ao Arduino (no Windows aparecerá algo como COM3, COM4..., no Mac/Linux algo como /dev/ttyACM0 ou /dev/ttyUSB0). Geralmente a porta aparece identificada com “Arduino Uno” ao lado na lista, o que ajuda.

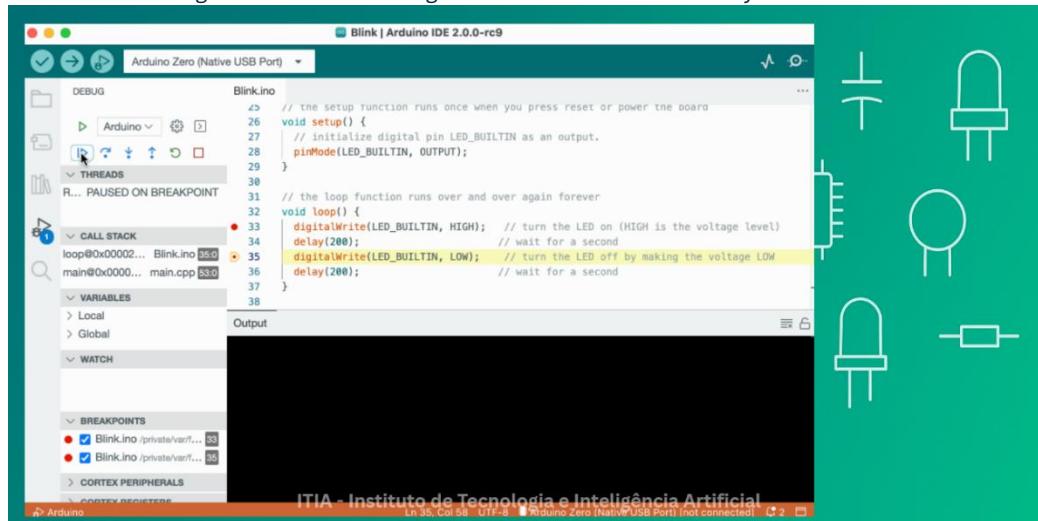
Figura 16 - Selecionando a porta USB e a placa Arduino



- Teste de exemplo (Blink):** Uma boa prática é testar se está tudo ok carregando um exemplo básico. No menu Arquivo > Exemplos > 01.Basics > Blink. Isso abrirá o código de piscar LED de exemplo. Clique no botão “” (Verificar/Compilar) para ver se não há erros. Depois clique no botão “” (Upload) para enviar para a placa. O LED “L” do Arduino (pino 13) deve

começar a piscar (1 segundo aceso, 1 segundo apagado). Se isso aconteceu, parabéns, seu Arduino está funcionando e pronto para ser programado! Se não, revise as etapas (placa/porta corretas, drivers instalados, etc.).

Figura 17 - Primeiro código com Arduino - Pisca Led - Blynk



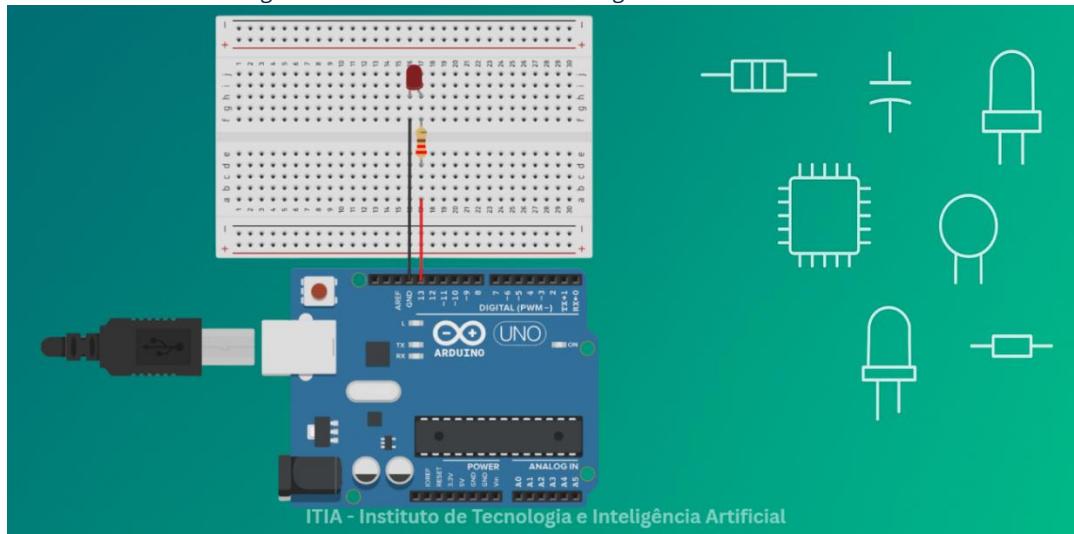
Agora que o ambiente está configurado, vamos criar **nossa próprio código** simples para entender a estrutura de programação Arduino.

2.4. Escrevendo e Executando o Primeiro Código: “Blink” (Piscar LED)

Vamos recriar o clássico exemplo **Blink** – piscar um LED – mas escrevendo o código do zero para aprender a estrutura básica de um programa Arduino.

Circuito mínimo: Você pode usar o LED embutido no pino 13, que já está conectado internamente. Se quiser usar um LED externo, monte assim: conecte um LED ao pino 13 (anodo do LED, perna positiva) e a outra perna do LED (cátodo) a um resistor de $\sim 220\ \Omega$, e o outro terminal do resistor ao GND. (O resistor é importante para limitar a corrente, evitando que o LED queime ou o pino do Arduino exceda a corrente segura.)

Figura 18 - Circuito Básico de Montagem Prática Pisca LED



Vamos usar o LED interno mesmo para simplificar – assim nem precisa montar nada extra. Abra o Arduino IDE, crie um novo sketch (Arquivo > Novo) e digite o seguinte código:

```
//  
https://www.tinkercad.com/things/FYCAQ8bAVW0/editel?returnTo=%2Fdashboard&sharecode=L  
8Jh8och7tVxSuY9ce50t-wSb-rlvhkovk4xUqO4lhU  
  
// Pisca-LED básico no Arduino  
int LED = 13;  
  
void setup() {  
    pinMode(LED, OUTPUT); // configura o LED embutido (pino 13) como  
    // saída  
}  
  
void loop() {  
    digitalWrite(LED, HIGH); // liga o LED (5V no pino)  
    delay(1000); // espera 1000 milissegundos (1  
    // segundo)  
    digitalWrite(LED, LOW); // desliga o LED (0V no pino)  
    delay(1000); // espera 1 segundo  
}
```

Vamos entender este código, mesmo que curinho, pois ele ilustra a estrutura básica de qualquer programa Arduino:

- Tudo que está após // em uma linha é comentário (nota explicativa que o Arduino ignora ao executar).
- Existem duas funções especiais: **setup()** e **loop()**.
 - **setup()** é executada **uma única vez** assim que o Arduino é ligado ou resetado. Usamos ela para configurações iniciais. No caso, estamos configurando o modo do pino do LED embutido.
 - **loop()** vem em seguida e é executada **continuamente em loop**, repetidamente, até que o Arduino seja desligado. É o coração do programa, onde colocamos as ações que queremos que fiquem rodando.
- **pinMode(LED, OUTPUT);** diz ao Arduino que o pino do LED (constante **LED_BUILTIN** corresponde ao número 13 no Uno) será usado como **saída**. Isso é necessário para poder acender/apagar o LED via código.
- Dentro de **loop()**, fazemos: **digitalWrite(LED, HIGH);** que coloca 5V no pino do LED, acendendo-o. Em seguida **delay(1000);** pausa o programa por 1000 milissegundos (1 segundo). Depois **digitalWrite(LED, LOW);** põe 0V no pino, apagando o LED, e outro **delay(1000);**. Assim, o LED fica 1 segundo aceso e 1 segundo apagado. Quando **loop()** termina, ele recomeça automaticamente, então isso pisca indefinidamente.

Se carregar este código no Arduino (não esqueça de salvar o arquivo com nome, ex: **Blink_meu**), você verá o LED “L” piscar em intervalos de 1s. Experimente modificar os valores de delay para piscar mais rápido ou devagar (por exemplo, 200 ms ao invés de 1000, ou 1000 aceso / 200 apagado, etc.) e faça upload novamente para ver a mudança.

Parabéns – você escreveu e executou seu primeiro programa Arduino! No nosso projeto final (jogo da memória), obviamente o código será mais complexo, mas ele seguirá essa mesma estrutura básica de `setup()` + `loop()`. Em `setup()` configuraremos os pinos (por exemplo, definir pinos dos LEDs e botões como `OUTPUT` ou `INPUT`), e no `loop()` ficará a lógica do jogo que se repete continuamente.

2.5. Conceitos Básicos de Programação Aplicados

Antes de prosseguirmos para montar o circuito do projeto, vamos revisar alguns **conceitos de programação** que já apareceram ou aparecerão no código, para garantir que todo mundo os entenda. Usaremos exemplos simples relacionados ao que faremos no jogo da memória:

- **Variáveis:** São como “caixinhas” onde guardamos valores (números, texto etc.) enquanto o programa executa. No Arduino (C/C++) precisamos declarar o tipo da variável. Exemplos:
 - `int pontuacao = 0;` – aqui criamos uma variável inteira chamada *pontuacao* e já iniciamos com zero. Conforme o jogo progride, podemos aumentar essa variável quando o jogador acerta uma sequência.
 - `int luzes[200];` – isso declara um **vetor/array** de inteiros de tamanho 200, chamado *luzes*. Vamos usá-lo no jogo para guardar a sequência de LEDs que devem acender. Por exemplo, `luzes[0] = 3` pode indicar que a primeira luz da sequência é a de cor 3 (vamos numerar as cores).
 - Variáveis podem mudar de valor durante a execução. Ex: `pontuacao = pontuacao + 1;` aumenta a pontuação (ou sintaxe abreviada `pontuacao++;`).
 - Usaremos tipos como `int` (inteiro), `bool` (booleano verdadeiro/falso), `long` (inteiro longo), `float` (para números com decimal, embora talvez não precise aqui), e também constantes definidas via `#define` ou `const`. No `blink`, `LED_BUILTIN` é uma constante que o Arduino já define internamente (como 13).

Exemplo de código:

```
int countUp = 0; //creates a variable integer called 'countUp'

void setup() {
    Serial.begin(9600); // use the serial port to print the number
}

void loop() {
    countUp++; //Adds 1 to the countUp int on every loop
    Serial.println(countUp); // prints out the current state of
    countUp
    delay(1000);
}
```

- **Funções:** São blocos de código que realizam uma tarefa específica e podem ser chamados (invocados) quando precisamos daquela tarefa. Já vimos duas funções especiais (`setup` e `loop`). Outras funções que usamos:

- `digitalWrite(x, HIGH)` – é uma função pronta do Arduino que escreve nível HIGH ou LOW no pino x.
- `pinMode(x, OUTPUT)` – configura modo do pino.
- `delay(ms)` – pausa o programa por ms milissegundos.
- Podemos também criar nossas próprias funções, caso queira organizar o código. Por exemplo, poderíamos fazer:

```
void acenderLedVerde() {
    digitalWrite(pinLedVerde, HIGH);
    tone(pinBuzzer, 262, 500); // toca som Dó (262 Hz) por
    0.5s
}
```

Assim sempre que quisermos acender o LED verde e tocar o som correspondente, basta chamar `acenderLedVerde()`; no código principal, ao invés de repetir as duas linhas. Funções ajudam a evitar repetição e deixam o código mais legível.

- No nosso jogo, não será estritamente necessário criar funções customizadas, mas fica a dica para organizar código em projetos maiores.

Código de exemplo:

Controle o LED embutido do Arduino no pino 13 (saída) atribuindo o mesmo valor de um botão conectado ao pino 7 (entrada).

```

int ledPin = 13; // LED connected to digital pin 13
int inPin = 7; // pushbutton connected to digital pin 7
int val = 0; // variable to store the read value

void setup() {
    pinMode(ledPin, OUTPUT); // sets the digital pin 13 as output
    pinMode(inPin, INPUT); // sets the digital pin 7 as input
}

void loop() {
    val = digitalRead(inPin); // read the input pin
    digitalWrite(ledPin, val); // sets the LED to the button's value
}

```

- Estruturas de Decisão (Condicionais):** São os famosos **if / else**, que permitem que o programa tome decisões de acordo com alguma condição booleana (verdadeiro/falso). Por exemplo, no jogo teremos algo assim:

```

if (botaoVerdeApertado) {
    // se o botão verde foi apertado...
    if (sequencia[indice] == VERDE) {
        // se a cor esperada na sequência era verde, então acertou
        indice++;
    } else {
        // caso contrário, errou
        jogoAtivo = false;
    }
}

```

Em português: *se o botão verde for pressionado, então verifica: se a sequência esperada naquele passo era a cor verde, então o jogador acertou e podemos passar para o próximo índice; senão, o jogador errou e o jogo acaba.* Condicionais são **fundamentais** para implementar lógica de jogo, porque precisamos reagir a eventos (um botão apertado pode ser certo ou errado dependendo da situação). Também usamos if para coisas simples, por exemplo: *se a pontuação atingir determinado valor, poderia aumentar a velocidade do jogo* (no nosso código base não há isso, mas poderíamos incrementar desafio).

O if pode ter apenas a parte *if* e *else*, ou até *else if* em cascata para múltiplas possibilidades. Também podemos ter condições compostas usando operadores lógicos: AND (**&&**), OR (**||**), NOT (**!**).

Código de exemplo:

Os colchetes podem ser omitidos após uma instrução if. Se isso for feito, a próxima linha (definida pelo ponto-e-vírgula) se tornará a única instrução condicional.

```
if (x > 120) digitalWrite(LEDpin, HIGH);  
if (x > 120) digitalWrite(LEDpin, HIGH);  
if (x > 120) {digitalWrite(LEDpin, HIGH);}  
  
if (x > 120) {  
    digitalWrite(LEDpin1, HIGH);  
    digitalWrite(LEDpin2, HIGH);  
}  
// all are correct
```

- **Laços de Repetição (Loops):** Já temos o loop principal void loop(), mas dentro dele ou de outras funções podemos usar estruturas de repetição como **for**, **while** e **do...while** para repetir um bloco de código várias vezes.

- O for é útil quando sabemos quantas vezes repetir. Exemplo do blink: usamos for para piscar 10 vezes:

```
for(int i=0; i<10; i++) {  
    digitalWrite(LED_BUILTIN, HIGH);  
    delay(200);  
    digitalWrite(LED_BUILTIN, LOW);  
    delay(200);  
}
```

Isso repetiria o bloco acender/apagar 10 vezes porque a variável i vai de 0 até 9.

- No jogo, usaremos for para exibir a sequência de luzes acumulada. Por exemplo, se quant for a quantidade de passos na sequência atual, faremos um loop for(i = 0; i < quant; i++) para acender cada LED correspondente na ordem.

```
// Brighten an LED using a PWM pin  
int PWMpin = 10; // LED in series with 470 ohm resistor from pin  
10 to ground  
  
void setup() {  
    // no setup needed  
}  
  
void loop() {  
    for (int i = 0; i <= 255; i++) {  
        analogWrite(PWMpin, i);  
        delay(10);  
    }  
}
```

- O while é utilizado para repetir enquanto uma condição for verdadeira. No código do jogo, há um while(inicio != 1023) que espera até o valor lido do botão inicial ser 1023 (ou seja, até o botão ser pressionado). Esse laço fica "travado" ali até a condição mudar. Usamos while também para esperar o jogador apertar botões em sequência.
- É preciso ter cuidado com while para não criar loops infinitos acidentalmente. No Arduino, se um while nunca terminar, o programa "congela" ali (exceto se tiver escapes como break). Mas às vezes queremos justamente esperar até algo acontecer, e o while é apropriado.
- O do...while é similar ao while, mas primeiro executa o bloco, depois verifica a condição (garantindo que rode pelo menos uma vez). Não é muito usado em Arduino normalmente, mas existe.
- No contexto do Arduino, dado que loop() já repete para sempre, muitos códigos nem usam while(true) ou similares, pois isso impediria o loop principal de rodar adequadamente. Porém, loops internos para certas funcionalidades (como esperar um botão soltar) são comuns.

```
var = 0;
while (var < 200) {
    // do something repetitive 200 times
    var++;
}
```

- **Comentários:** Ok, não é bem um conceito de lógica, mas é importante. Comentar o código ajuda você e os outros a entenderem o que ele faz. No código do jogo que estudaremos, você verá comentários indicando cada parte. Sempre escreva pelo menos comentários essenciais, por exemplo, ao definir uma variável, indique o que ela representa (pontuação, vetor de sequência, etc.). Um código bem comentado é como um livro aberto do seu pensamento.

Esses são os pilares básicos. Se você nunca programou antes, não se preocupe em decorar tudo imediatamente. Quando formos apresentar o código do jogo da memória (na Unidade 05), vamos relembrar cada um desses conceitos no contexto do código de verdade. Programação se aprende praticando: lendo exemplos e escrevendo os seus. Então, se possível, tente brincar com o Arduino IDE – modifique o exemplo do blink, acrescente mais LEDs se tiver, teste outros exemplos da Arduino (tem exemplo de buzzer, de leitura analógica, etc.). Essa experimentação vai te deixar mais confortável para entender o código do projeto principal.

2.6. Arduino e o Jogo da Memória: visão adiante

Para conectar com o que vem pela frente: no projeto do **Jogo da Memória (Genius)**, usaremos 4 LEDs de cores diferentes e 4 botões correspondentes a esses LEDs. O Arduino vai gerar uma sequência aleatória de cores (acendendo LEDs um a um) e o jogador deve repetir apertando os botões. A cada rodada bem-sucedida, a sequência fica mais longa (e o jogo mais difícil). Usaremos também um **buzzer** para emitir sons (cada cor tem um tom musical diferente, como no brinquedo Genius original), e um **display LCD** para mostrar mensagens (como “Vamos jogar?”, a pontuação, etc.). Parece bastante coisa, mas cada elemento funciona de forma relativamente simples – e já temos os conhecimentos básicos para entendê-los:

- O Arduino controlará os LEDs e o buzzer via pinos digitais (saídas).
- Lerá os botões via pinos (entradas).
- Gerará números aleatórios para escolher as cores (há funções prontas para isso).
- Manterá variáveis para pontuação, sequência, etc.
- Usará condicionais e loops para comparar o input do jogador com a sequência correta.
- Usará a biblioteca do LCD para escrever textos no visor.

Dica: O Arduino possui uma função chamada `random()` e `randomSeed()` que usaremos no jogo para gerar sequências imprevisíveis. A semente de aleatoriedade pode ser inicializada lendo uma entrada analógica desconectada (que fornece um valor “ruído”), garantindo sequências diferentes a cada partida. Fique tranquilo que explicaremos isso no momento certo no código.

Por enquanto, você já deu um grande passo entendendo o essencial do Arduino e testando um código simples. No projeto principal, construiremos sobre essa base sólida.

Código de exemplo

O código gera números aleatórios usando as diferentes variantes da função e os imprime no Serial Monitor.

```
long randNumber;

void setup() {
  Serial.begin(9600);

  // if analog input pin 0 is unconnected, random analog
  // noise will cause the call to randomSeed() to generate
  // different seed numbers each time the sketch runs.
  // randomSeed() will then shuffle the random function.
  randomSeed(analogRead(0));
}

void loop() {
  // print a random number from 0 to 299
  randNumber = random(300);
  Serial.println(randNumber);
```

```
// print a random number from 10 to 19
randNumber = random(10, 20);
Serial.println(randNumber);

delay(50);
}
```

Resumo Rápido (Unidade 02):

- Arduino é uma plataforma hardware+software para criar projetos interativos. O modelo Uno tem 14 pinos digitais e 6 analógicos, entre outros.
- Os pinos digitais servem para **saídas** (ligar/desligar componentes) ou **entradas** (ler liga/desliga de sensores/botões). Nível HIGH ≈ 5V, LOW = 0V.
- Pinos analógicos (A0–A5) servem para ler sinais variáveis entre 0–5V, resultando num valor 0–1023. Também podem funcionar como digitais se preciso.
- A placa fornece pinos de alimentação: 5V, 3.3V, GND, etc., e tem LED embutido no pino 13.
- Instalamos a IDE Arduino no PC, conectamos a placa via USB, escolhemos porta e placa corretas, e carregamos um primeiro programa (blink). Isso nos garante que o ambiente está ok.
- A estrutura básica de código Arduino: função setup() (executa uma vez no início) e loop() (executa repetidamente).
- Vimos um exemplo de código que pisca um LED, usando funções pinMode, digitalWrite e delay.
- Conceitos de programação: variáveis (armazenam valores, tipos int, etc.), funções (blocos reutilizáveis de código, além das nativas do Arduino), condicionais (if/else para lógica de decisão) e loops (for, while para repetição controlada).
- Essas ferramentas serão usadas para construir a lógica do jogo da memória adiante.

No próximo capítulo (Unidade 03), vamos sair um pouco do software e mergulhar de volta no **hardware**: aprender fundamentos de eletrônica para compreender os componentes que usaremos no projeto. Isso inclui entender por que precisamos de resistores com LEDs, o que um potenciômetro faz, como funcionam os botões e o buzzer, e mais. Assim, quando formos montar de verdade (Unidade 04), você saberá exatamente o que está conectando e por quê. Até lá!

Exercícios – Unidade 02

1. **Definição:** Em suas palavras, o que é o **Arduino** e para que ele serve?

2. **Placa Uno:** Quantos pinos digitais e quantos pinos analógicos o Arduino Uno possui? Para que servem os pinos GND e 5V?

3. **LED Embutido:** O Arduino Uno tem um LED embutido ligado a um pino específico. Qual é esse pino e como fazemos para acendê-lo via código?

4. **IDE Arduino:** Qual a função do Arduino IDE? Descreva os passos básicos para escrever um código e carregá-lo na placa (mencione: selecionar porta, placa, compilar, upload...).

5. **Estrutura de Código:** Qual a diferença entre as funções `setup()` e `loop()` em um programa Arduino? Quando cada uma é executada?

6. **Variáveis:** O que são variáveis em programação? Dê um exemplo de uma variável que poderíamos usar no jogo da memória (diga o nome, o tipo e o que ela representaria).

7. **Condicional:** No contexto do jogo, precisamos comparar se o jogador apertou o botão correto. Que estrutura de programação usamos para fazer

algo somente se uma condição for verdadeira? Escreva um pseudo-código de exemplo comparando uma variável corEsperada com uma variável corApertada (não precisa ser sintaxe exata de C, apenas lógica).

8. **Loop (repetição):** Se quisermos que o Arduino cheque constantemente se um botão foi pressionado (por exemplo, o botão iniciar), que tipo de estrutura ou abordagem podemos usar no código?

9. **Piscar LED:** No código de piscar LED, usamos a função delay(1000). O que acontece se colocarmos um delay muito grande, tipo delay(10000) (10 segundos)? E se usarmos um delay bem pequeno, tipo delay(50)?

10. **Desafio de Código:** Suponha que temos um LED no pino 9 e queremos piscar ele 5 vezes rápido (0,2s acesso, 0,2s apagado). Escreva um pequeno trecho de código usando um loop for que faria isso. (Dica: parecido com o blink, mas usando for).

UNIDADE 03: Eletrônica Aplicada ao Projeto

Nesta unidade, vamos aprender os **fundamentos de eletrônica** necessários para montar e entender o circuito do nosso jogo da memória. Mesmo que você nunca tenha estudado eletricidade antes, não se assuste – vamos explicar de forma simples os conceitos-chave: corrente, tensão, resistência e a famosa Lei de Ohm. Além disso, apresentaremos cada componente eletrônico usado no projeto, detalhando sua função e como conectá-lo corretamente. Ao final desta unidade, você terá conhecimento suficiente para identificar e manusear resistores, LEDs, botões, potenciômetros, displays LCD (com módulo I2C), buzzers, baterias e a protoboard (placa de montagem). Também aprenderá a calcular valores básicos para não queimar componentes e dimensionar corretamente resistores no circuito.

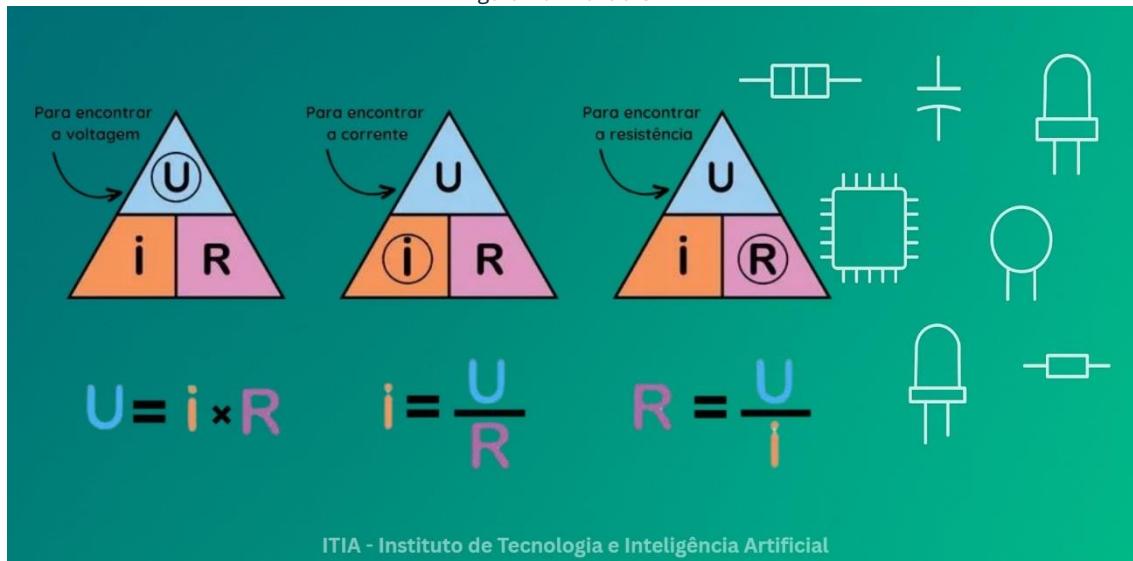
3.1. Noções Básicas de Eletricidade: corrente, tensão e resistência

Eletricidade básica: Imagine que os componentes eletrônicos são percorridos por um "fluxo" de algo invisível – esse algo é a **corrente elétrica**, que consiste em elétrons se movendo pelos fios e componentes. Para que haja corrente, precisamos de um circuito fechado: uma fonte de energia (por exemplo a bateria ou porta 5V do Arduino) empurra os elétrons através dos componentes e eles retornam pelo terra (GND) até a fonte, num ciclo contínuo. Se o circuito estiver aberto (um fio desconectado, um botão desligado), a corrente para, assim como água não flui num cano interrompido.

- **Corrente elétrica (I):** é a taxa de fluxo de carga elétrica, medida em **ampères (A)**. Pode pensar como a quantidade de elétrons passando por um ponto do circuito por segundo. No nosso projeto, as correntes são pequenas, medidas em miliampères (mA). Um LED típico consome algo entre 5 a 15 mA, por exemplo.
- **Tensão elétrica (U ou V):** é a **diferença de potencial elétrico** entre dois pontos, medida em **volts (V)**. É como a "força" que empurra a corrente pelo circuito. Nossa fonte principal será o Arduino, que fornece 5V entre seus pinos 5V e GND. A bateria de 9V fornece 9V entre seus terminais. Componentes como LEDs têm uma queda de tensão (por exemplo ~2V no LED), etc.
- **Resistência elétrica (R):** é a oposição à passagem de corrente, medida em **ohms (Ω)**. Todo material oferece alguma resistência (uns pouquíssima, como fios de cobre; outros muita, como um resistor de 10 k Ω). Resistores são componentes feitos especificamente para oferecer uma resistência conhecida no circuito, permitindo controlar a corrente.

Essas três grandezas estão relacionadas pela **Lei de Ohm**, fundamental em eletrônica: a tensão é igual à resistência multiplicada pela corrente. Em fórmula: $U = R * I$. Podemos rearranjar: $I = U/R$ ou $R = U/I$. Essa relação vale para muitos componentes (os chamados dispositivos ôhmicos, como resistores e fios, a temperaturas constantes).

Figura 19 - Lei de Ohm



Exemplo simples: se você tem uma bateria de 9V e conecta um resistor de $100\ \Omega$ entre o terminal positivo e o negativo, qual corrente irá fluir? Pela Lei de Ohm, $I = U/R = 9 / 100 = 0,09\ A$, ou seja, $90\ mA$. Se diminuirmos o resistor para $50\ \Omega$, $I = 9/50 = 0,18\ A$ ($180\ mA$). Ou seja, **menos resistência -> mais corrente, mais resistência -> menos corrente**, dado a mesma tensão.

Por que isso importa? Porque os componentes têm limites de corrente e tensão. Um LED, por exemplo, acende brilhante com $\sim 10\ mA$, mas se você deixar passar $100\ mA$ por ele, provavelmente queimaré. Por isso adicionamos resistores em série com LEDs: para limitar a corrente que passa pelo LED, protegendo-o. Sem resistor, ligando um LED diretamente nos $5V$ e GND , a corrente seria altíssima (limitada quase só pela interna do LED e do pin do Arduino) – resultado: LED queima e possivelmente o pin do Arduino também pode ser danificado pois excede $40\ mA$. Então lembre: **resistores são seus amigos para controlar a corrente!**

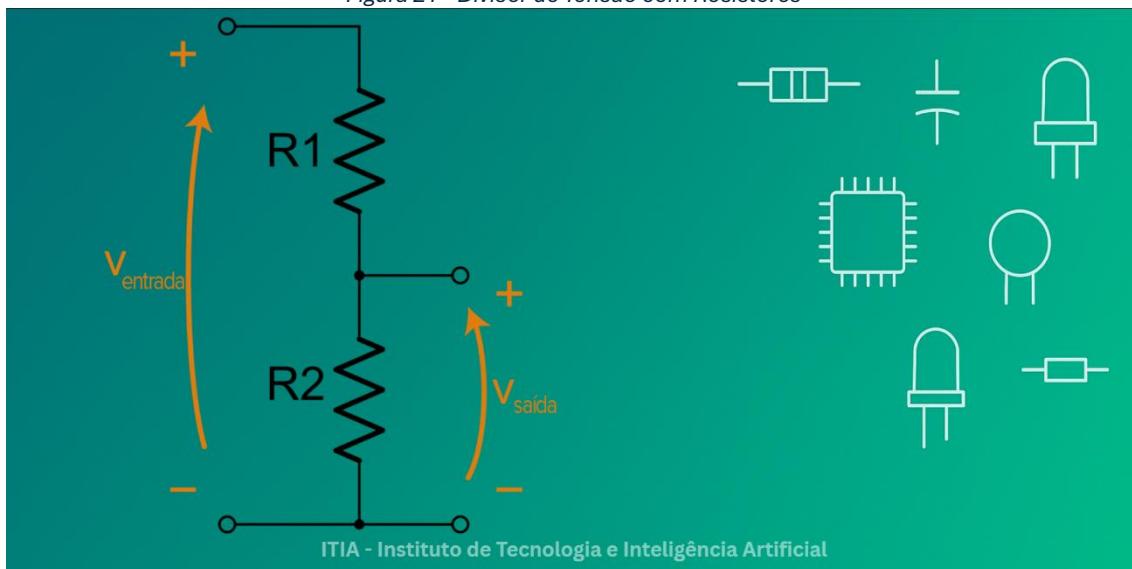
Circuito em série e paralelo: Nosso projeto basicamente terá subcircuitos em paralelo controlados pelo Arduino. Por exemplo, cada LED com seu resistor forma um caminho separado ligado ao Arduino, todos compartilhando o mesmo $5V$ e GND no final. Em paralelo, as tensões em cada ramo são as mesmas ($5V$ fornecidos), mas correntes variam de acordo com resistências. Em série, a corrente seria a mesma em todos elementos, mas a tensão se divide. Não entraremos muito nisso, mas saiba que um LED e resistor colocados em série significa que a mesma corrente atravessa ambos, e as tensões se distribuem ($5V$ da fonte = queda no LED + queda no resistor).

Figura 20 - Associação de Resistores em Paralelo e em Série



Divisor de tensão: Um caso útil de dois resistores em série é o divisor de tensão. Coloque dois resistores R1 e R2 em série entre 5V e GND. No ponto entre eles, a tensão será uma fração do total, calculada por: $V_{no_meio} = 5V * (R2 / (R1+R2))$. Assim podemos obter, por exemplo, 2,5V se R1 = R2 (igualmente dividida), etc. Como veremos a seguir, um potenciômetro é basicamente um resistor variável que atua como divisor de tensão (o meio dele fornece uma tensão ajustável). Em suma, **um divisor de tensão pega uma tensão de entrada maior e “divide” em uma saída menor fixa.** Divisores são úteis para ler sensores analógicos ou adaptar níveis, mas cuidado: eles não servem para fornecer correntes altas, só sinal mesmo.

Figura 21 - Divisor de Tensão com Resistores



Com essa base, vamos item a item nos **componentes eletrônicos** do projeto nos próximos capítulos.

3.2. Resistores

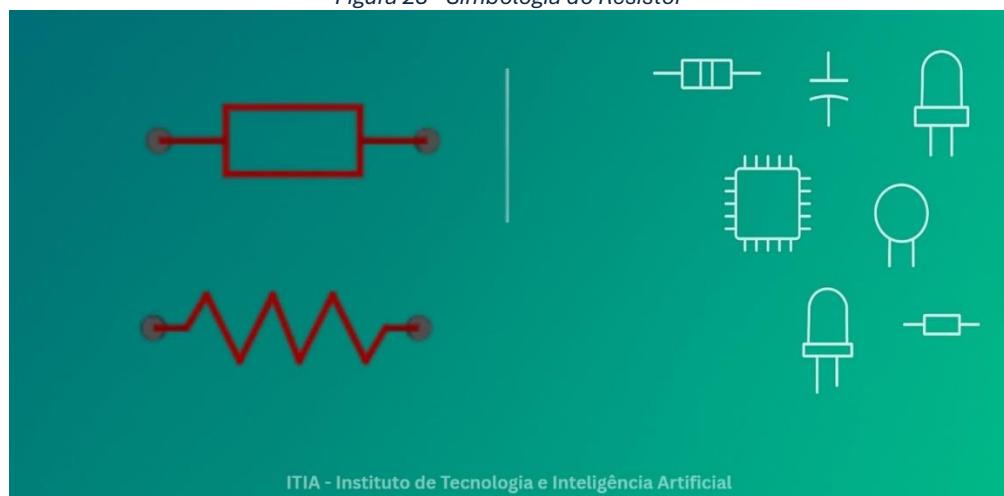
Resistor é um pequeno componente cilíndrico (com listras coloridas) cuja função é, como o nome sugere, oferecer resistência ao fluxo de corrente. Eles são especificados em ohms (Ω). No kit do projeto, usamos resistores de **220 Ω** (ohms) para os LEDs. Provavelmente eles vêm com código de cores Vermelho-Vermelho-Marrom (e outra cor para tolerância). Esse valor limita a corrente do LED a algo em torno de 10-15 mA, o que é seguro. Vamos fazer um cálculo rápido: LED geralmente tem $\sim 2V$ de queda. Então em 5V sobrariam 3V no resistor. Pela lei de Ohm: $I = 3V / 220\Omega \approx 0,0136 A = 13,6$ mA. Perfeito. Poderíamos usar 330 Ω também (daria ~ 9 mA, LED acenderia um pouco menos forte, mas ainda bem visível). Já 100 Ω daria 30 mA, o que ultrapassa o ideal do Arduino (20 mA) e estressa o LED – então 220 é um meio-termo prudente. **Conclusão:** sempre coloque resistor em série com LED ao ligar no Arduino.

Figura 22 - Resistores



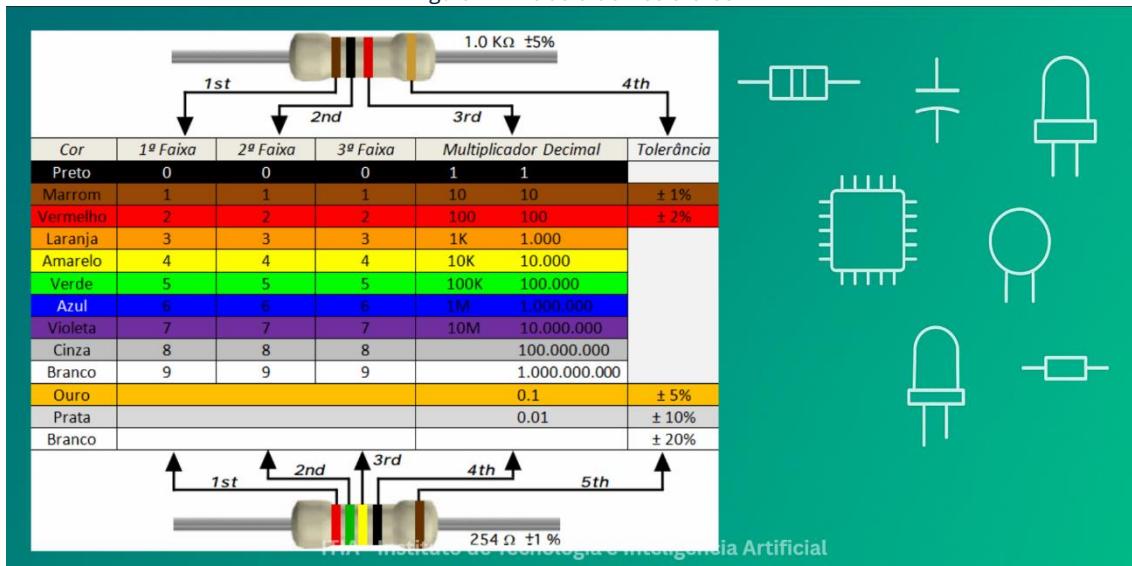
Resistores possuem **potência** também (medida em Watts) – quanto de calor eles dissipam. Os comuns de 1/4 W (0,25W) são suficientes para nossas correntes pequenas. Se passasse corrente grande, o resistor esquenta e pode queimar se exceder a potência. No nosso caso, $3V * 0,014A \approx 0,042W$, bem abaixo de 0,25W.

Figura 23 - Símbologia do Resistor



Identificando resistores: Eles têm um código de cores. Se tiver curiosidade: $220\ \Omega$ é vermelho (2), vermelho (2), marrom (multiplicador $10^1 = 10$, então $22 * 10 = 220$), e geralmente uma quarta faixa dourada (tolerância 5%). Mas você não precisa decorar – pode ler a tabela ou usar multímetro. Nossa kit tem resistores já separados pelo valor.

Figura 24 - Tabela de Resistores



No projeto usaremos:

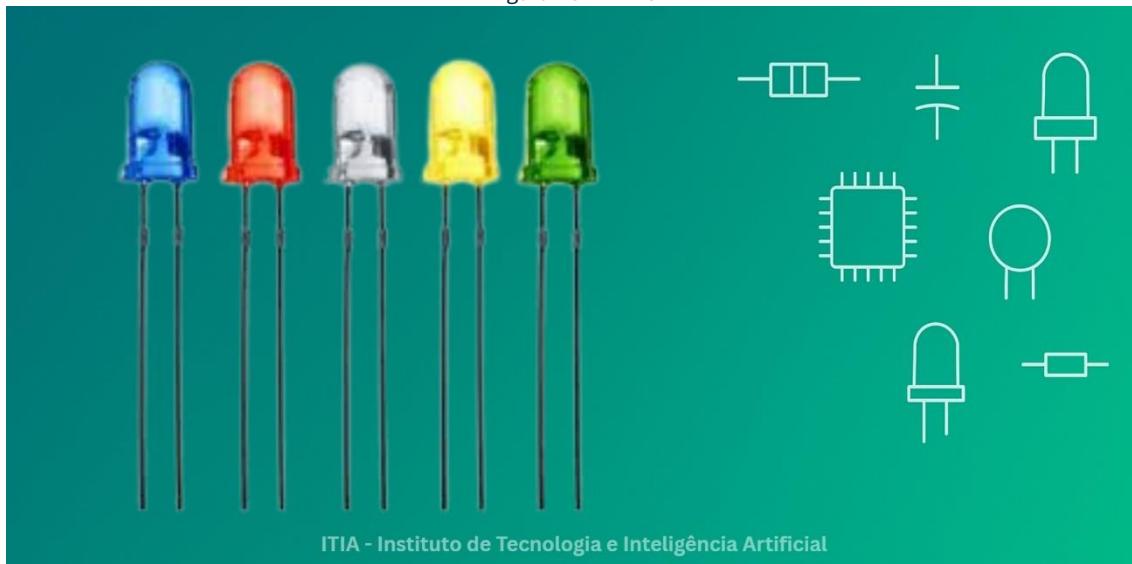
- Resistores de $220\ \Omega$ para cada LED (provavelmente 4 LEDs -> 4 resistores).
- Possivelmente um resistor pull-down para cada botão, se aplicássemos (mas no nosso caso, estamos lendo botões via analogRead de maneira simplificada, então não foi listado resistores de pull-down – comentaremos isso adiante).
- O display LCD com módulo I2C não requer resistores externos (já tem no módulo).
- O potenciômetro é um tipo de resistor variável, falaremos dele em seguida.

3.3. LEDs

LED significa *Light Emitting Diode* ou diodo emissor de luz em português. É um componente eletrônico semicondutor que brilha ao atravessá-lo corrente elétrica (em direção correta).

Essa tecnologia é amplamente utilizada em diversas aplicações, desde iluminação residencial e displays até eletrônicos e sinalização.

Figura 25 - LED's

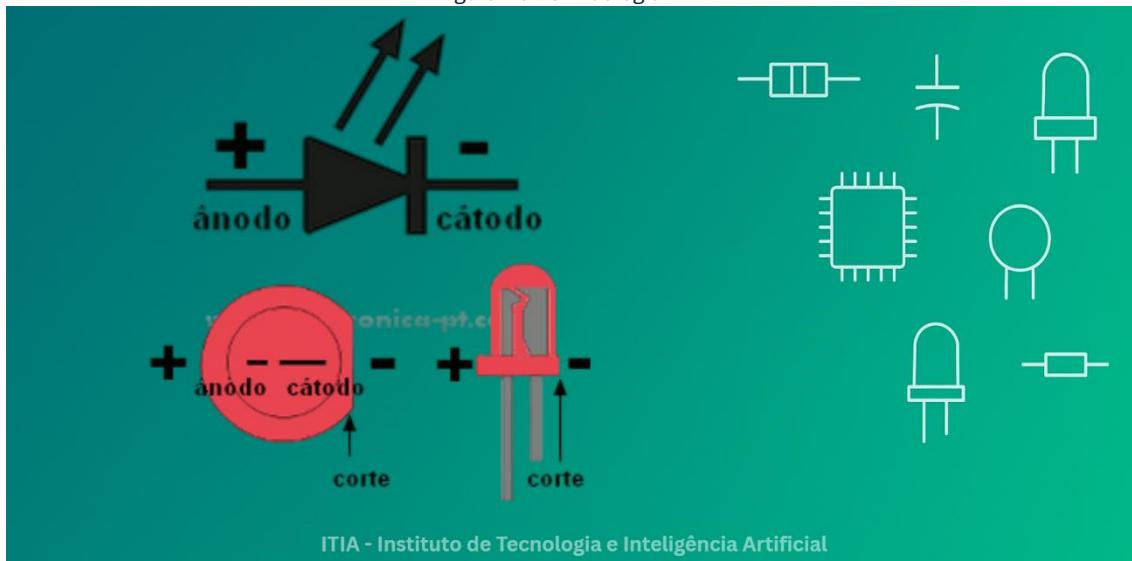


ITIA - Instituto de Tecnologia e Inteligência Artificial

Coisas importantes sobre LEDs:

- Possuem polaridade: um lado positivo (anodo) e um negativo (cátodo). **Devem ser conectados corretamente**, anodo para o lado positivo do circuito (ex: pino do Arduino), cátodo para o negativo (GND), se não, não conduzem (ou seja, se invertido, não acendem). Uma dica: a perna mais **longa** do LED geralmente é o **anodo** (+). A base do LED às vezes tem um chanfro indicando o lado do cátodo (-).
- Precisam de resistor em série para limitar corrente, conforme discutido. Exceção: se ligar no pino 13 do Arduino Uno, note que esse pino já tem um resistor interno ligado ao LED embutido na placa, mas para LED externo ainda precisa, pois o resistor do LED embutido não protege o seu LED, apenas o interno.
- Cores: temos LEDs de várias cores (vermelho, verde, amarelo, azul, etc.). Cada cor tem uma queda de tensão típica: vermelhos ~1.8V, amarelo/verde ~2.0V, azul ~3.0V. Isso é relevante na hora de calcular resistor se quiser ser muito preciso, mas usando 220 Ω serve para todos tranquilamente.
- No projeto, teremos 4 LEDs coloridos representando as “cores do jogo”. Por exemplo, LED **verde**, **vermelho**, **amarelo** e **azul** (as cores clássicas do Genius). Iremos conectá-los a pinos digitais do Arduino como saídas. Quando o Arduino colocar HIGH nesse pino, corrente fluirá pelo LED para o GND e ele acenderá.

Figura 26 - Símbologia LED



Como conectar: Anodo do LED -> pino digital Arduino (via resistor). Cátodo do LED -> GND do Arduino. Pode ser também: pino -> LED -> resistor -> GND, a ordem LED-resistor em série não importa (pode por resistor antes ou depois do LED). Muitos montam pino -> resistor -> LED -> GND, tanto faz, o circuito é em série. Uma dica: use fios de cores semelhantes para ajudar a identificar (ex: fio verde para LED verde, etc.), isso facilita debug.

3.4. Push-Buttons (Botões de pressão)

Os **push-buttons** (botões de pressão) usados no projeto são aqueles pequenos botões táticos de 4 perninhos. Quando você pressiona, ele conecta duas pernas que antes estavam separadas. Os botões táticos geralmente têm duas pernas interconectadas de cada lado (um par de pernas conectado entre si constantemente, e separado do outro par; ao apertar, os dois pares se conectam formando continuidade).

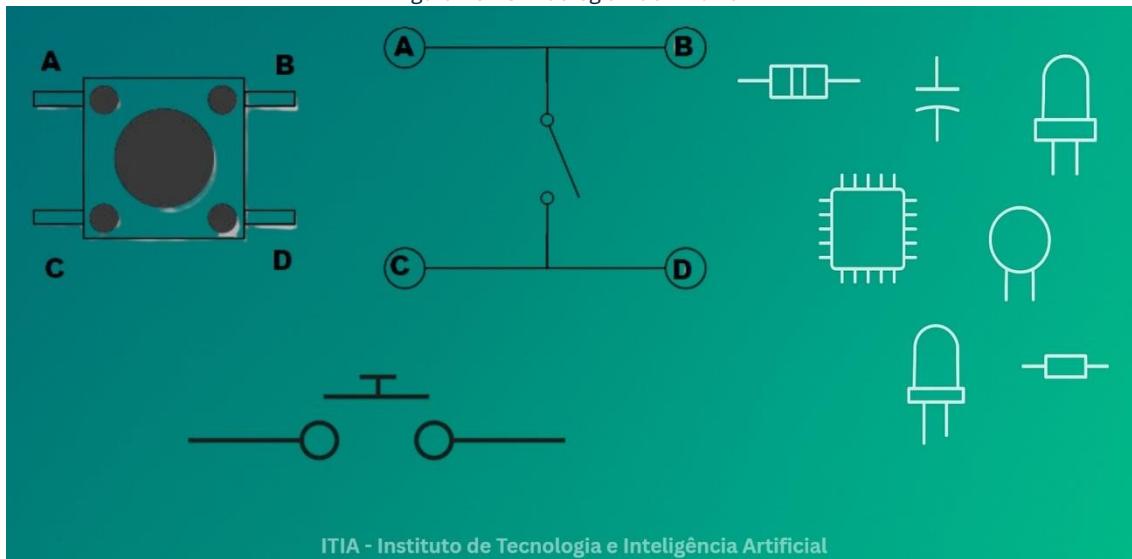
Figura 27 - Push Button



Colocar o botão na protoboard requer atenção: essas 4 pernas devem ser posicionadas de forma que duas fiquem de um lado da fenda central e duas do outro lado, caso contrário ele pode ficar sempre conectado. Regra: coloque o botão **atravessando a fenda central** da protoboard. Assim, cada par de pernas fica em uma metade, isolados. Ao pressionar, eles fecham o circuito entre as metades. (Veremos imagem disso na próxima unidade.)

Como usar botões no Arduino: O Arduino não “detecta pressão” diretamente, ele mede se um pino está recebendo HIGH (5V) ou LOW (0V). Então um jeito comum é: conectar um lado do botão ao 5V, o outro lado ao pino de entrada do Arduino **e também** a um resistor de pull-down para GND. Quando botão não pressionado, o pino está conectado a GND via resistor (LOW). Quando pressionado, o pino fica conectado direto ao 5V (HIGH). O resistor de pull-down ($\sim 10\text{k}\Omega$ geralmente) garante que em repouso o pino não fique “flutuando”.

Figura 28 - Símbologia Push-Button



No nosso projeto, no entanto, usamos outra técnica: ler os botões via entradas analógicas sem resistor, usando a característica do `analogRead`: se o botão conectado a 5V está solto, o pino analógico flutua perto de 0 (nem sempre exatamente 0, mas dificilmente chega a 1023 sem estar pressionado) – e quando pressiona, lê 1023. Assim, o código verifica se `analogRead == 1023` para saber se apertou. Isso simplifica não precisar resistor externo, embora não seja a técnica mais robusta, é aceitável nesse caso educativo. Vale saber: a forma mais robusta seria usar o recurso de **pull-up interno** do Arduino (configurar `pinMode(pino, INPUT_PULLUP)` e ligar botão do pino para GND, invertendo a lógica), mas como vamos no `analog`, vamos manter o método do projeto.

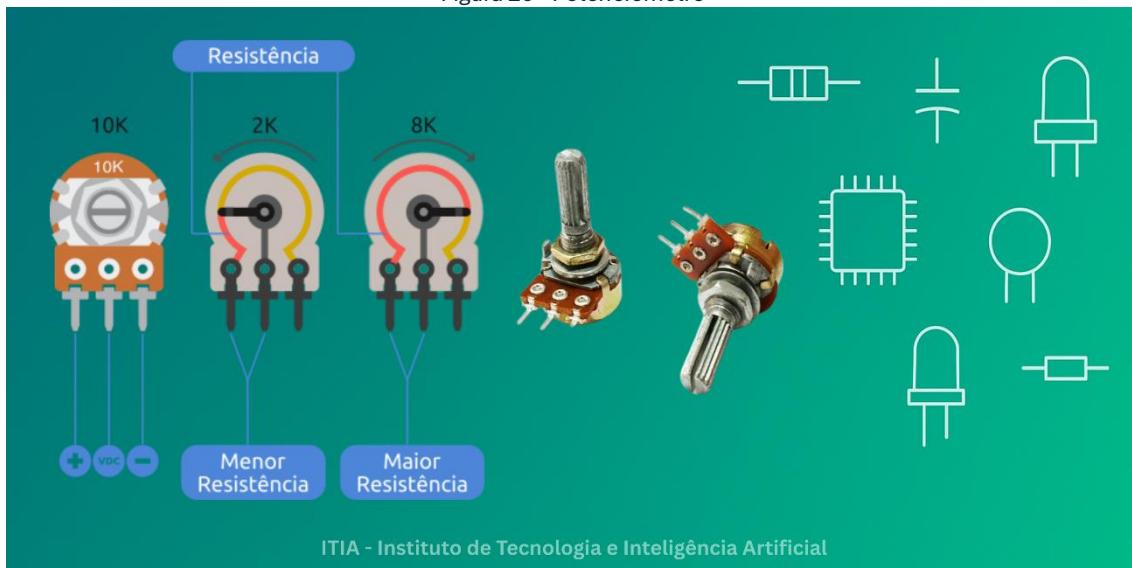
Resumindo a ligação no projeto: cada botão tem um lado ligado no **5V** (pode usar o barramento 5V da protoboard para distribuir para todos), e o outro lado ligado a um pino analógico (A0, A1, A2, A3 por exemplo para 4 botões). Sem apertar, pino flutua (lendo ~ 0 aleatório); apertado, pino é forçado a 5V (leitura 1023). No código, consideraremos leitura 1023 como “apertado”.

Cuidados: Certifique-se que os botões estejam corretamente orientados e que você não tenha conectado 5V direto no pino sem resistor *a menos que* você esteja lendo analogicamente sem configurar como saída (o que é nosso caso). Se acidentalmente configurasse um pino digital como saída LOW e apertasse um botão que liga 5V nele, isso faria um curto (5V direto em GND) – por isso ao usar botões, ou usamos resistores, ou usamos a configuração de entrada para evitar isso. Em nosso projeto, os pinos de botões **são entradas analógicas**, então não há esse risco enquanto não configurarmos erroneamente.

3.5. Potenciômetro

O **potenciômetro** é um resistor variável de três terminais. O modelo comum é um trimpot ou um knob rotativo. Dentro dele tem uma trilha resistiva e um contato móvel. Os terminais das extremidades correspondem às pontas da trilha, e o terminal do meio é o contato móvel (chamado cursor). Assim, entre as extremidades do pot temos a resistência máxima (ex: 10 kΩ), e o terminal do meio “divide” essa resistência em duas partes variáveis conforme a rotação, formando um **divisor de tensão**.

Figura 29 - Potenciômetro



Se conectarmos as extremidades do potenciômetro em 5V e GND, o terminal central fornecerá uma tensão ajustável de 0 a 5V conforme giramos o knob (variando a proporção R_1/R_2). Isso é extremamente útil para entrada analógica: podemos ler A0 e obter um valor que varia de ~0 a ~1023 proporcional à posição. Em muitos projetos, potenciômetro serve para controlar intensidade, volume, etc.

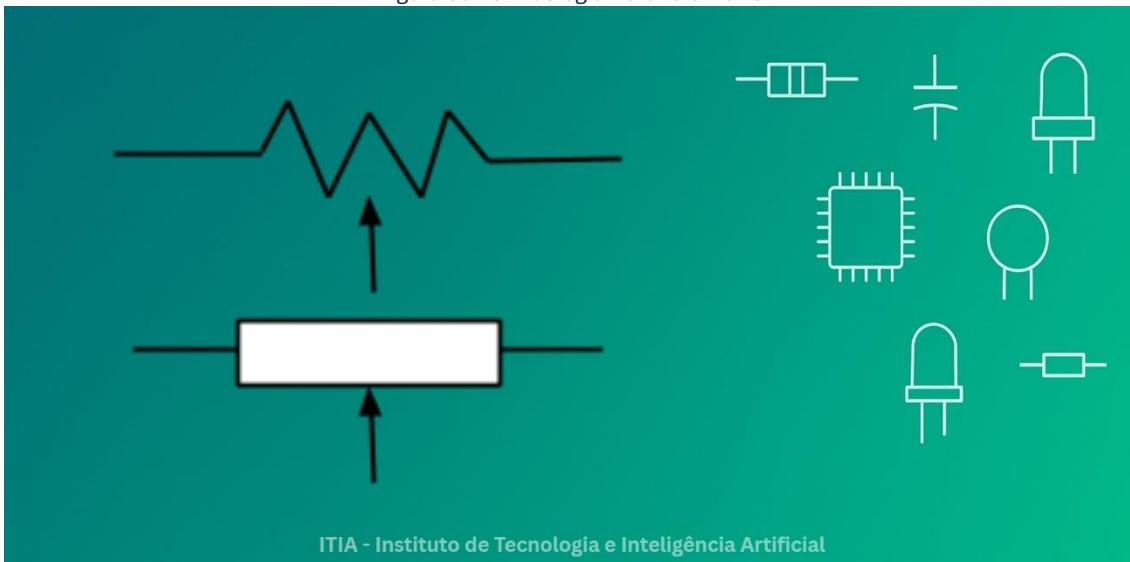
No nosso projeto, o potenciômetro foi listado entre os componentes possivelmente porque:

- **Se estivéssemos usando o LCD em modo tradicional (não I2C),** precisaríamos de um pot para ajustar o contraste do LCD. Com o módulo

I²C, normalmente esse módulo já tem um pequeno trimpot de contraste embutido.

- Ou podemos usar o pot como um componente extra para demonstrar leitura analógica (embora não esteja integrado ao jogo da memória em si).

Figura 30 - Símbologia Potenciômetro



É possível que o eBook inclua o pot apenas para explicar conceito ou para eventual ajuste. Por exemplo, se quiséssemos adicionar uma funcionalidade de **controle de volume** do buzzer (no caso de buzzer passivo, poderíamos variar amplitude – não trivial – ou se fosse um falante com PWM), ou controle de dificuldade (poderia usar pot para definir velocidade do jogo, mas isso não estava no código original). Não há menção de usar o valor do pot no código fornecido, então possivelmente ele era para contraste do LCD se fosse paralelo.

De todo modo, explicando: **Como conectar o potenciômetro:** conecte uma extremidade ao **5V**, a outra ao **GND**, e o terminal central a uma entrada analógica (ex: A6, mas Uno não tem A6; ou algum outro A não usado). Assim você pode ler analogRead desse pino e ver valores de 0 a 1023 conforme gira. Se for para contraste do LCD (no caso de LCD sem I²C), aí conectaria as extremidades em 5V e GND e o meio ao pino VO (contrast) do LCD. Mas como nosso LCD tem I²C, já já falamos dele.

3.6. Display LCD com módulo I²C

O projeto inclui um **display LCD 16x2** (16 colunas, 2 linhas de caracteres) com backlight (luz de fundo). Esses displays alfanuméricicos (geralmente com um fundo azul e texto claro ou fundo verde e texto preto) normalmente têm 16 pinos e usam um controlador chamado HD44780. Controlá-los diretamente requer ~6 pinos do Arduino (4 bits de dados + 2 de controle) mais alimentação e contraste. Para simplificar, utiliza-se um **módulo I²C** acoplado ao LCD. Esse módulo é uma pequena placa que encaixa no LCD e converte as comunicações para um padrão

serial de 2 fios (I2C). Ele normalmente tem um chip expansor (PCF8574 ou similar) e um trimpot para ajuste de contraste.

Figura 31- Display LCD 16x2



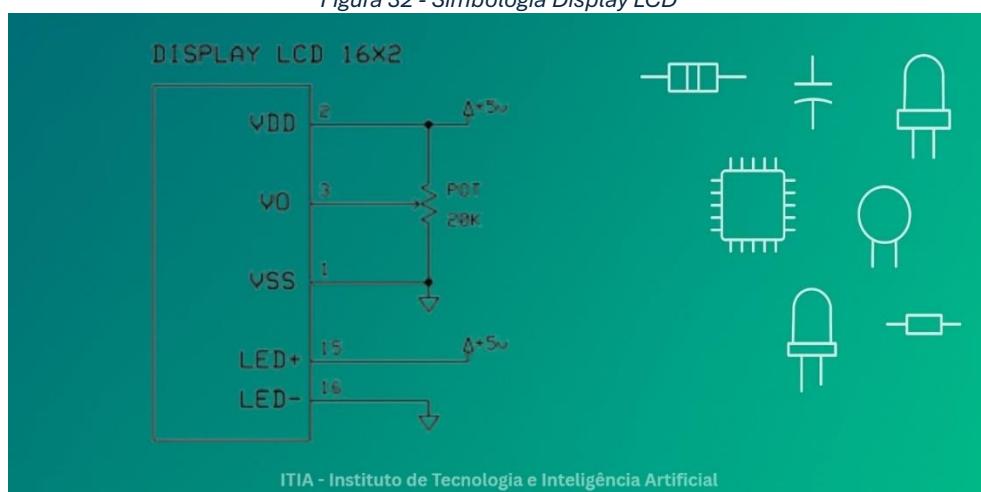
I2C (Inter-Integrated Circuit) é um protocolo serial de dois fios: SDA (dados) e SCL (clock), mais alimentação. O Arduino Uno disponibiliza SDA e SCL nos pinos analógicos A4 e A5 (além de possuírem duplicatas perto do conector USB nos modelos R3). Com I2C, podemos enviar comandos ao LCD usando apenas esses 2 pinos, ao invés de 6.

Ligações do LCD I2C:

- VCC do módulo I2C -> 5V do Arduino
- GND -> GND
- SDA -> A4 (no Uno)
- SCL -> A5

Isso basta para dados e energia. O backlight do LCD e o contraste são controlados no módulo: geralmente o módulo tem um jumper para ligar/desligar backlight e um trimpot azul (aquele parafuso) que ajusta o contraste. Antes de usar, normalmente giramos esse trimpot até aparecer claramente os caracteres no LCD.

Figura 32 - Símbologia Display LCD



No código, para usar o LCD via I2C, usamos uma biblioteca específica (`LiquidCrystal_I2C.h`) e informamos o endereço I2C do módulo (geralmente 0x27 ou 0x3F dependendo do chip) e o tamanho (16x2). No código fornecido no eBook original, eles usaram `LiquidCrystal` (paralela), mas vamos supor que adaptamos para I2C. O importante aqui em eletrônica é: **cione os 4 fios corretamente e ajuste o contraste** se necessário.

Funcionamento: O LCD 16x2 pode mostrar caracteres ASCII simples (letras, números, etc.). Usaremos para exibir mensagens como "Vamos jogar?" e a pontuação, etc. Ele facilita o usuário entender o que está acontecendo, complementando os LEDs e sons.

3.7. Buzzer

O **buzzer** é um dispositivo que emite som quando energizado. Existem dois tipos principais: **ativo** e **passivo**.

- O buzzer **ativo** emite um tom fixo (geralmente um apito agudo ~2kHz) ao ser alimentado com tensão DC. Ou seja, se você ligar 5V nele, ele já faz barulho contínuo.
- O buzzer **passivo** é basicamente um pequeno alto-falante; ele precisa de um sinal alternado (oscilante) para gerar som. Isso significa que se você só mandar HIGH/DC nele, não faz som, mas se você alternar HIGH/LOW rapidamente, ele vibra e produz tom. A função `tone(pino, frequência, duração)` do Arduino justamente alterna o pino na frequência pedida, funcionando perfeitamente com buzzers passivos (ou até com alto-falantes pequenos).

Figura 33 - Buzzer



ITIA - Instituto de Tecnologia e Inteligência Artificial

No material do projeto, foi listado "Buzzer 5V ativo". Porém, o código usa `tone()` com frequências diferentes para cada cor (262 Hz, 294 Hz, 330 Hz, 349 Hz), que correspondem às notas musicais dó, ré, mi, fá, possivelmente. Isso sugere que

nosso buzzer está sendo usado *como se fosse passivo*, pois só um buzzer passivo reproduziria notas diferentes. Se for um buzzer ativo, ele só toca uma frequência (normalmente ~2300 Hz) independentemente do tone (apesar de tone() poderia ligá-lo e desligá-lo rapidamente, mas não vai mudar a frequência do som fundamental dele). Pode ter sido apenas um detalhe do kit: chamaram de ativo mas talvez era passivo. Suponhamos que seja **passivo**, pois assim conseguimos a variedade de sons.

Figura 34 - Simbologia Buzzer



ITIA - Instituto de Tecnologia e Inteligência Artificial

Conexão do buzzer: O buzzer tem dois pinos, geralmente marcados (+) e (-) se for ativo (polarizado), ou nenhum sinal se passivo (mas mesmo passivo às vezes marcam). Conecte o pino positivo no pino digital designado do Arduino (no código, usaram pino 7 para buzzer, chamado buz no define) e o pino negativo do buzzer ao GND do Arduino. Se for passivo, polaridade não importa muito (apesar de às vezes marcam + para indicar convenção interna, mas em passivo tanto faz). Se for ativo, recomenda-se respeitar com + no 5V, mas como vamos usar ele ligado/desligado, também funcionará.

Uso no projeto: Quando o jogador acerta uma sequência ou quando uma cor pisca, tocaremos um tom curto correspondente àquela cor (tipo cada cor tem seu beep). Quando o jogador erra, o código toca um tom diferente prolongado (415 Hz por 3s) para indicar "game over".

Uma observação: a função tone(pin, freq, dur) ocupa internamente o temporizador do Arduino para gerar a onda e não bloqueia o programa (ela retorna imediatamente, tocando no background). No código original, após chamar tone, eles ainda usam delay para esperar o tempo da nota, o que é comum para sincronizar com LED.

3.8. Bateria de 9V

Para que o circuito funcione **sem estar conectado no USB do computador**, vamos usar uma **bateria de 9V** como fonte. A placa Arduino Uno tem um conector DC

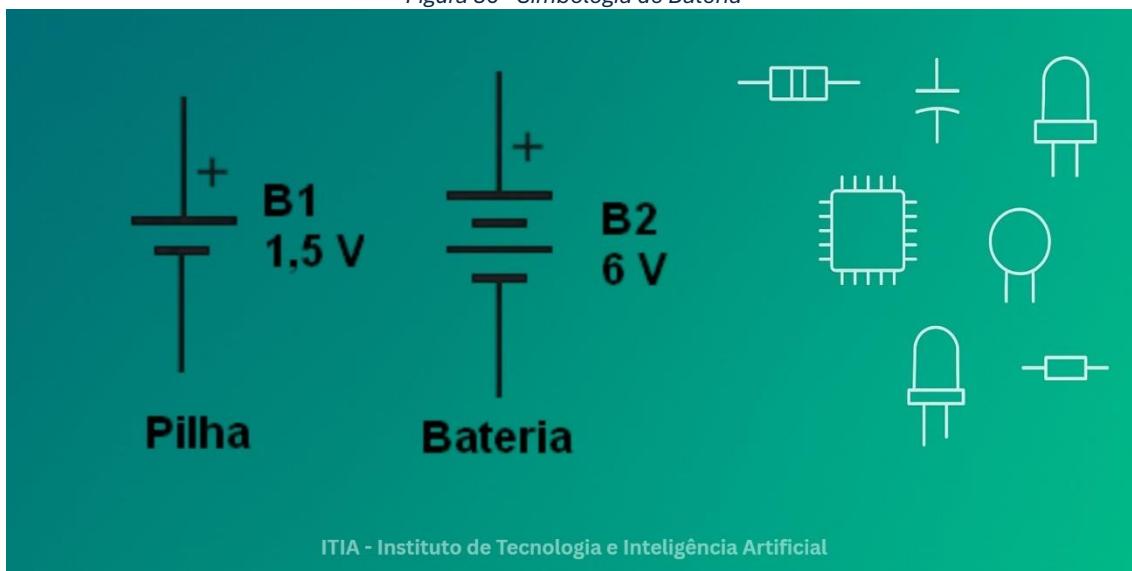
(barril) onde se pode encaixar a bateria via um clip adaptador. Quando fornecemos 9V na entrada do Arduino, a placa possui um regulador linear que reduz para 5V estáveis para a eletrônica. Esse regulador funciona bem para correntes moderadas (até uns 500 mA no máximo, mas nosso projeto consome bem menos). Assim, alimentar o Arduino pelo conector 9V é conveniente. Alternativamente, poderíamos usar 6 pilhas AA (dando ~9V), ou uma powerbank via USB (5V direto). Mas o clássico é a bateria 9V.

Figura 35 - Bateria 9v + conector



Conexão: O clip da bateria 9V tem dois fios – geralmente vermelho (+) e preto (-). No Arduino, se usando o plugue, apenas plugue corretamente. Se fosse pelos pinos, conectaria vermelho ao pino VIN do Arduino e preto ao GND. Mas recomendo usar o plugue jack para evitar erros de polaridade. **Nunca** conecte a bateria 9V direto ao pino 5V – isso pulava o regulador e lançaria 9V na lógica, possivelmente queimando tudo. Use sempre VIN ou o jack.

Figura 36 - Símbologia de Bateria



Duração: Baterias 9V não têm grande capacidade, mas nosso consumo não é enorme. Os elementos principais: o LCD retroiluminado consome ~20mA, cada LED ~10mA quando aceso, buzzer ~<30mA, Arduino ~50mA. Em jogo normal, talvez uns 100 mA em média. Uma bateria alcalina 9V (~500 mAh) duraria algumas horas de jogo contínuo. Se for de demonstração, tudo bem.

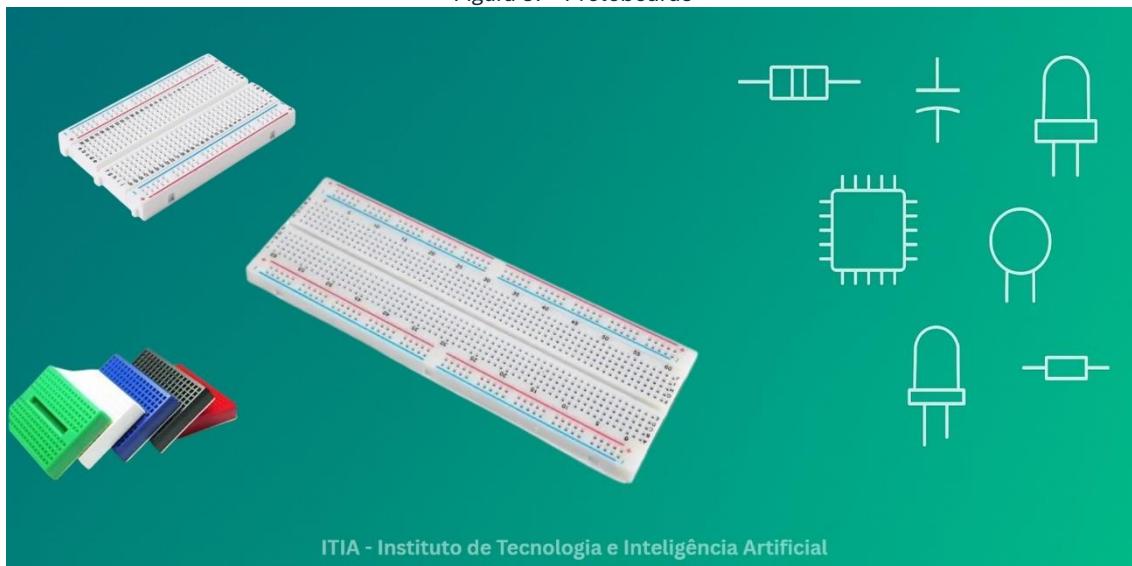
Dica: Desconecte a bateria quando não estiver usando, para preservar carga e para evitar aquecimento do regulador desnecessariamente. O Arduino Uno possui diodo de isolamento entre jack e USB, então se você estiver com USB plugado e bateria, ele irá preferir a fonte mais alta (bateria 9V) e não desperdiça USB. Mas não há necessidade de deixar ambos – no desenvolvimento, use USB; na apresentação autônoma, use a bateria.

3.9. Protoboard (Placa de ensaio)

O **protoboard**, ou **placa de ensaio sem solda**, é onde montaremos nosso circuito de forma organizada e reversível (sem soldar nada). É importante entender **como as conexões internas funcionam**:

- A placa tem **fileiras horizontais** de furos conectados internamente (normalmente grupos de 5 furos consecutivos em uma linha). Costuma ter duas grandes áreas separadas por um canal central (onde encaixamos Cls ou botões).
- Geralmente há **barras de alimentação** nas extremidades (marcadas com + e -), que correm verticalmente, conectando muitos furos em coluna. Nessas barras, você costuma conectar 5V e GND para distribuir facilmente.
- Na porção central de montagem: os furos estão conectados em fileiras de 5 na horizontal. Cada fileira de 5 furos representa um nó do circuito (todos aqueles 5 pontos estarão eletricamente unidos se algo for conectado neles).
- A canaleta central é para separar as duas metades e permitir componentes DIP. No nosso caso, usamos para botões: colocamos duas pernas de cada lado do canal, assim cada lado está isolado até apertar.

Figura 37 - Protoboard



Ao montar:

- **Distribua 5V e GND:** normal é pegar dois fios do Arduino 5V e GND e conectar às tiras de alimentação do protoboard (se disponível). Assim você pode facilmente pegar 5V ou GND em qualquer ponto do board dessas faixas.
- Monte componentes relacionados próximos: ex: LED e seu resistor podem estar próximos, ligados na mesma linha.
- Tente usar uma cor padrão para **GND (preto)** e **5V (vermelho)** nos fios, para evitar confusão.
- Use fios menores se possível, para a montagem ficar “limpa” e fácil de visualizar, especialmente quando trabalhar com crianças, para não virar um ninho de cabos.

No projeto do jogo da memória:

- Podemos reservar um canto da protoboard para os LEDs e resistores. Ex: colocar os 4 LEDs em 4 fileiras separadas, com seus cátodos numa linha comum indo ao GND, e seus anodos cada um com resistor indo para pino do Arduino.
- Os botões podemos colocar em outra área, cada um atravessando a fenda central. Um lado de todos os botões poderia compartilhar uma linha de 5V (todos +5V nas extremidades de um lado). O outro lado de cada botão vai individualmente a um fio para o Arduino (A0, A1, A2, A3).
- O buzzer podemos espertar também na protoboard (ele tem dois terminais como LED). Colocar um terminal na linha GND da protoboard, outro terminal numa linha livre que conectamos com fio a pin 7 do Arduino.
- O potenciômetro, se formos usar, geralmente tem 3 pinos que podem ser inseridos nas linhas do protoboard. Conectamos uma extremidade na faixa +5V, outra na faixa GND, e o meio com um fio a Arduino (por ex. A6 ou A0 se livre). Se é só para contraste do LCD, no caso do módulo I2C não precisa externo.

Quando formos montar de fato na Unidade 04, daremos as instruções precisas, mas esse pano de fundo ajuda.

3.10. Lei de Ohm e Divisor de Tensão (revisão rápida com aplicação)

Já explicamos, mas vamos amarrar os conceitos:

- **Lei de Ohm:** $U = R * I$. **Aplicação prática:** calcular resistor para LED. Suponha LED verde (2V) deseja 10 mA com fonte 5V. $R = (5V - 2V) / 0,01A = 300 \Omega$. Usamos 220 Ω para ~14 mA ou 330 Ω para ~9 mA.
- **Divisor de tensão:** dois resistores em série. **Aplicação prática:** Potenciômetro e leitura analógica – O pot de 10k entre 5V e GND fornece no meio um divisor ajustável. Se o cursor divide 70% e 30% por exemplo, teremos ~3.5V no meio (70% de 5V). Assim analogRead daria $\sim (3.5/5)*1023 \approx 716$. Divisor de tensão às vezes também é usado para sensores (ex: LDR com resistor fixo). Em nosso projeto específico, não implementamos

nenhum sensor de analog (exceto o pot se usar). Mas, por curiosidade, se quiséssemos ler um sensor de 9V, poderíamos usar um divisor para trazê-lo a 5V – mas isso é avançado e não presente aqui.

3.11. Dimensionamento Correto de Componentes

Essa seção resume as escolhas que fizemos e por quê:

- **Resistores dos LEDs:** dimensionados para proteger os LEDs e não sobrecarregar o Arduino. Escolhemos $220\ \Omega$, que limita a $\sim 10\text{-}15\ \text{mA}$ por LED, dentro do seguro ($20\ \text{mA}$ máx. por pino, $200\ \text{mA}$ total). Com 4 LEDs possivelmente acesos no máximo simultaneamente, isso daria $\sim 60\ \text{mA}$ se todos acesos, ok.
- **Botões:** se fôssemos usar pull-down resistors, seriam valores altos ($10k$) para não drenar muita corrente quando botão aperta (porque quando apertado, $5V$ iria direto ao pino e ao resistor para GND; com $10k$, corrente = $5/10000=0.5\ \text{mA}$ insignificante). No nosso design sem resistor, usamos flutuação – dimensionamento aqui foi “nenhum resistor” contando que analog input consome quase nada. Talvez fosse refinado com um $100k\Omega$ se quisesse.
- **Potenciômetro:** $10\ k\Omega$ é um valor comum para entrada analógica e contraste LCD. Um pot muito baixo (ex $1k$) consumiria corrente desnecessária ($5V/1k = 5\ \text{mA}$ contínuos só jogados fora), um muito alto (ex $1M$) faria a leitura instável. $10k$ é padrão e consome $0.5\ \text{mA}$ entre $5V$ e GND, razoável.
- **Buzzer:** não precisa resistor, pois ele é um dispositivo ativo de certa impedância (um buzzer passivo tem resistência interna da bobina uns 16Ω ou mais, mas a função tone alterna sinal evitando DC contínua alta). De qualquer forma, o pino do Arduino suporta tranquilamente o buzzer. Se fosse um alto-falante 8Ω grande, aí sim precisaria cautela ou driver.
- **Display LCD:** O backlight tem internamente um resistor ou arranjo que limita corrente (geralmente $\sim 20\text{mA}$). O contraste pot se ajusta conforme necessário. O módulo I²C cuida de que as saídas PCF8574 não excedam corrente. Dimensionamento aqui foi usar as libs e circuito padrão, sem alterações.

Em resumo, **todos os componentes do projeto estão sendo usados dentro de suas faixas ideais:**

- LED: corrente segura e brilho ok,
- Arduino pinos: corrente somada OK (pior caso: 4 LEDs $\sim 60\ \text{mA}$ + buzzer $\sim 30\ \text{mA}$ + backlight LCD $20\ \text{mA} = \sim 110\ \text{mA}$ total da placa, está no limite do USB mas o Arduino aguenta isso, e o regulador $9V\rightarrow 5V$ dissipará $(9-5)*0.11=0.44W$, um pouco quentinho mas deve suportar).
- Bateria 9V: conseguirá fornecer esses $110\ \text{mA}$ (a maioria das 9V alcalinas podem $\sim 500\ \text{mA}$ se curto, mas claro drenando $0.11A$ ela durará poucas horas).
- Nenhum componente sem proteção (usamos resistores adequadamente, exceto botões confiando no analog).

- Fiação: Use fios adequados, de preferência não muito finos para não quebrar, mas para 100 mA qualquer jumper serve.

Com todo esse conhecimento de eletrônica básica, você já tem condições de compreender o **porquê** de cada fio e peça quando formos montar o circuito. Não seremos meros “montadores por receita”, mas sim **makers conscientes** do que está acontecendo no projeto! Isso torna a atividade muito mais educativa.

No próximo capítulo (Unidade 04), finalmente montaremos o circuito completo passo a passo. Tenha em mãos todos os componentes listados e monte junto. Lembre-se de desconectar a alimentação (USB ou bateria) sempre que for alterar conexões. Segurança e cuidado são fundamentais. Vejo você na Unidade 04, com a protoboard a postos!

Exercícios – Unidade 03

1. **Lei de Ohm:** Qual a fórmula da Lei de Ohm e o que cada letra representa? Use a fórmula para calcular que corrente passaria num resistor de $100\ \Omega$ conectado em 5V.

2. **Resistor no LED:** Por que devemos colocar um resistor em série com um LED ao ligá-lo no Arduino? O que poderia acontecer se não colocarmos?

3. **Cálculo de Resistor:** Temos um LED azul (queda $\sim 3V$) e queremos limitar a 15 mA usando 5V. Que valor aproximado de resistor devemos usar? (Use $U=R*I$ com $U = 5-3 = 2V$, $I=0,015\ A$). Mostre o cálculo.

4. **Divisor de Tensão:** Explique com suas palavras o que é um divisor de tensão. Se eu tenho dois resistores iguais em série em 10V, qual será a tensão no ponto entre eles?

5. **Protoboard:** Como são conectados internamente os furos de um protoboard? O que acontece se você colocar as duas perninha de um botão na mesma linha do protoboard?

6. **Identificando Componentes:** No seu kit, identifique um resistor de $220\ \Omega$ (dica: veja as listras) e um de $10\ k\Omega$. Quais cores eles têm? (Opcionalmente, use um multímetro se disponível para medir e confirmar).

7. **Polaridade:** Cite dois componentes do projeto que possuem polaridade definida (ou seja, têm lado positivo e negativo). Como podemos identificar o terminal positivo neles?

8. **Botão (Pulldown/Pullup):** No projeto, optamos por ler o botão via analogRead sem resistor externo. Em uma montagem clássica digital, qual seria o papel de um resistor de $10\text{k}\Omega$ ligado ao botão? Explique a diferença entre **pull-down** e **pull-up** nesse contexto.

9. **Potenciômetro:** O potenciômetro tem 3 terminais. Se ligarmos as extremidades em 5V e GND, o que acontece com a tensão no terminal central ao girar o eixo? Para que poderíamos usar isso em um circuito?

10. **Dimensão de Corrente:** Suponha que no nosso circuito todos os LEDs estejam acesos (4 LEDs com 10 mA cada) e o buzzer tocando consumindo 30 mA, além do LCD consumindo 20 mA. Qual seria a corrente total aproximada? Isso está dentro do limite de 200 mA da placa?

UNIDADE 04: Montagem do Projeto

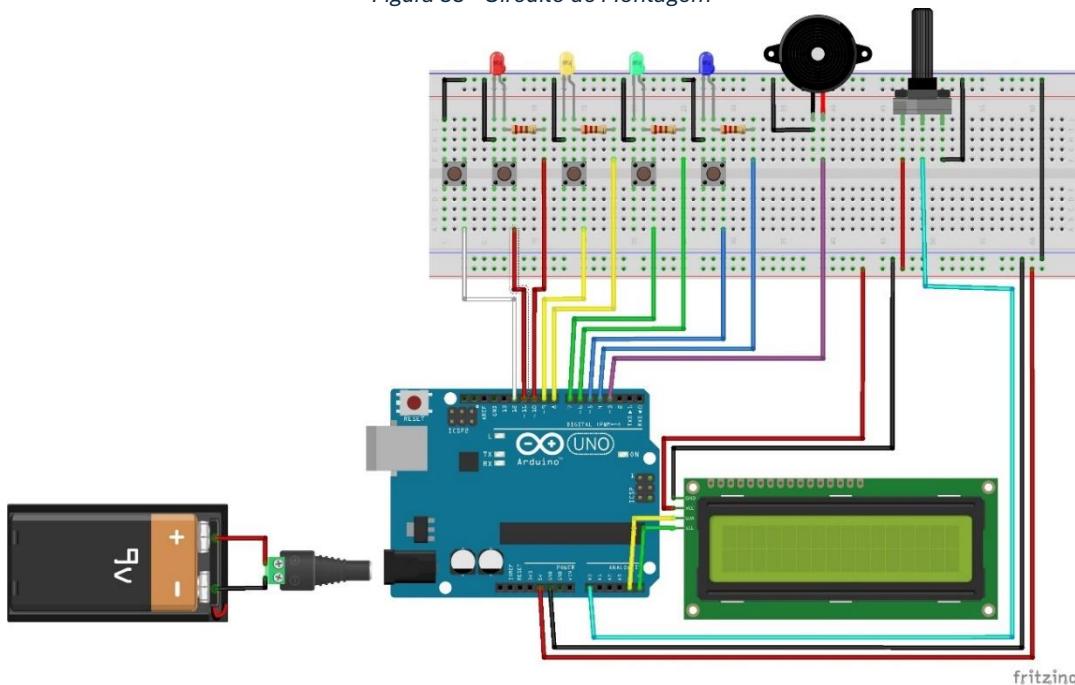
Chegou a hora de **colocar em prática** tudo o que aprendemos e montar o circuito completo do **Jogo da Memória com Arduino**! Nesta unidade, faremos um passo a passo detalhado da montagem na protoboard, incluindo dicas de organização, segurança e verificação para garantir que tudo funcione corretamente. Também vamos analisar a imagem do circuito montado para entender visualmente as conexões. Lembre-se: **sempre desligue a alimentação** do Arduino (desconecte o cabo USB e/ou bateria) enquanto estiver montando ou alterando conexões no circuito, para evitar curtos ou danos durante a montagem.

Tenha em mãos:

- **Protoboard** (placa de ensaio)
 - **Arduino Uno** (ou equivalente)
 - **Cabos jumper** (fios de conexão)
 - **4 LEDs** (vermelho, verde, amarelo, azul)
 - **4 resistores de $220\ \Omega$** (um para cada LED)
 - **4 botões (push-buttons)** tátteis
 - **1 buzzer** (passivo de preferência)
 - **1 display LCD 16x2 com módulo I2C**
 - **1 Potenciômetro $10k\Omega$** (se for usar, possivelmente para contraste LCD)
 - **Bateria 9V** com conector (não ligue ainda)
(E um computador com cabo USB para testar depois de montar, mas isso mais à frente)

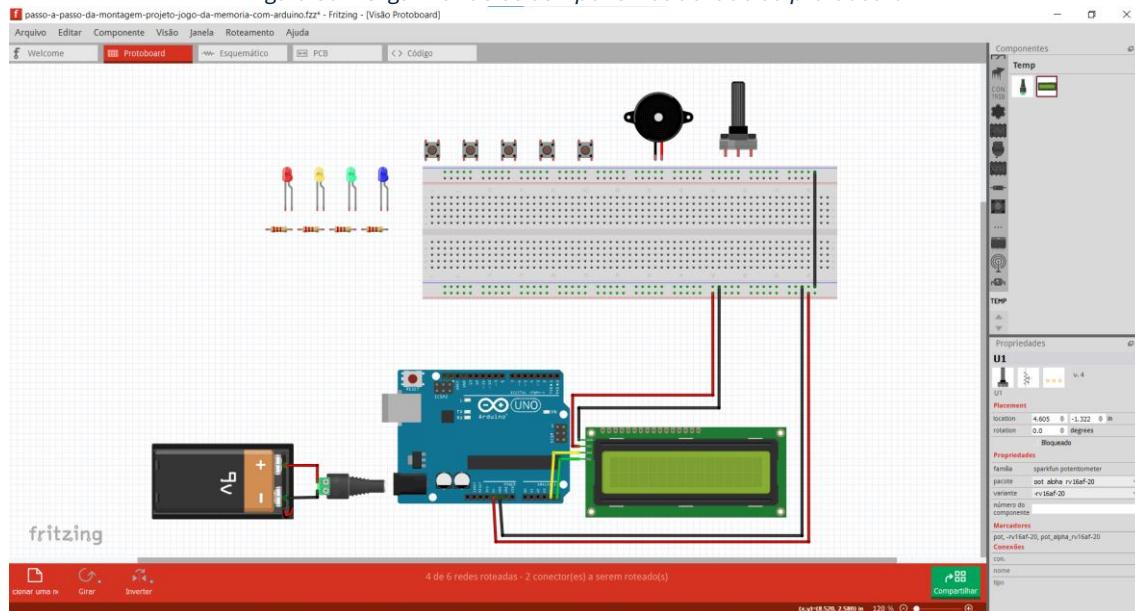
Vamos montar por etapas lógicas: primeiro os componentes de saída (LEDs, buzzer, LCD), depois os de entrada (botões, pot), cuidando de ligar todos os GNDs e 5V adequadamente.

Figura 38 - Circuito de Montagem



4.1 Preparando a Protoboard e Alimentação

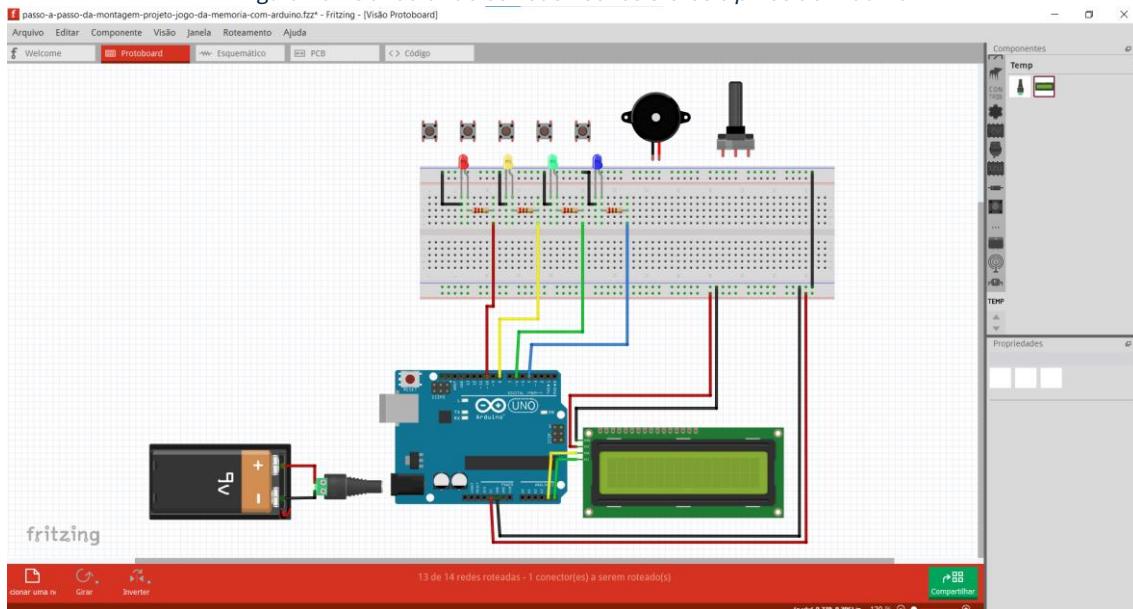
Figura 39 - Organizando os componentes ao lado da protoboard



- 1. Conecte as barras de alimentação:** Se sua protoboard tem colunas marcadas com + e - nas laterais, utilize-as para distribuir 5V e GND. Pegue um jumper **vermelho** e ligue o 5V do Arduino (pino 5V) à coluna “+” da protoboard. Depois, pegue um jumper **preto** e ligue um GND do Arduino à coluna “-” da protoboard. Agora toda a extensão dessas barras terá 5V e GND disponíveis. (Em protoboards grandes, as barras podem ser separadas na metade; verifique se precisa conectar as metades das colunas – algumas vêm já contínuas, outras têm ruptura central.)
- 2. Posicione o Arduino:** Pode deixar o Arduino solto ao lado da protoboard ou fixá-lo se tiver base. O importante é que os fios consigam alcançá-lo. Muitos optam por deixar a protoboard próxima ao Arduino e usar jumpers curtos.
- 3. Plano de montagem:** Reserve um setor da protoboard para os **LEDs e resistores**. Podemos colocar, por exemplo, os 4 LEDs e resistores no lado esquerdo do board, cada um em uma fileira diferente. Reserve outro setor para os **botões** – podemos alinhá-los lado a lado no centro do board (atravessando a canaleta). O buzzer e pot podem ficar na direita. E não se esqueça do **display LCD**: se seu módulo I2C tem pinos, você pode espalhá-lo nas tiras do topo da protoboard, ou conectar via jumpers diretamente ao Arduino.

4.2 Montando os LEDs com resistores

Figura 40 - Conectando os Leds nos resistores e pinos do Arduino



1. Inserir LEDs:

Pegue o **LED Verde**. Identifique sua perna longa (anodo +) e perna curta (cátodo -). Coloque-o na protoboard: por exemplo, perna longa em um furo da coluna 6 e perna curta em um furo da coluna 7 (só um exemplo – o importante é que as pernas fiquem **em linhas diferentes**, não na mesma linha, senão estariam em curto). Separe-os por uns 2-3 furos de distância para dar espaço. Faça o mesmo para os outros LEDs (Vermelho, Amarelo, Azul), cada um em uma linha horizontal diferente, mas mantendo a mesma orientação (ex: anodos à esquerda e cátodos à direita, ou vice-versa). Deixe um pequeno espaçamento entre eles para não confundir.

2. Conectar resistores aos LEDs:

Para cada LED, conecte um resistor de **220 Ω** na perna **cátodo** (a negativa). Como? Insira uma extremidade do resistor no MESMO conjunto de 5 furos onde está a perna curta do LED, e a outra extremidade em uma linha livre que podemos conectar ao GND. Uma abordagem é: coloque todas as pernas de resistores livres em uma **mesma linha horizontal** da protoboard e conecte essa linha ao GND comum. Alternativamente, cada resistor pode ir direto à barra de GND. Vamos supor o seguinte: coloquei os 4 resistores alinhados verticalmente de modo que todos os seus terminais inferiores vão para a barra negativa (-) do protoboard. Então:

- LED verde cátodo -> resistor -> GND
- LED vermelho cátodo -> resistor -> GND
(e assim por diante).

O outro terminal de cada resistor precisa estar numa linha que

tenha acesso ao GND. Se a sua protoboard tem a barra -, pode usar um fio pequeno do resistor até a barra -, ou se conseguir, enfie direto na coluna da barra - (às vezes o resistor alcança). Verifique que agora todos os cátodos dos LEDs estão conectados ao GND (através dos resistores).

3. Ligar anodos dos LEDs aos pinos Arduino:

Agora precisamos ligar cada perna longa (anodo) de LED ao seu respectivo pino digital no Arduino, conforme definiremos:

- o LED Vermelho-> digamos que conectaremos no pino 10 do Arduino (como no código original #define ledG 10).
- o LED Amarelo -> pino 8 do Arduino (ledR 8).
- o LED Verde -> pino 6 do Arduino (ledY 6).
- o LED Azul -> pino 4 do Arduino (ledB 4).

Use jumpers de cores correspondentes se possível: fio verde do anodo do LED verde até o pino 8 no header do Arduino; fio vermelho do LED vermelho até pino 9; amarelo -> 10; azul -> 13. Conecte-os um a um, certifique-se de colocar na mesma linha da perna do LED (por exemplo, se o LED verde anodo está na linha 10, coloque o jumper pegando um furo nessa linha 10 e levando ao pino8 do Arduino).

Dica: Organize os fios de modo que não fiquem muito cruzados. Em todo caso, revise: **nenhum LED anodo deve tocar 5V direto** – eles vão aos pinos. **Nenhum cátodo vai direto ao GND** – passam pelo resistor. E **nenhum LED sem resistor**.

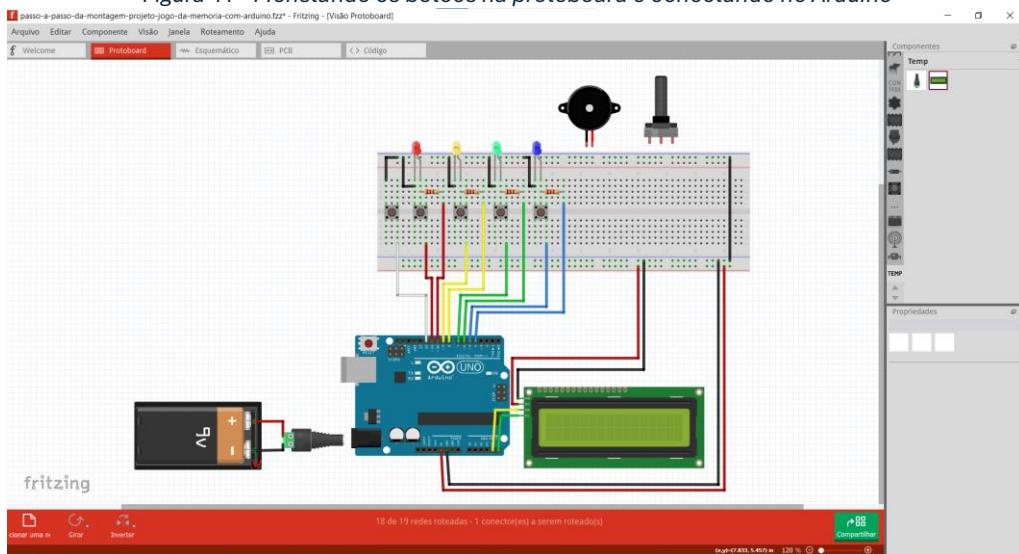
4. Verificação LED:

Até aqui, sem energia ligada, você pode testar continuidade se tiver um multímetro: do pino 10 do Arduino até GND do Arduino deve dar cerca de 220Ω (caminho: pino10-> LED vermelho -> resistor -> GND). Similar para outros pinos (8,6,4). Se não tiver multímetro, revise visualmente: cada LED anodo vai a um pino, cada LED cátodo vai a um resistor e daí ao GND comum. Quando ligarmos e programarmos, se um LED não acender, provavelmente é inversão ou mau contato – mas com essa montagem deve funcionar.

4.3 Montando os Botões

Tenha muita atenção na montagem dos botões, pois eles tem 04 pinos e como estes serão dispostos na protoboard mudará completamente a funcionalidade do componente, deixando de ser um botão e podendo a ser usado como um jumper de circuito.

Figura 41 - Monstando os botões na protoboard e conectando no Arduino



1. Posicionar Botões:

Pegue os **push-buttons** e insira-os na protoboard atravessando a fenda central. Coloque os 5 botões lado a lado na mesma fileira talvez, ou em colunas adjacentes para organização. Por exemplo, posicione o primeiro botão (botão do início do código) de modo que duas pernas fiquem em colunas 01 e 03 (apenas ilustrativo) do lado esquerdo e as outras duas em 01 e 03 do lado direito (nas linhas do meio a, digamos, linha e-f se marcar colunas). O importante: as perninhas do botão devem ficar **duas de um lado, duas do outro** do canal central. Assim, quando NÃO pressionado, o lado esquerdo não conecta com o direito. Quando pressionado, o botão internamente conecta os dois lados.

Faça o mesmo para os outros 4 botões, podendo espaçar por algumas colunas para evitar confusão de fios. Por exemplo, botão do led vermelho nas colunas 6-8, botão do led amarelo nas col 12-14, botão do led verde nas 18-20, botão do led azul nas 26-28, cada um atravessando a fenda.

2. Ligar um lado dos botões ao 0V (GND):

Escolha um lado comum de todos os botões para ligar ao negativo. Digamos que decidimos que o lado esquerdo de cada botão (duas pernas interligadas daquele lado) será o lado do 0V. Podemos simplificar unindo esse lado esquerdo de todos os botões numa mesma linha de 0V: por exemplo, conecte cada botão com um jumper para a barra 0V do protoboard. Alternativamente, se eles estão próximos, você pode usar um único fio que passe por todos via a protoboard (mas geralmente mais fácil: fio de cada botão ao +).

Recomendo: insira um jumper do *lado esquerdo* do botão do start até a barra (-) protoboard. Repita para botão dos leds vermelho, amarelo, verde e azul – cada um terá seu lado esquerdo atado ao 0V. Agora, isso significa que quando qualquer botão for pressionado, ele vai levar 0V para o lado direito daquele botão.

3. Ligar o outro lado dos botões aos pinos Arduino:

O lado direito de cada botão (que fica desconectado até apertar, momento em que se conecta ao 0V do outro lado) deve ir a entradas digitais do Arduino para leitura. Conforme o código:

- Botão Reset -> Pino 12 (pushReset 12)
- Botão Vermelho -> Pino 11 (pushG 11)
- Botão Amarelo -> Pino 9 (pushR 9)
- Botão Verde -> Pino 7 (pushY 7)
- Botão Azul -> Pino 5 (pushB 5)

Use fios (preferencialmente da cor correspondente ao LED ou do botão se você marcou de algum jeito):

Conecte um jumper do *lado direito* do botão reset até o pino 12 do Arduino. Depois, botão vermelho lado direito -> 11, amarelo -> 9, verde -> 7 e azul -> 5.

Certifique-se que você pega a linha correta: lembra que cada botão lado direito terá duas perninhas metal ligadas – tanto faz qual dos dois furinhos você insere o jumper, pois estão conectados entre si. Pode até aproveitar e conectar o buzzer ou pot se proximidades, mas não confunda.

4. Opcional:

Resistores pull-down? No nosso projeto optamos não usar, confiando na função de INPUT_PULLUP ao definir o botão como pino de entrada do arduino. Caso quisesse estabilidade, poderia conectar um resistor de $\sim 100\text{k}\Omega$ de cada entrada 12,11,9,7,5 para +5V, para garantir leitura 1 quando solto. Não foi listado, então não colocaremos. Mas se por curiosidade quiser, colocaria um resistor extra da linha do botão lado direito (que já vai pro pino 12) até o +5V barra. Isso garantiria 5 V quando aberto. Com 100k a leitura pode ainda flutuar um pouquinho, mas é bem fraca. Enfim, manteremos sem resistor conforme instruções do projeto original.

5. Verificação Botões:

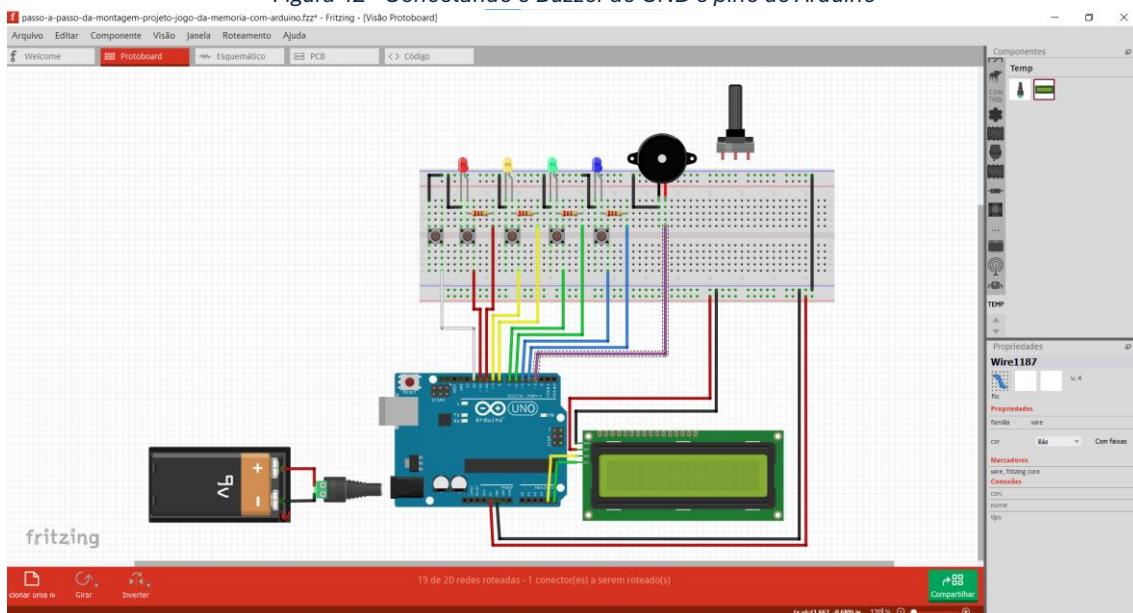
Com tudo desligado, se você tiver um multímetro, pode medir entre o pino 12 e GND: não pressionado deve estar desconectado (resistência mega-ohm, flutuante); pressionado deve mostrar $\sim 0\Omega$ (porque A0 se conectou via botão a 0V e se o Arduino não alimenta, você mediria circuito meio flutuante – melhor seria medir continuidade do pino 12 ao GND: ao apertar, 12 está contíguo ao +0V temporariamente). Sem instrumentação: revise visualmente – cada botão: um lado vai a 0V, outro vai a um fio para Arduino. Não há conexões extras erroneamente entre botões ou ao GND.

Importante: **Certifique-se que os botões estejam orientados corretamente.**

Uma forma de testar manual: desconecte Arduino do PC (sem energia), use um LED/resistor teste ou multímetro para ver se, ao apertar, o 0V barra aparece no fio do pino. Se sim, tá ok.

4.4 Conectando o Buzzer

Figura 42 - Conectando o Buzzer ao GND e pino do Arduino



1. Posicionar o Buzzer:

Insira o buzzer na protoboard. Ele tem dois terminais. Se houver um + marcado, coloque-o de preferência em direção ao lado positivo da protoboard (só para lembrar). Qualquer espacinho serve – pode ser nas extremidades ou entre os botões, desde que não toque outros circuitos indevidamente. Por exemplo, coloque o buzzer em duas linhas livres próximas ao final do board.

2. Conectar pino do buzzer ao Arduino:

Pegue o terminal positivo (+) do buzzer e conecte um jumper dele para o pino 3 do Arduino (que definimos como buz 3). O outro terminal do buzzer (negativo -) deve ser conectado ao GND da protoboard. Provavelmente a posição do buzzer permite você enfiar o terminal negativo diretamente na coluna de GND (-) do protoboard, o que já o conecta ao GND comum. Se não, use um fio do buzzer negativo para GND barra.

3. Verificação Buzzer:

Buzzer + -> pino3, buzzer - -> GND. Simples. Se invertido (pino3 no - e + no GND) ele não vai funcionar porque para um buzzer ativo polaridade invertida = silencioso. Em passivo tanto faz, mas mantenha padrão.

4.5 Conectando o Display LCD (com I2C)

Já tínhamos previamente conectado o display, mas pra reforçar iremos detalhar o passo a passo, e nesse passo o circuito estaria conforme imagem anterior.

1. Posicionar o LCD:

Se o seu LCD já está acoplado ao módulo I2C, ele terá 4 pinos: VCC, GND, SDA, SCL. Você pode tanto conectar esses 4 pinos com jumpers diretamente aos do Arduino, ou encaixar o módulo em algum canto da protoboard e usar as barras. Muitas vezes, porém, o LCD não cabe nas protoboard pequenas com todos pinos, então é comum usar jumpers M/F para ligar direto no Arduino.

Vamos supor que conectaremos via jumpers por clareza:

2. Ligar alimentação do LCD:

Conecte o pino VCC do módulo I2C do LCD ao 5V (barra + ou direto no 5V do Arduino – mas como já temos barra + usada, pode usar ela). Conecte o pino GND do módulo ao GND comum (barra -). Isso alimenta o display.

3. Ligar SDA e SCL:

Conecte SDA do módulo ao **A4** do Arduino (no Uno, A4 é SDA). Conecte SCL do módulo ao **A5** do Arduino (A5 é SCL). Use jumpers curtos e de cores diferentes (por ex. SDA verde, SCL azul). Se seu Arduino Uno tem os pinos SDA/SCL perto do USB, poderia usar eles também – mas internamente são os mesmos A4/A5.

4. Ajustar contraste:

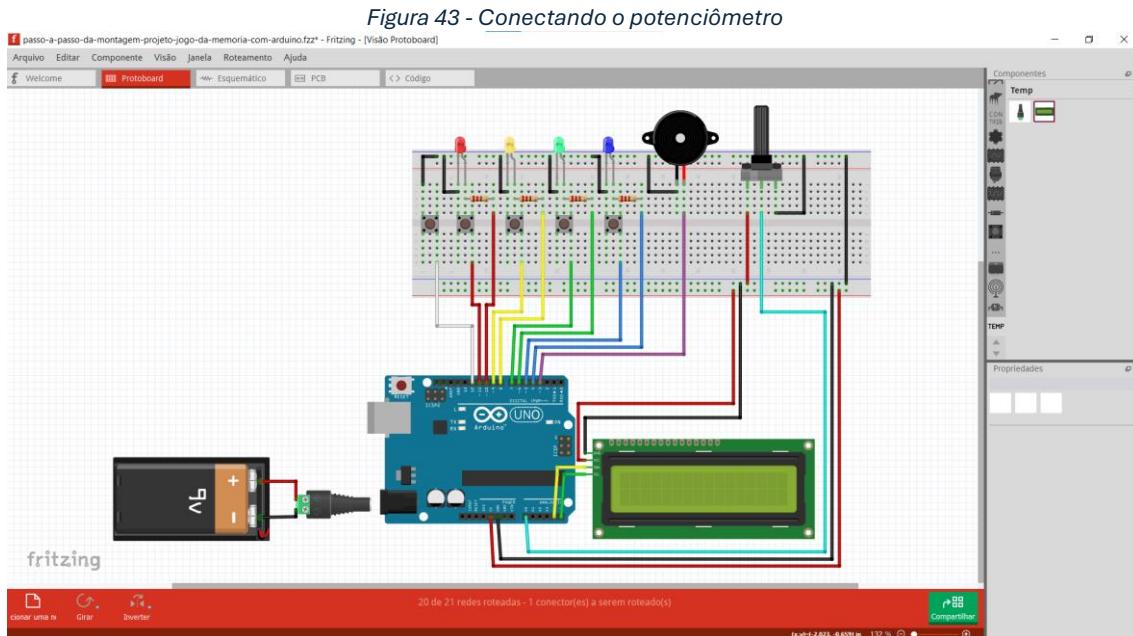
O módulo I2C tem um pequeno potenciômetro de ajuste (um trimpot de parafuso). Quando energizarmos, se não aparecer nada no LCD (nem aqueles blocos pretos), possivelmente o contraste está no extremo. Com uma pequena chave de fenda, gire o potenciômetro até começar a ver os caracteres. Faça isso *depois* quando testarmos o código, mas lembre-se disso.

5. Verificação LCD:

VCC->5V, GND->GND, SDA->A4, SCL->A5. Reforçando: sem esses 4, o LCD não funciona. Polos ok? sim. Evite confundir SDA e SCL se inverter, não comunica (mas não queima). Cheque labels no módulo ou manual se duvida.

Não se preocupe, a tela não vai mostrar nada até programarmos e ligarmos mas anote que no final de montar e antes de rodar, vale conferir se nenhum fio SDA/SCL soltou ou se GND do LCD está realmente no comum.

4.6 Conectando o Potenciômetro:



1. Posicionar o Pot:

Coloque o potenciômetro no protoboard – geralmente os potenciômetros de 3 pinos se encaixam nas fileiras com as pernas cada uma em linhas diferentes (alguns conseguem ficar atravessando a fenda central também). Conecte as pernas extremas do pot: uma na barra +5V, outra na barra GND. Ex: se pot pernas estão na colunas 1,2,3: ligue perna1 -> +, perna3 -> -.

2. Ligar pino central do pot ao Arduino:

Conecte a saída do pot (terminal central) a uma entrada analógica livre. Como já usamos os pinos A4/A5 com o I2C do display. Então não temos entradas livres no Uno DIP. Poderíamos usar A0-A3 pra pot, logo escolhi usar o pino A0.

4.7 Conectando a Alimentação Externa (Bateria 9V)

Atenção: não conecte a bateria até verificartermos todo o circuito e preferencialmente até termos carregado o código e testado via USB. O Arduino pode ser alimentado pelo USB do computador por enquanto, o que facilita testes, e só depois usaremos a bateria para rodar de forma autônoma.

Mas vamos planejar:

- O clip da bateria 9V tem um plugue que vai ao Arduino. Se você tem esse adaptador de bateria com plug P4, aguarde para plugar no jack do Arduino **apenas no final** quando formos testar sem PC.
- Se fosse ligar via VIN/GND: vermelho do clip -> VIN pin, preto -> GND pin do Arduino.
- **Nunca** ponha o fio da bateria nos 5V/GND diretamente.

- Antes de usar a bateria, assegure-se de desconectar USB (o Arduino tem diodo ORing então até pode deixar os dois, mas a bateria vai ficar drenada sem necessidade se USB tá alimentando).
- A bateria 9V, se nova, deve ler ~9.5V. Se for recarregável NiMh 8.4V ou 9.6V valem também.

4.8 Análise da Imagem do Circuito Montado

Vamos observar a imagem a seguir do circuito montado para conferir se tudo está de acordo:

Figura: Montagem do circuito do jogo da memória em protoboard (visão esquemática). Podemos identificar o Arduino Uno à baixo de uma protoboard à esquerda: os quatro LEDs coloridos na parte superior do protoboard, cada um em série com um resistor até o barramento de GND (fios pretos). Logo abaixo dos LEDs estão quatro botões tátteis lado a lado e um botão extra para o start do jogo; um lado de todos os botões está ligado ao barramento de +0V (fios pretos), e o outro lado de cada botão ligado a um pino digital do Arduino (fios vermelho, amarelo, verde e azul). Na parte superior a direita da protoboard, vemos o buzzer (com fio lilás indo para o pino 3 do Arduino e outro terminal no GND). O display LCD 16x2 está conectado ao Arduino via módulo I2C (fios verde e amarelo para SDA/SCL em A4/A5, vermelho para 5V, preto para GND). Por fim, a bateria 9V está conectada ao conector de energia do Arduino para alimentação externa.

Na imagem acima, confirmamos:

- Os **LEDs** (na imagem provavelmente distinguíveis pelas cores) estão ligados cada um a um resistor (podemos ver resistores próximos a eles) e indo ao GND comum (todos os resistores aterrados na linha azul/negativa). Os fios coloridos saindo dos LEDS para o Arduino correspondem aos pinos digitais designados.
- Os **botões** (representados por pequenos cubos marrons) têm fios pretos unindo um lado deles ao +0V (linha preta ou azul do protoboard), e fios coloridos (de acordo com a cor dos leds) conectando o outro lado a entradas do Arduino.
- O **buzzer** aparece como um cilindro preto, com um fio indo para o Arduino (pino 3) e o outro para GND (provavelmente não visível pois pode estar conectado internamente).
- O **LCD com módulo I2C**: vemos o LCD ao fundo com cabos indo para o Arduino: um vermelho (5V), um preto (GND), e os de dados (verde e amarelo por exemplo) indo para A4/A5.
- A **bateria 9V** está ligada ao Arduino via o conector jack, fornecendo energia (observamos o plug conectado na placa).
- A organização de fios está arrumada para clareza, mas note como todas as conexões descritas estão presentes.

Essa imagem serve como referência visual. Se o seu circuito montado não está “igualzinho” em layout, não tem problema, contanto que as conexões elétricas sejam as mesmas. O importante é a correspondência de cada componente ao seu ponto correto no Arduino e ao GND/5V.

4.9 Recomendações de Segurança e Organização

- **Desligado ao montar:**
Never make changes to connections with the Arduino powered (USB or battery plugged). This prevents shorts that can occur when plugging wires. Always turn off power, organize wires, and then turn on and test.
- **Fios bem encaixados:**
Make sure jumpers are well inserted into the holes; sometimes they look like they are but didn't make contact. If something isn't working, try pushing the wires or trying another hole (it might be a bad connection on the protoboard).
- **Evitar curto-circuitos:**
The places with the highest risk of short are: wires of +5V touching GND or outputs. Keep the protoboard organized, if possible use short and adjusted wires. For example, a LED anode should never jump over a resistor directly to GND – this would short when turned on. A button should never connect 5V directly to GND when pressed (in our case, it connects 5V to input, so it's ok). If you notice overheating or strange smell when turning on, disconnect immediately and inspect the circuit for shorts.
- **Cuidado com polaridade de componentes:**
An inverted LED does not light up (but it doesn't break if inverted, it just doesn't conduct in the wrong direction). An active buzzer inverted may not sound or sound weak. An inverted LCD (if it swaps VCC/GND, it could burn the I2C board – pay attention to this).
- **Fios do mesmo grupo juntos:**
To facilitate, group visually: for example, use black wires for all GND connections (resistors to GND, buzzer GND, etc.), red for +5V, and other colors for signals. This helps the eye identify problems (for example, if a red wire is going where it shouldn't, it's a suspect).
- **Botões próximos:**
When placing buttons side-by-side, be careful not to connect them accidentally. If they are on separate lines, there is no problem, but if two buttons share the same line on one side, pressing one will affect the other. Make sure each button is isolated. On the protoboard, horizontal lines of 5 pins are interconnected – if two buttons are accidentally sharing the same signal line on one side,

eles ficariam "OR" logic, o que talvez não cause grande estrago, mas impediria leitura individual. É fácil evitar: dê pelo menos uma linha vazia de intervalo ou use colunas separadas.

- **Resistores proximidade:**

Nossos resistores no protoboard que vão ao GND ficam próximos. Tome cuidado para não conectar ambos terminais do resistor no mesmo barramento por engano. Resistores devem conectar a duas nodes diferentes (LED e GND); se sem querer colocar as duas pontas do resistor na mesma linha, ele não faz nada.

- **Verificação final passo a passo:**

Uma boa prática antes de ligar é pegar um marcador e um diagrama e ticá-lo:

- LED vermelho anodo -> pino10, cátodo->res->GND
- LED amarelo anodo -> pino8, cátodo->res->GND
- LED verde anodo -> pino6, cátodo->res->GND
- LED azul -> pino4 -> res->GND..
- Botão fio branco -> 12 & 0V etc...

Assim garante que nada ficou esquecido.

4.10 Dicas Práticas para Crianças Montarem

Montar circuitos pode parecer um quebra-cabeças inicialmente. Aqui vêm dicas que facilitam especialmente para jovens estudantes:

- **Monte por partes e teste incrementalmente:**

Por exemplo, primeiro monte um LED e seu resistor e escreva um código rapidinho de blink para testá-lo. Depois acrescente outro LED e teste. Em seguida os botões (testando leitura via Serial print se pressionados), etc. Isso evita o efeito "montei tudo e não funciona, e agora não sei onde está o erro". No contexto didático, é ótimo ir conquistando vitórias parciais.

- **Use código de teste simples:**

O professor ou pai pode fornecer sketches de teste para cada etapa. Ex: um código que acende todos LEDs em sequência para verificar a fiação deles antes de prosseguir ao código final. Ou um código que lê os botões e imprime no PC.

- **Manter a mesa organizada:**

Separe os componentes necessários antes de começar. Evite muita bagunça de peças, pois isso confunde. Talvez colar uma etiqueta ou fita colorida nos fios correspondentes às cores dos LEDs ajude (ex: um adesivo verde no fio do LED verde e no próprio LED).

- **Ajuda visual:**

Muitos iniciantes gostam de seguir diagramas de montagem estilo Fritzing (como a imagem que fornecemos). Tenha essa imagem impressa ou no tablet para referencia. Entretanto, incentive a não somente copiar, mas entender passo a passo ("Agora vamos ligar todos os negativos dos LEDs no GND, veja a cor dos fios...").

- **Evitar força excessiva:**

Ao inserir componentes no protoboard, não entortar pernas abruptamente ou forçar demais – leds e botões podem quebrar se manuseados bruscamente. Ensine a segurar e empurrar com cuidado, às vezes usar um alicate de bico para segurar a perna enquanto encaixa.

- **Trabalhar em dupla ou trio:**

Muitas vezes nas aulas, montar em equipe ajuda, pois um pode ler as instruções enquanto o outro conecta, e todos aprendem discutindo ("Você ligou o vermelho no pino correto?" etc.). Só cuidado para todos participarem, não deixar um fazer tudo.

- **Pedir ajuda e revisar:**

Se não acender algo, não se frustre. É comum errar alguma coisinha. Vá debugando: aquele LED está invertido? Aquele fio soltou? Use método científico: troque uma peça por vez, ou teste o LED direto no 5V com resistor para ver se não está queimado, etc.

- **Orgulho do feito:**

-]Uma vez montado corretamente, celebre! Tire uma foto do seu circuito, mostre para os colegas. Essa é a base do seu jogo eletrônico, montado do zero!

Com a montagem pronta, no próximo capítulo, iremos **programar o jogo da memória** e então **testar o funcionamento** completo do projeto (primeiro com o Arduino ligado ao computador para monitorar, depois de forma autônoma com a bateria). Se algo não funcionar logo de cara, não desanime – vamos aprender a identificar problemas tanto de hardware quanto de software e corrigi-los. A persistência faz parte da robótica!

Mas até aqui, se tudo foi conectado adequadamente, você deve ter um protótipo robusto montado. No próximo passo, daremos vida a ele com código!

Exercícios – Unidade 04

1. **Checklist LEDs:** Liste as conexões necessárias de **um LED** no circuito. (Por exemplo: “Anodo do LED -> pino digital X através de fio Y; Cátodo do LED -> resistor -> GND”).

2. **Botões e Pull-down:** No nosso circuito, não usamos resistores de pull-down nos botões. O que garante que o pino do Arduino leia LOW quando o botão não está pressionado? (Dica: consideramos que ele “flutua” próximo de 0, mas qual fenômeno ou característica faz isso ocorrer?)

3. **Protoboard:** Se um LED não acendeu, uma possível causa é colocar ambas as pernas do LED na mesma linha da protoboard. O que acontece se alguém fizer isso?

4. **Cores dos Fios:** Por que é recomendado usar fio vermelho para +5V e preto para GND? O que pode acontecer se você misturar as cores aleatoriamente?

5. **Curto-circuito:** Imagine que, por engano, um fio do +5V encostou diretamente num fio do GND na montagem. O que aconteceria quando ligasse o Arduino? Cite dois possíveis sintomas.

6. **Botão não responde:** Você percebe que ao apertar um dos botões, o Arduino não detecta (o jogo não reage). Qual pode ser a causa? (a) O botão foi colocado errado na protoboard e suas pernas estão todas do mesmo lado do canal; (b) O fio do botão ao Arduino está no pin errado ou solto; (c) O botão não está recebendo 5V; (d) Qualquer uma das anteriores.

7. **Organização:** Qual componente você achou mais desafiador de conectar corretamente? Como você garantiria que está certo (ex: revisando com imagem, pedindo colega para conferir)?

8. **Teste incremental:** Explique com suas palavras por que testar o circuito em partes (por exemplo, primeiro só LEDs, depois só botões) pode ajudar na montagem final.

9. **Evitar danos:** Você conectou tudo, e quer testar o jogo com a bateria de 9V. Que passos você toma antes de plugar a bateria? (pense: conferir polaridade, remover USB, etc.)

10. **Imagen de referência:** Olhe atentamente a imagem do circuito montado. Descreva algo que você reconheceu nela que bate com o que você montou, e algo que te ajudou a entender a conexão de algum componente.

UNIDADE 05: Programação do Jogo da Memória

Com o circuito montado e conferido, estamos prontos para dar vida ao projeto através da **programação!** Nesta unidade, vamos dissecar o código completo do jogo da memória, entendendo cada parte: desde a declaração de variáveis, passando pelas funções principais (`setup()` e `loop()`), até a lógica que gera a sequência aleatória, lê os botões do jogador e determina vitória ou derrota. Faremos uma análise linha a linha, explicando de forma acessível o que o código faz. Também apresentaremos um **fluxograma** (lógica de blocos) para visualizar o funcionamento do jogo, o que ajuda a entender a sequência de etapas e decisões tomadas pelo programa.

Lembre-se: se você não programou muito em Arduino antes, não tem problema. Vamos guiá-lo pelos conceitos no contexto do projeto. Você verá comandos já conhecidos (como `digitalWrite`, `pinMode`, `if`, `for loops`) e aprenderá alguns novos (como `random()` e `tone()`). Ao final desta unidade, você deve ser capaz de explicar como o jogo funciona internamente e, quem sabe, até pensar em modificar algo ou melhorar!

5.1 Visão Geral do Código

Antes de mergulhar nos detalhes, vamos entender **o que o programa precisa fazer** em alto nível:

1. Configurar o hardware:

definir os modos dos pinos (entradas para botões, saídas para LEDs e buzzer), iniciar o display LCD, etc.

2. Esperar início do jogo:

no início, mostrar uma mensagem ("Vamos jogar?") e acender as luzes para indicar pronto, e esperar o jogador apertar o botão de iniciar (no nosso caso, escolhemos o botão verde como "start").

3. Gerar sequência e mostrar ao jogador:

o jogo escolherá aleatoriamente uma nova cor (LED) para adicionar na sequência, acenderá as luzes da sequência (do primeiro até o novo) para o jogador memorizar, tocando sons correspondentes.

4. Ler a resposta do jogador:

então esperará o jogador reproduzir a sequência apertando os botões em ordem. Para cada botão pressionado:

- Acender o LED correspondente, tocar o som,
- Checar se está correto (se corresponde à próxima cor da sequência esperada).
- Se errar, fim de jogo.
- Se acertar, continua esperando o próximo botão.

5. Verificar se completou a sequência:

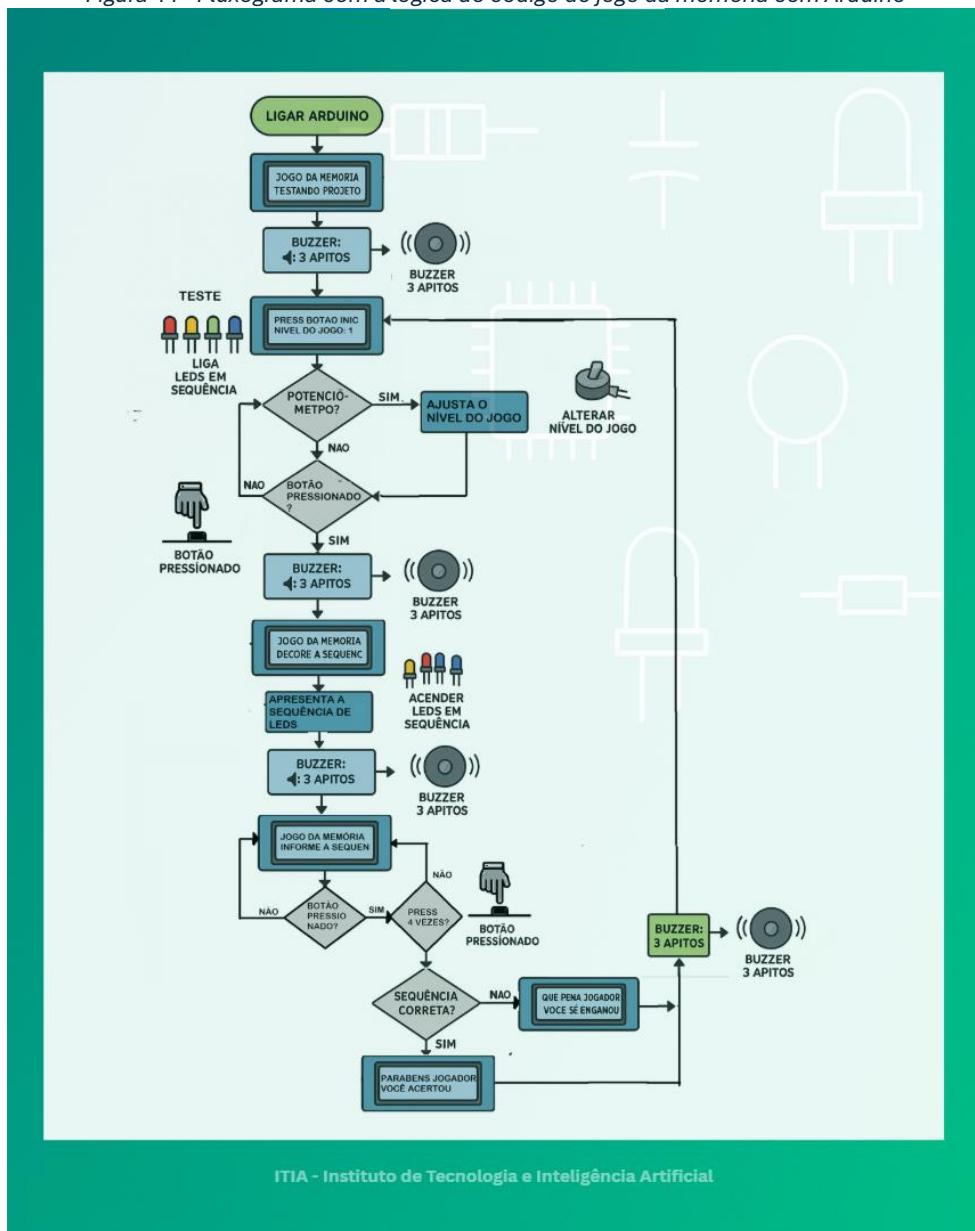
se o jogador conseguiu repetir toda a sequência atual corretamente, incrementa a pontuação e volta ao passo 3 (adicionando mais uma cor e aumentando o desafio).

6. Encerrar jogo:

se o jogador errar em algum momento, tocar um som de erro (buzzer longo), mostrar "ERROU" e a pontuação final no LCD, e então basicamente reiniciar para o estado inicial (voltar a "Vamos jogar?" esperando novo start).

Em resumo, é um laço de jogo que continua enquanto o jogador acerta, e quebra quando ele erra. Cada rodada aumenta a sequência de cores em +1 e a pontuação do jogador em +1.

Figura 44 - Fluxograma com a lógica do código do jogo da memória com Arduino



Sabendo disso, fica mais fácil entender o código e suas variáveis.

5.2 Código Completo

A seguir, apresentamos o código do jogo da memória. Vamos exibi-lo em partes comentadas para analisar. Preste atenção nos comentários // ao lado, pois eles explicam o propósito de cada linha ou bloco.

```
/*=====
 * JOGO DA MEMÓRIA - Versão final (LCD I2C + pinos revistos)
 -----
 Mapping físico (ver esquema de fios coloridos):
    LEDs ..... VERM = 10 | AMAR = 8 | VERD = 6 | AZUL = 4
    Botões ... INIC = 12 | VERM = 11 | AMAR = 9 | VERD = 7 | AZUL = 5
    Buzzer ..... 3
    Pot. ..... A0
    LCD I2C ..... SDA = A4 SCL = A5 (addr 0x27)
=====
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// ----- PINOS DE SAÍDA -----
const int LED_VERM = 10;
const int LED_AMAR = 8;
const int LED_VERD = 6;
const int LED_AZUL = 4;
const int LEDs[4] = {LED_VERM, LED_AMAR, LED_VERD, LED_AZUL};

const int BUZZER = 3;

// ----- PINOS DE ENTRADA -----
const int BTN_START = 12; // Botão branco (Start/Reset)
const int BTN_VERM = 11;
const int BTN_AMAR = 9;
const int BTN_VERD = 7;
const int BTN_AZUL = 5;
const int BOTOES[4] = {BTN_VERM, BTN_AMAR, BTN_VERD, BTN_AZUL};

const int POT_NIVEL = A0; // Potenciômetro (cursor fio ciano)

// ----- LCD -----
LiquidCrystal_I2C lcd(0x27, 16, 2);

// ----- CONSTANTES -----
const byte TAM_SEQ = 5;
byte sequencia[TAM_SEQ]; // Guarda sequência sorteada
const uint16_t intervalo[5] = {2000, 1600, 1200, 1000, 800};
```

Vamos destrinchar o código um pouco mais:

```
/*=====
 JOGO DA MEMÓRIA - Versão final (LCD I2C + pinos revistos)
=====
```

Esse comentário identifica a **versão consolidada** do código do jogo, destacando que agora está utilizando **LCD com interface I²C** (mais simples de conectar) e que os **pinos foram revistos** de acordo com o circuito real.

Mapeamento físico (para montagem e verificação prática)

Mapping físico (ver esquema de fios coloridos):

LEDs	VERM = 10		AMAR = 8		VERD = 6		AZUL = 4		
Botões ...	INIC = 12		VERM = 11		AMAR = 9		VERD = 7		AZUL = 5
Buzzer	3								
Pot.	A0								
LCD I ² C	SDA = A4 SCL = A5 (addr 0x27)								

=====*/

Serve como **guia rápido** de referência para o professor ou aluno montar corretamente o circuito, **relacionando cor do fio, função e número do pino**.

O uso dos **fios coloridos** (como no esquema ilustrado) facilita a **associação visual** entre código e hardware.

Observação especial ao uso do **I²C** (SDA em A4, SCL em A5), indicando também o endereço hexadecimal padrão (0x27) do módulo LCD com expansor PCF8574.

Inclusão das bibliotecas

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
```

`Wire.h` é a biblioteca padrão do Arduino para comunicação via **I²C**.

`LiquidCrystal_I2C.h` é uma biblioteca auxiliar que simplifica a utilização do display **LCD** através do **I²C**.

Ambas são fundamentais para controlar o **LCD 16x2**, e reduzem significativamente a quantidade de fios e comandos necessários em comparação ao modo 4 ou 8 bits tradicional.

Declaração dos pinos de saída (LEDs e Buzzer)

```
// ----- PINOS DE SAÍDA -----
const int LED_VERM = 10;
const int LED_AMAR = 8;
const int LED_VERD = 6;
const int LED_AZUL = 4;
const int LEDs[4] = {LED_VERM, LED_AMAR, LED_VERD, LED_AZUL};
```

Aqui são definidos os **pinos digitais** que controlam os quatro **LEDs coloridos**.

As constantes tornam o código mais **legível e intuitivo** — evita usar “números mágicos” dentro do código.

A criação do **array LEDs[4]** é uma prática inteligente para iterar facilmente sobre todos os **LEDs** com loops *for*, tornando o código mais modular e compacto.

```
const int BUZZER = 3;
```

Define o pino de controle do **buzzer piezoelétrico** (emite sons). Também será utilizado com a função `tone()` para gerar sons com frequência e duração controladas.

Declaração dos pinos de entrada (botões e potenciômetro)

```
// ----- PINOS DE ENTRADA -----
const int BTN_START = 12; // Botão branco (Start/Reset)
const int BTN_VERM = 11;
const int BTN_AMAR = 9;
const int BTN_VERD = 7;
const int BTN_AZUL = 5;
const int BOTOES[4] = {BTN_VERM, BTN_AMAR, BTN_VERD, BTN_AZUL};
```

Mapeamento dos **botões tátteis**, incluindo o botão branco de início (`BTN_START`) e os quatro botões de resposta, cada um da mesma cor de seu LED. Assim como com os LEDs, os botões são agrupados num array `BOTOES[4]`, permitindo varrer todos eles com loops for de forma prática. Os botões são configurados posteriormente como **INPUT_PULLUP**, ou seja, trabalham com **lógica invertida** (pressionado = LOW).

```
const int POT_NIVEL = A0; // Potenciômetro (cursor fio ciano)
```

O potenciômetro está ligado ao pino **análogo A0** e é usado para escolher o **nível de dificuldade** (controlando o tempo de exibição dos LEDs). O valor de entrada varia entre **0** e **1023**, sendo mapeado para valores de nível de 1 a 5 com a função `map()`.

Declaração do LCD via I²C

```
// ----- LCD -----
LiquidCrystal_I2C lcd(0x27, 16, 2);
```

Criação do objeto `lcd`, que representa o display. 0x27 é o endereço padrão I²C da maioria dos módulos LCD com expensor PCF8574. O display é do tipo **16 colunas por 2 linhas** (16x2), bastante comum em kits Arduino.

Constantes auxiliares

```
// ----- CONSTANTES -----
const byte TAM_SEQ = 5;
byte sequencia[TAM_SEQ]; // Guarda sequência sorteada
const uint16_t intervalo[5] = {2000, 1600, 1200, 1000, 800};
```

`TAM_SEQ` define o **tamanho da sequência de memória** (quantidade de passos por rodada).

`sequencia[]` é um **vetor que armazena os índices** correspondentes às cores sorteadas (0 a 3).

`intervalo[]` determina o **tempo de exibição** dos LEDs em cada nível (nível 1 = 2000 ms, nível 5 = 800 ms).

↳ Essa abordagem facilita alterar o tempo por nível sem repetir valores no código principal.

Continuando o código, vamos ao `setup()`:

```
// -----
void setup() {
    // Configura pinos digitais
    for (byte i = 0; i < 4; i++) {
        pinMode(LEDs[i], OUTPUT);
        pinMode(BOTOES[i], INPUT_PULLUP);
    }
    pinMode(BTN_START, INPUT_PULLUP);
    pinMode(BUZZER, OUTPUT);

    // Inicia LCD
    lcd.init();
    lcd.backlight();

    // Mensagem de teste
    lcd.setCursor(0,0); lcd.print("JOGO DA MEMORIA");
    lcd.setCursor(0,1); lcd.print("TESTANDO PROJETO");
    piscaleds(1, 200);
    beep(3, 100, 100);

    // Semente aleatória
    randomSeed(analogRead(A1));
}
```

Análise detalhada da função `setup()`

```
// -----
void setup() {
```

`setup()` é executada **uma única vez** sempre que a placa Arduino é energizada ou reiniciada. Aqui preparamos todo o hardware antes de iniciar o laço principal (`loop()`).

```
// Configura pinos digitais
for (byte i = 0; i < 4; i++) {
    pinMode(LEDs[i], OUTPUT);
    pinMode(BOTOES[i], INPUT_PULLUP);
}
```

Loop for percorre os índices 0-3, simplificando a tarefa de configurar quatro LEDs e quatro botões sem repetir código.

```
pinMode(LEDs[i], OUTPUT);
```

- Cada elemento do array LEDs[] (pinos **10, 8, 6, 4**) é declarado **saída digital** permitirá ligar/apagar o LED aplicando HIGH/LOW.

```
pinMode(BOTOES[i], INPUT_PULLUP);
```

- Os quatro botões coloridos (pinos **11, 9, 7, 5**) tornam-se **entradas com resistor de pull-up interno**.
- Significa que, em repouso, o pino lê HIGH; ao pressionar, o botão conecta à linha de **GND (fio preto)** e o valor cai para LOW.
- Explicar aos alunos: evita usar resistores externos e simplifica a fiação.

Botão de início e buzzer

```
pinMode(BTN_START, INPUT_PULLUP);
pinMode(BUZZER, OUTPUT);
```

BTN_START (pino **12**, botão branco) também é configurado com **INPUT_PULLUP**, obedecendo mesma lógica de LOW quando pressionado.

BUZZER (pino **3**, fio roxo) como **saída**: o microcontrolador enviará pulsos que o piezo transforma em som (função `tone()`).

Inicialização do display LCD I²C

```
// Inicia LCD
lcd.init();
lcd.backlight();
```

Aqui nós temos as funções de inicialização:

- `lcd.init();` estabelece comunicação I²C no endereço **0x27** usando **SDA (A4)** e **SCL (A5)**.
- `lcd.backlight();` acende o LED interno do módulo, garantindo legibilidade.
- Mostrar aos alunos que, graças ao I²C, apenas **dois fios de dados** controlam todo o display, versus seis no modo paralelo.

Self-test visual e sonoro

```
// Mensagem de teste
lcd.setCursor(0,0); lcd.print("JOGO DA MEMORIA");
lcd.setCursor(0,1); lcd.print("TESTANDO PROJETO");
piscalLeds(1, 200);
beep(3, 100, 100);
```

Nesta parte do código se faz os testes de LCD, LED's e buzzer:

- Escreve título e status na primeira e segunda linhas do LCD.
- `piscalLeds(1, 200);` percorre os quatro LEDs uma vez, ligados por **200 ms** cada → verifica fiação de saída.
- `beep(3, 100, 100);` gera **3 bipes de 100 ms com 100 ms de pausa** → confirma buzzer e reforça feedback multimodal (visão + audição).

Semente de aleatoriedade

```
// Semente aleatória
randomSeed(analogRead(A1));
}
```

`randomSeed()` inicializa o gerador de números pseudo-aleatórios.

Lê **A1** (desconectado) para capturar ruído analógico e produzir uma semente imprevisível → garante que cada partida crie **sequências diferentes de LEDs**, aumentando a rejogabilidade.

Excelente gancho didático para discutir “aleatoriedade” em computadores e ruído elétrico.

Agora entra o loop principal. Ele é extenso, vamos por partes:

```
void loop() {
    byte nivel = escolherNivel();      // 1) Potenciômetro + botão Start
    gerarSequencia();                 // 2) Array aleatório
    exibirSequencia(nivel);          // 3) Mostra LEDs
    bool venceu = lerResposta();      // 4) Jogador responde
    mostrarResultado(venceu);        // 5) Feedback
}
```

A função `loop()` é o **coração do Arduino** – ela se repete infinitamente enquanto a placa estiver ligada. Neste projeto do **Jogo da Memória**, ela organiza o fluxo completo de uma rodada do jogo, dividindo-a em **cinco etapas bem definidas**, fáceis de explicar ao aluno e de relacionar com o fluxograma visto anteriormente.

Etapa 1: `escolherNivel();`

```
void loop() {
    byte nivel = escolherNivel();      // 1) Potenciômetro + botão Start
```

Chamada e retorno da função responsável por escolher o nível do jogo:

- O jogo começa com o **usuário ajustando a dificuldade** através do potenciômetro conectado ao pino **A0** (fio ciano).
- Ao girar o botão, a tensão lida é convertida em um valor entre **nível 1 e nível 5** usando a função `map()`.
- Quando o botão **START** (pino 12, fio branco) é pressionado, o nível atual é confirmado.
- O valor é armazenado na variável `nivel` e será usado para determinar a **velocidade** com que os LEDs da sequência serão mostrados.

Objetivo pedagógico: Explora leitura analógica, mapeamento de valores e reforça o conceito de entrada ativa (ação do usuário).

Etapa 2: gerarSequencia();

```
gerarSequencia();           // 2) Array aleatório
```

Chamada da função responsável por gerar a sequência que os LED's irão ligar e desligar para que o jogador memorize:

- Cria um vetor sequencia[] com 5 valores aleatórios entre 0 e 3 (referentes aos 4 LEDs).
- A função usa random() e gera combinações que o jogador deverá memorizar e reproduzir.

Importância: Mostra o uso de **vetores**, **laços for**, e o conceito de indexação. Também reforça a aleatoriedade como ferramenta para criar jogos imprevisíveis.

Etapa 3: exibirSequencia(nível);

```
exibirSequencia(nível);      // 3) Mostra LEDs
```

- Mostra visualmente a sequência de LEDs, com **tempo de exibição definido pelo nível escolhido**.
- A cada elemento da sequência, o LED correspondente é aceso por um tempo curto (t_on), depois apagado por um tempo de pausa (t_off).
- O buzzer também pode emitir sinais sonoros como reforço.

Conceito trabalhado: Temporização (delay()), controle de saídas, uso de LCD para instruções.

Etapa 4: lerResposta();

```
bool venceu = lerResposta();    // 4) Jogador responde
```

- Após ver a sequência, o jogador deve **reproduzi-la corretamente** apertando os botões coloridos na mesma ordem.
- A função lê os botões um por um, aguardando que sejam pressionados na ordem certa.
- Se errar um botão, a função retorna false, indicando que o jogador perdeu a rodada.

Importância didática: Envolve leitura digital, comparação de dados e controle de fluxo com if.

Etapa 5: mostrarResultado(venceu);

```
mostrarResultado(venceu);      // 5) Feedback  
}
```

- Dá o **feedback final** ao jogador no LCD e por meio de bipes no buzzer.
- Se venceu for true, mostra mensagem de vitória; caso contrário, exibe erro.

Conclusão: Ensina como dar **respostas visuais e sonoras**, essenciais em jogos interativos. Também evidencia o uso de funções booleanas.
Agora vamos entrar nas funções:

Função: piscaLeds

```
void piscaLeds(byte vezes, uint16_t tOn) {
    for (byte v = 0; v < vezes; v++) {
        for (byte i = 0; i < 4; i++) {
            digitalWrite(LEDs[i], HIGH);
            delay(tOn);
            digitalWrite(LEDs[i], LOW);
        }
    }
}
```

Objetivo da Função:

A função piscaLeds tem como **objetivo fazer os 4 LEDs do jogo acenderem um por um em sequência**, repetindo esse ciclo uma quantidade de vezes definida pelo parâmetro vezes. Esse efeito é usado como um **teste visual** no início do jogo, ou em outras partes do programa, como uma espécie de "animação de boas-vindas".

Parâmetros:

Parâmetro	Tipo	Significado
vezes	byte	Quantas vezes a sequência dos 4 LEDs deve ser repetida
tOn	uint16_t	Tempo (em milissegundos) que cada LED ficará aceso

Estrutura da Função:

- Laço externo for (byte v = 0; v < vezes; v++)**
 - Objetivo:** Repetir o ciclo completo de piscadas dos 4 LEDs.
 - Exemplo:** Se vezes = 2, a função fará duas voltas no conjunto completo dos LEDs (vermelho → amarelo → verde → azul).
- Laço interno for (byte i = 0; i < 4; i++)**
 - Objetivo:** Acessar cada um dos quatro LEDs individualmente.
 - Os LEDs estão armazenados no vetor LEDs[], que foi definido assim:

```
const int LEDs[4] = {LED_VERM, LED_AMAR, LED_VERD, LED_AZUL};
```

3. Dentro do laço interno:

```
digitalWrite(LEDs[i], HIGH);
delay(tOn);
digitalWrite(LEDs[i], LOW);
```

- **Acende o LED atual** (LEDs[i]) — por exemplo, LED_VERM (vermelho).
- **Espera o tempo tOn** — por exemplo, 200 ms.
- **Desliga o LED.**

Esse processo acontece para os 4 LEDs, um após o outro, e o ciclo se repete de acordo com o valor de vezes.

Exemplo de Uso:

Suponha que você chame:

```
piscaLeds(2, 150);
```

O que acontece?

1. LED vermelho acende por 150 ms → apaga.
2. LED amarelo acende por 150 ms → apaga.
3. LED verde acende por 150 ms → apaga.
4. LED azul acende por 150 ms → apaga.
5. Esse ciclo é repetido uma segunda vez.

Aplicações Didáticas:

- **Introduz laços aninhados** (for dentro de for).
- Ensina o uso de **vetores para componentes físicos**.
- Mostra como **modularizar efeitos visuais** em funções reutilizáveis.
- Permite que os alunos vejam **respostas imediatas físicas** no hardware — o que é muito importante no ensino com Arduino.

Função: beep

```
void beep(byte v, uint16_t tOn, uint16_t pausa) {  
    for (byte i = 0; i < v; i++) {  
        tone(BUZZER, 1000);           // 1 kHz  
        delay(tOn);  
        noTone(BUZZER);  
        delay(pausa);  
    }  
}
```

Objetivo da Função:

A função beep emite **sinais sonoros curtos (beeps)** em um buzzer passivo conectado ao Arduino. Ela serve como **feedback sonoro** para ações importantes no jogo — como início, acertos ou erros.

Parâmetros:

Parâmetro	Tipo	Significado
v	byte	Número de beeps (quantas vezes repetir o som)
tOn	uint16_t	Duração de cada beep (tempo com som), em milissegundos
pausa	uint16_t	Intervalo entre os beeps (tempo em silêncio), em milissegundos

Explicação do Código:

1. Laço for (byte i = 0; i < v; i++)

- Esse laço controla **quantas vezes o som será emitido**.
- Exemplo: se v = 3, ele repetirá o processo de tocar e pausar 3 vezes.

2. Comando tone(BUZZER, 1000);

- Ativa o **buzzer** emitindo uma frequência de **1000 Hz (1 kHz)**.
- Esse valor gera um som audível agudo típico de um beep.
- O pino do buzzer foi previamente definido como:

```
const int BUZZER = 3;
```

- A função tone() envia **um sinal PWM** ao pino do buzzer, fazendo-o vibrar e emitir som.

3. delay(tOn);

- Espera **pelo tempo de duração do som**, determinado por tOn.
- Exemplo: se tOn = 100, o som dura 100 milissegundos (0,1 segundo).

4. noTone(BUZZER);

- Interrompe o som no buzzer.
- Essa função **desativa a frequência PWM** gerada por tone().

5. delay(pausa);

- Aguarda o tempo em silêncio antes de emitir o próximo beep.

Exemplo de Uso:

Se você chamar:

```
beep(3, 150, 100);
```

O que acontece?

- O buzzer toca um som de 1 kHz por 150 ms → pausa por 100 ms
- Isso é repetido 3 vezes

Aplicações Didáticas:

- Introduz conceitos de **frequência sonora** e **tempo de sinal**.
- Ensina como gerar **feedback auditivo com Arduino**, importante para jogos e sistemas de alerta.
- Mostra uso de **funções prontas da biblioteca Arduino** (tone() e noTone()), e como combiná-las com delay() para controlar sons.

Função: escolherNivel()

Objetivo

Essa função permite que o jogador escolha o nível de dificuldade do jogo, girando o potenciômetro e confirmando a seleção com o botão branco de início. A escolha é exibida em tempo real no display LCD.

```
byte escolherNivel() {
    byte niv = 1;
    lcd.clear();
    lcd.setCursor(0,0); lcd.print("PRESS BOTAO INIC");
    while (digitalRead(BTN_START) == HIGH) {
        byte novo = map(analogRead(POT_NIVEL),0,1023,1,5);
        if (novo != niv) {
            niv = novo;
            lcd.setCursor(0,1);
            lcd.print("NIVEL DO JOGO: ");
            lcd.print(niv);
            lcd.print(" ");           // apaga dígito antigo
        }
        delay(100);
    }
    beep(3, 80, 80);
    while (digitalRead(BTN_START) == LOW); // espera soltar
    delay(200);
    return niv;
}
```

Explicação Passo a Passo

```
byte escolherNivel() {
    byte niv = 1;
```

Inicializa a variável niv com valor 1. Esse é o **nível padrão** antes de qualquer movimentação do potenciômetro.

```
lcd.clear();
```

Limpa o display LCD, apagando qualquer mensagem anterior.

```
lcd.setCursor(0,0); lcd.print("PRESS BOTAO INIC");
```

Mostra na **primeira linha do LCD** a mensagem orientando o jogador a **pressionar o botão de início (branco)**.

```
while (digitalRead(BTN_START) == HIGH) {
```

Enquanto o botão de início **não for pressionado**, o Arduino:

- Lê o valor do **potenciômetro** (ligado ao pino A0, com fio **ciano** no circuito)
- Usa map() para converter a leitura analógica (0 a 1023) para um valor inteiro entre **1 e 5**, que representa os **níveis do jogo**

```
byte novo = map(analogRead(POT_NIVEL), 0, 1023, 1, 5);
```

Se o valor lido for diferente do anterior, atualiza o LCD com:

```
if (novo != niv) {
    niv = novo;
    lcd.setCursor(0,1);
    lcd.print("NIVEL DO JOGO: ");
    lcd.print(niv);
```

O espaço extra " " é uma técnica simples para apagar possíveis dígitos anteriores (por exemplo, ao trocar do 10 para o 9, o dígito 0 anterior não ficaria na tela).

```
lcd.print(" ");
} // apaga dígito antigo
delay(100);
```

Insere uma **pequena pausa** (100ms) para evitar leituras muito rápidas e instáveis do potenciômetro (debounce visual).

```
}
```

```
beep(3, 80, 80);
```

Ao pressionar o botão de início, emite 3 beeps curtos como feedback sonoro, confirmando a seleção.

```
while (digitalRead(BTN_START) == LOW); // espera soltar
```

Espera que o botão de início seja **soltó** antes de continuar, para evitar múltiplas leituras ou duplo clique.

```
delay(200);
```

Insere uma **pequena pausa final** para estabilidade da interface.

```
return niv;
```

```
}
```

Retorna o valor do **nível escolhido** (entre 1 e 5) para que o programa principal (loop()) use na próxima etapa.

Exemplo Prático

Imagine o seguinte:

- O jogador gira o potenciômetro até escolher o nível 4
- O LCD mostra:

PRESS BOTAO INIC

NIVEL DO JOGO: 4

- Ao pressionar o botão branco, o buzzer apita 3x
- O valor 4 é retornado para o jogo configurar a velocidade da sequência

Aplicação Pedagógica

Essa função é um **ótimo exemplo de integração entre entrada analógica (potenciômetro), entrada digital (botão), saída visual (LCD) e saída sonora (buzzer)**.

Ela permite aos alunos:

- Trabalhar com **mapeamento de faixas analógicas**
- Praticar **interface de usuário com Arduino**
- Criar sistemas de **seleção dinâmica com confirmação**
- Refletir sobre **debounce** e feedback em interfaces interativas

Função: gerarSequencia()

Objetivo

Gerar **aleatoriamente** uma sequência de cores (representadas por LEDs) que o jogador deverá **memorizar e repetir**. Essa é a "resposta correta" do jogo.

Código

```
void gerarSequencia() {
    for (byte i = 0; i < TAM_SEQ; i++)
        sequencia[i] = random(0,4);
}
```

Explicação Passo a Passo

```
for (byte i = 0; i < TAM_SEQ; i++)
```

Laço que **varre o vetor sequencia[]** da posição 0 até TAM_SEQ - 1.

Lembre-se: TAM_SEQ é uma constante definida anteriormente:

```
const byte TAM_SEQ = 5;
```

Ou seja, o vetor sequencia[] tem **5 posições**: 0, 1, 2, 3, 4.

```
sequencia[i] = random(0, 4);
```

A cada iteração, o código:

- Chama a função random(0, 4), que **gera um número aleatório inteiro entre 0 e 3**
- Atribui esse número à posição i do vetor sequencia

Cada número corresponde a **uma das 4 cores de LED** do jogo:

Valor	LED (cor)	Fio no circuito
0	Vermelho	Vermelho
1	Amarelo	Amarelo
2	Verde	Verde
3	Azul	Azul

Resultado Esperado

Suponha que o random() gere os valores: 2, 0, 1, 3, 2

Então o vetor sequencia[] ficará assim:

```
sequencia[0] = 2 // LED Verde
sequencia[1] = 0 // LED Vermelho
sequencia[2] = 1 // LED Amarelo
sequencia[3] = 3 // LED Azul
sequencia[4] = 2 // LED Verde
```

*Essa é a **sequência correta que será exibida ao jogador**, e que ele deverá **reproduzir pressionando os botões** na mesma ordem.*

Observações Técnicas

- A função random() depende de uma **semente aleatória**, que neste projeto é definida no setup() com:

```
randomSeed(analogRead(A1));
```

*Isso garante que cada vez que o Arduino é ligado, uma **sequência diferente** será gerada, criando variedade no jogo.*

Aplicação Pedagógica

Essa função é **simples mas poderosa** para ensinar:

- Conceito de **vetores (arrays)**
- Uso de **números aleatórios**
- Representação simbólica (número → cor/LED)
- Pensamento computacional para **gerar desafios dinâmicos**

Professores podem, inclusive, pedir aos alunos para:

- Exibir no LCD a sequência gerada (modo debug)
- Ampliar o vetor para mais de 5 elementos
- Adicionar restrições (ex: evitar repetir cores consecutivas)

Função: exibirSequencia(byte niv)

Objetivo

Esta função tem a responsabilidade de **mostrar a sequência de LEDs ao jogador**, com base no nível de dificuldade escolhido. Cada cor acende por um tempo controlado, formando uma **sequência visual** que o jogador deverá memorizar.

Código da Função

```
void exibirSequencia(byte niv) {  
    lcd.clear();  
    lcd.setCursor(0,0); lcd.print("JOGO DA MEMORIA");  
    lcd.setCursor(0,1); lcd.print("DECORE A SEQUENC");  
    uint16_t passo = intervalo[niv-1];  
    uint16_t tOn = passo/2, tOff = passo - tOn;  
    delay(600);  
    for (byte i = 0; i < TAM_SEQ; i++) {  
        digitalWrite(LEDs[sequencia[i]], HIGH);  
        delay(tOn);  
        digitalWrite(LEDs[sequencia[i]], LOW);  
        delay(tOff);  
    }  
    beep(3, 80, 80);  
}
```

Etapa por Etapa

Preparação do LCD

```
void exibirSequencia(byte niv) {  
    lcd.clear();  
    lcd.setCursor(0,0); lcd.print("JOGO DA MEMORIA");  
    lcd.setCursor(0,1); lcd.print("DECORE A SEQUENC");
```

- Limpa o visor LCD.
- Exibe duas mensagens:
 - Linha 0: "JOGO DA MEMORIA"
 - Linha 1: "DECORE A SEQUENC" (a palavra "SEQUENC" é truncada devido ao limite de 16 caracteres)

Essas mensagens informam ao jogador que a sequência de LEDs será exibida em instantes.

2. Cálculo do tempo por passo

```
uint16_t passo = intervalo[niv-1];
```

```
uint16_t tOn = passo/2, tOff = passo - tOn;
```

- `intervalo[]` é um vetor com os tempos de exibição, que **diminuem conforme o nível aumenta**, deixando o jogo mais difícil.
- O tempo `passo` é dividido em:
 - `tOn`: tempo com o LED aceso
 - `tOff`: tempo com o LED apagado antes de acender o próximo

Exemplo (nível 2):

```
intervalo[1] = 1600  
→ tOn = 800  
→ tOff = 800
```

3. Pausa antes de iniciar a sequência

```
delay(600);
```

Dá uma **pausa de 600ms** para o jogador se concentrar antes de começar a piscar os LEDs.

4. Mostrar cada LED da sequência

```
for (byte i = 0; i < TAM_SEQ; i++) {  
    digitalWrite(LEDs[sequencia[i]], HIGH);  
    delay(tOn);  
    digitalWrite(LEDs[sequencia[i]], LOW);  
    delay(tOff);  
}
```

- Laço que percorre a sequência aleatória gerada anteriormente.
- Para cada item do vetor `sequencia[]`, ele:
 - Acende o LED correspondente
 - Aguarda `tOn` milissegundos
 - Apaga o LED
 - Espera `tOff` milissegundos antes de passar para o próximo

Isso constrói a **sequência visual de luzes** que o jogador deverá memorizar.

5. Feedback sonoro

```
beep(3, 80, 80);  
}
```

- Após exibir toda a sequência, emite **3 bipes curtos** com o buzzer.
- Serve como **sinal sonoro** para indicar ao jogador que a exibição terminou e é hora de **responder**.

Relação com a montagem do circuito

Cada valor no vetor sequencia[i] corresponde a um LED colorido, conforme a seguinte tabela:

Índice	Cor do LED	Pino Arduino	Cor do Fio no Esquema
0	Vermelho	10	Vermelho
1	Amarelo	8	Amarelo
2	Verde	6	Verde
3	Azul	4	Azul

O código aciona os LEDs através do vetor LEDs[], com base nos índices armazenados em sequencia[].

Aplicação Pedagógica

Essa função é um **exemplo didático excelente** para ensinar:

- Laços for com vetores
- Controle de tempo com delay()
- Manipulação de hardware (LEDs)
- Representação visual de informações
- Adaptação do jogo ao nível de dificuldade

Função: bool lerResposta()

Objetivo

Essa função realiza a **comparação entre a sequência memorizada pelo jogador** e a sequência sorteada pelo sistema. A cada botão pressionado, o código verifica se o botão corresponde ao LED exibido na mesma posição da sequência. Se houver algum erro, a função retorna false. Se o jogador acertar toda a sequência, retorna true.

Código da Função

```
bool lerResposta() {
    lcd.clear();
    lcd.setCursor(0,0); lcd.print("JOGO DA MEMORIA");
    lcd.setCursor(0,1); lcd.print("INFORME A SEQUEN");
    for (byte p = 0; p < TAM_SEQ; p++) {
        int idx = aguardaBotao();
        beep(1, 60, 20);
        if (idx != sequencia[p]) return false;
    }
    return true;
}
```

Etapa por Etapa

1. Exibe instruções no display

```
bool lerResposta() {  
    lcd.clear();  
    lcd.setCursor(0,0); lcd.print("JOGO DA MEMÓRIA");  
    lcd.setCursor(0,1); lcd.print("INFORME A SEQUEN");
```

- Limpa a tela do LCD.
- Mostra a mensagem informando que o jogador deve **informar a sequência**.
- Isso sinaliza que o sistema está esperando **entradas do jogador**.

2. Laço para receber e verificar cada botão pressionado

```
for (byte p = 0; p < TAM_SEQ; p++) {
```

- O loop percorre cada **posição da sequência sorteada** (por padrão, são 5 posições).
- A cada iteração, o jogador precisa pressionar **um botão correspondente ao LED da sequência original**.

3. Aguarda o botão ser pressionado e lê o índice

```
int idx = aguardaBotao();
```

- Chama a função auxiliar aguardaBotao() que:
 - Espera **um dos 4 botões coloridos** ser pressionado.
 - Retorna o índice correspondente (0 a 3) baseado no botão detectado.

Isso transforma a entrada física do botão em um valor lógico comparável com a sequência.

4. Emite um beep curto como feedback

```
beep(1, 60, 20);
```

- Toca **um bip curto** no buzzer após cada botão pressionado.
- Serve como **sinal sonoro de confirmação** de que o sistema registrou a entrada.

5. Verifica se a entrada do jogador está correta

```
if (idx != sequencia[p]) return false;
```

- Compara o índice retornado pelo botão pressionado com o valor armazenado na mesma posição do vetor sequencia[].
- Se houver **qualquer erro**, ou seja, se o jogador pressionar um botão que não corresponde à sequência, a função **interrompe imediatamente** e retorna false.

Essa é a **condição de falha** do jogo.

6. Se todos os botões estiverem corretos...

```
}
return true;
}
```

- Se o jogador acertar todas as 5 posições da sequência, o loop termina e a função retorna true.

Isso representa **vitória** na rodada.

Integração com o hardware

Índice	Cor do Botão	Pino Arduino	LED Correspondente
0	Vermelho	11	LED Vermelho (10)
1	Amarelo	9	LED Amarelo (8)
2	Verde	7	LED Verde (6)
3	Azul	5	LED Azul (4)

O vetor BOTOES[] é usado por aguardaBotao() para mapear essas entradas.

Resumo funcional

- Mostra instruções no LCD.
- Aguarda e lê uma sequência de 5 botões.
- Compara cada botão com a sequência original.
- Emite bipes de confirmação.
- Em caso de erro, retorna false.
- Se acertar tudo, retorna true.

Valor didático

Essa função ensina conceitos importantes:

- Comparação de vetores
- Controle de fluxo (if, for, return)
- Interação com o usuário via botões

- Boas práticas de feedback com buzzer e display
- Modularização usando funções auxiliares (aguardaBotao)

Função: int aguardaBotao()

Objetivo

Aguardar até que o jogador pressione um dos quatro botões coloridos (vermelho, amarelo, verde, azul) e retornar o índice (de 0 a 3) que representa qual botão foi pressionado, com proteção contra leituras erradas (como ruídos ou múltiplos cliques).

Código

```
int aguardaBotao() {
    while (true) {
        for (byte i = 0; i < 4; i++) {
            if (digitalRead(BOTOES[i]) == LOW) {
                while (digitalRead(BOTOES[i]) == LOW); // espera soltar
                delay(40); // debounce
                return i;
            }
        }
    }
}
```

Análise Detalhada

Loop infinito

```
while (true) {
```

- A função entra em um **loop contínuo**, ou seja, **fica travada até que um botão seja pressionado**.

Verificação dos 4 botões

```
for (byte i = 0; i < 4; i++) {
```

- Percorre os 4 botões definidos no vetor BOTOES[], que são:

Índice i	Botão	Cor	Pino
0	BTN_VERM	Vermelho	11
1	BTN_AMAR	Amarelo	9
2	BTN_VERD	Verde	7
3	BTN_AZUL	Azul	5

Detectação de botão pressionado

```
if (digitalRead(BOTOES[i]) == LOW) {
```

- Como os botões estão configurados com INPUT_PULLUP, o valor padrão do pino é HIGH.
- Quando o botão é pressionado, o pino vai para LOW, indicando uma pressão válida.

Espera o botão ser solto

```
while (digitalRead(BOTOES[i]) == LOW); // espera soltar
```

- Após detectar que o botão foi pressionado, o sistema **aguarda até que ele seja solto**.
- Isso evita registrar a mesma pressão mais de uma vez.

Debounce manual

```
delay(40); // debounce
```

- Pequena pausa de 40 milissegundos para evitar "**rebotes**" (**bounce**), que são leituras múltiplas causadas pela vibração dos contatos mecânicos do botão.

Retorno do índice

```
    return i;
}
}
}
```

- Assim que um botão for **pressionado e solto**, retorna o índice i correspondente no vetor BOTOES[].

Exemplo de uso prático

Se o jogador pressionar o botão azul (ligado ao pino 5), digitalRead(BOTOES[3]) detecta o LOW, e a função retorna 3.

Esse valor é usado para comparar com a posição correspondente do vetor sequencia[], dentro da função lerResposta().

Conjunto de boas práticas envolvidas

- **Verificação contínua** até detectar uma ação.
- **Uso de pull-up interno do Arduino** (INPUT_PULLUP).
- **Leitura confiável** com espera de liberação do botão.

- **Filtro contra ruído (debounce)** manual com delay(40).

Conclusão

A função aguardaBotao() é **crítica para a interatividade do jogo**, pois:

- Garante que cada entrada do jogador seja corretamente identificada.
- Minimiza erros de leitura.
- Faz o sistema esperar **ativamente por uma ação humana**, o que é ideal em jogos com resposta em tempo real.

Função: void mostrarResultado(bool ok)

Objetivo

A função mostrarResultado(bool ok) tem como objetivo **exibir o resultado final da rodada** do jogo da memória no display LCD e emitir um **alerta sonoro com o buzzer**. Ela atua diretamente sobre o feedback ao jogador, reforçando o acerto ou erro de forma visual e auditiva.

```
void mostrarResultado(bool ok) {  
    lcd.clear();  
    if (ok) {  
        lcd.setCursor(0,0); lcd.print("PARABENS JOGADOR");  
        lcd.setCursor(0,1); lcd.print("VOCE ACERTOU    ");  
    } else {  
        lcd.setCursor(0,0); lcd.print("QUE PENA      ");  
        lcd.setCursor(0,1); lcd.print("VOCE SE ENGANOU");  
    }  
    beep(3, 120, 120);  
    delay(2500);  
}
```

Assinatura da Função

```
void mostrarResultado(bool ok) {
```

- Recebe um único argumento do tipo **booleano** chamado ok.
- Se ok == true, o jogador acertou a sequência.
- Se ok == false, o jogador errou.

Etapas da Função

1. Limpa o display LCD

```
lcd.clear();
```

- Apaga qualquer texto anteriormente exibido.
- Prepara a tela para uma nova mensagem.

2. Exibe a mensagem de acordo com o resultado

```
if (ok) {  
    lcd.setCursor(0,0); lcd.print("PARABENS JOGADOR");  
    lcd.setCursor(0,1); lcd.print("VOCE ACERTOU      ");  
} else {  
    lcd.setCursor(0,0); lcd.print("QUE PENA          ");  
    lcd.setCursor(0,1); lcd.print("VOCE SE ENGANOU");  
}
```

Se o jogador acertou (ok == true):

- Linha 1 do display: "PARABENS JOGADOR"
- inha 2 do display: "VOCE ACERTOU " (com espaços no fim para apagar texto residual da rodada anterior, se houver)

Se o jogador errou (ok == false):

- Linha 1 do display: "QUE PENA " (preenchido com espaços para manter o texto limpo)
- Linha 2 do display: "VOCE SE ENGANOU"

3. Emite 3 apitos com o buzzer

```
beep(3, 120, 120);
```

- Chama a função beep() para emitir **3 apitos sonoros**.
- Cada apito dura 120 milissegundos com intervalo de 120 milissegundos entre eles.
- Reforça o feedback: apitos podem simbolizar sucesso ou falha dependendo do contexto.

4. Aguarda antes de reiniciar

```
}  
delay(2500);
```

- Mantém a mensagem na tela por **2,5 segundos** para que o jogador possa ler e entender o resultado antes que o display seja atualizado para uma nova rodada.

Integração com o fluxo do jogo

Essa função é chamada após a comparação da resposta do jogador com a sequência gerada, logo após lerResposta():

```
bool venceu = lerResposta();  
mostrarResultado(venceu);
```

Resumo Didático

Ação	Quando ocorre
lcd.clear()	Limpa a tela para nova mensagem
Mensagem de acerto ou erro	De acordo com a variável ok
beep(3, 120, 120)	Emite som de feedback
delay(2500)	Espera 2,5 segundos antes do loop

Conclusão

A função mostrarResultado() é um bloco essencial para o encerramento de cada rodada, fornecendo feedback imediato, claro e didático para o jogador, tanto visual quanto sonoro. Ela ajuda a consolidar a experiência lúdica e educativa do jogo, criando um ciclo de aprendizado com reforço positivo ou negativo.

Checando fluxos possíveis:

- Vitória de rodadas: cada completada incrementa pont e continua. Pont então conta quantas rodadas concluiu, ou seja, quant-0 sequence = 1 etc. Se perder, pont já incrementou a última completada. E esse pont é exibido em "Resultado: X".
- Se perder na primeira sequência, pont será 0 (pont++ só acontece quando termina um round, ou seja, adicionado e repetiu).
- Faz sentido.
-

Tem apenas um pequeno bug possível: lcd.begin(16,2) sendo chamado a cada new loop top e não limpando se a scoreboard muito curta anteriormente. Mas eles limpam ao erro e no loop standby limpam no start. De toda forma, funciona.

5.3 Representação em Lógica de Blocos (Fluxograma)

Assim que o aluno liga a placa **Arduino Uno** (alimentada pelo conector P4 – fio **vermelho** no +9 V e fio **preto** no GND), o microcontrolador inicializa:

- **LCD 16 × 2 via I²C** — cabos **verde-claro** (SDA em A4) e **verde-escuro** (SCL em A5) transportam os dados; fios **vermelho** (+5 V) e **preto** (GND) energizam o módulo.
- **Quatro LEDs** posicionados na coluna esquerda da protoboard:
 - LED **vermelho** (pino 10, fio **vermelho**)
 - LED **amarelo** (pino 8, fio **amarelo**)
 - LED **verde** (pino 6, fio **verde**)
 - LED **azul** (pino 4, fio **azul**)
- **Cinco botões tácteis**:
 - Botão **branco** de Start/Reset (pino 12, fio **branco**)
 - Botões de resposta nas cores dos LEDs (11 – vermelho, 9 – amarelo, 7 – verde, 5 – azul).

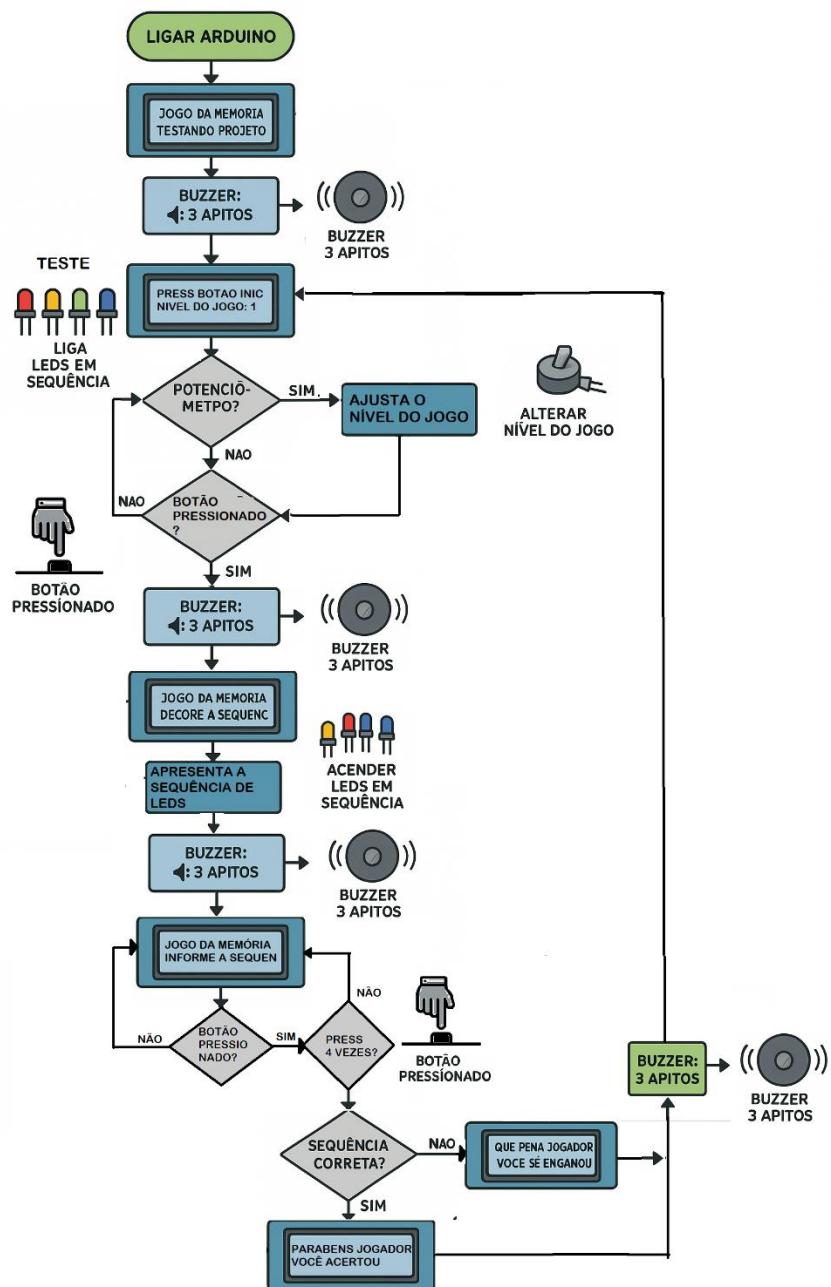
- **Buzzer piezoelétrico** (pino 3, fio **roxo**) para sinais sonoros.
- **Potenciômetro de 10 kΩ** (cursor em A0, fios **ciano** para cursor, **vermelho** para +5 V e **preto** para GND) – regula a velocidade / nível do jogo.
-

O jogo segue quatro macro-fases que se repetem em loop:

1. **Auto-teste e escolha do nível**
2. **Geração e exibição da sequência**
3. **Leitura da resposta do jogador**
4. **Feedback sonoro-visual e reinício**
- 5.

Cada ponto da lógica é representado visualmente no fluxograma a seguir.

Figura 45 - Fluxograma do Jogo da Memória



OVAL VERDE – “LIGAR ARDUINO”

- Fios de alimentação vermelhos e pretos já conectados.
- LCD mostra JOGO DA MEMORIA / TESTANDO PROJETO.
- LEDs piscam um de cada vez (vermelho → amarelo → verde → azul).
- Buzzer emite **3** bipes curtos (tom 1 kHz).

DECISÃO AMARELA – “Potenciômetro girado?”

- Cursor do potenciômetro (fio **ciano** em A0) lê 0-1023.
- Se girar, LCD atualiza NIVEL DO JOGO: N (1 a 5); seta circular volta ao losango.
- Se não girar, mantém nível atual (padrão = 1).

RETÂNGULO AZUL – “PRESS BOTAO INIC”

- Programa aguarda **botão branco** (pino 12) ir a LOW.
- Quando pressionado, buzzer faz **3** bipes → segue.

RETÂNGULO LARANJA – “DECORE A SEQUENCIA”

- Arduino gera array pseudo-aleatório de 5 valores (0-3).
- De acordo com o nível, define intervalo (2000 → 800 ms).
- Acende LEDs pela ordem; buzzer toca bipes individuais.

RETÂNGULO AZUL – “INFORME A SEQUENCIA”

- LCD mostra instrução; programa libera leitura dos **4 botões coloridos**.
- Cada pressão acende o LED correspondente e toca beep curto.

DECISÃO VERMELHA – “Sequência correta?”

- **Sim:** retângulo verde exibe PARABENS / VOCE ACERTOU, buzzer toca melodia rápida, seta curva retorna ao bloco “Potenciômetro girado?”.
- **Não:** retângulo vermelho exibe QUE PENA / VOCE SE ENGANOOU, buzzer faz 3 bipes graves, seta curva retorna também ao ajuste de nível.

5.4 Entendendo e Adaptando o Código

Agora que dissecamos o código, alguns pontos merecem atenção e possivelmente adaptações no nosso caso específico:

- **Uso do LCD via I2C:** O código de exemplo usava LiquidCrystal e manipulava pinos diretamente. No nosso hardware, estamos usando um módulo I2C para o LCD. Isso significa que:
 - Precisamos incluir a biblioteca LiquidCrystal_I2C.h (que talvez requeira instalação do pacote).
 - Iniciar o LCD de forma diferente: por exemplo LiquidCrystal_I2C lcd(0x27,16,2); lcd.init(); lcd.backlight();.
 - As chamadas lcd.print, lcd.setCursor, lcd.clear() continuam funcionando igual.
 - O lcd.begin(16,2) que repetem não seria necessário, mas podemos mantê-lo se init() já fez similar. Provavelmente substituímos todas instâncias de lcd.begin(16,2) por lcd.setCursor(0,0) ou remover.

- Isso é detalhe de implementação, mas conceptual não muda.
- **Ajustes de interface:**
A mensagem "Taca-lhe pau!!" é divertida mas um professor/pai poderia preferir outra frase para crianças se achar estranho (embora crianças achem graça desse meme). Poderia ser "Boa sorte!" ou "Memorize:" ou até mostrar o nível ("Nível 3" ou "Sequencia 3").
- **Velocidade do jogo:**
O tempo de acender LED e pausas definem a dificuldade. Eles usaram 500ms on, 200ms off entre sinais. A resposta do jogador espera tempo indefinido até apertar (não puseram limite de tempo, a menos que professor quisesse, mas normal do Simon original não tem time limit, só progressivamente se torna difícil). Poderíamos ajustar: se quiséssemos aumentar desafio, poderíamos diminuir os delays de exibição ou introduzir um timeout.
- **Debounce dos botões:**
Eles usaram uma estratégia de segurar até soltar, com while(button pressed). Isso e os delays implementados resolvem bounce e múltiplas contagens, simples e eficaz. Poderia ser refinado, mas para 6mm tactile buttons, isso basta.
- **Tamanho da sequência:**
Arranjo de 200 pode ser exagero (mas sem problema, ocupa 200 bytes). Nenhum 8-15 anos deve conseguir 200 de memória! O recorde do genuíno Simon (por humanos) é bem menor. 50 já é inumano. Mas deixaram 200 por garantia ou por conveniência.
- **Reiniciar ao final:**
Após erro, eles esperam 3s e loop repetirá. O LCD ainda terá "ERROU!!!!..." por um instante até ser sobreescrito por "Vamos jogar?" no próximo loop (porque eles clear no start). Isso fica ok.
- **Som:**
Usaram notas C (261.63 Hz), D (293.66 Hz), E (329.63 Hz), F (349.23 Hz) para os 4 color notes, e G# (415.30 Hz) para erro. Isso imita o Genius original? O Simon original tinha tonzinhos mais altos e diferentes (Simon's tones are 880Hz, 740Hz, 587Hz, 494Hz or similar - not sure exato, mas eles escolheram do escala musical, que fica bom pro ouvido). Tudo bem, as relativas do, re, mi, fa soam ascendentes. O som erro outro. Perfeito.
- **Analog vs Digital Input para botões:**
Eles optaram por analogRead. Poderíamos questionar: e se ruído analógico leve fizesse ele ler, digamos, 10 quando solto? Ele compara ==1023, então só reagirá se bem próximo de 1023, que de fato quando não pressionado raramente atinge saturação. Era mais garantido talvez usar digital + resistor

pull-down ou internal pull-up invertido. Mas isso complicaria o hardware com 4 resistores ou explicar INPUT_PULLUP invert logic. A analog hack é mais simples de montagem, e pragmática. Em teste prático, deve funcionar bem.

- **Memory footprint:**

O code run in Arduino UNO fine, not heavy.

5.5 Carregando e Testando o Código

Ao passar esse código (adaptado para I2C se necessário) para o Arduino:

- Mantenha o Arduino conectado via USB.
- Abra o Serial Monitor (mesmo não usamos serial prints no code, mas se quisesse debug, poderíamos add). Aqui, sem prints, não aparece nada no PC. Todo feedback é via LED, buzzer, LCD.
- Aperte o botão verde para iniciar: deve apagar standby, maybe scoreboard with "Pontuacao: 0" appears and sequence of 1 LED flash.
- Teste reagir pressionando o correspondente LED:
 - If correct, next round begins with score 1 and 2 lights sequence.
 - If wrong, all done sequence, error sound + "ERRO!!! Resultado: X".
- Check that LED, sound, sequence logic all align with your expectation:
 - E.g. If you purposely press wrong button at second step, does it immediate do error sequence? It should, because acertou =0 break out.

Possíveis ajustes após o teste inicial:

- Se o LCD não estiver exibindo nada, ajuste o potenciômetro de contraste no módulo até que o texto fique visível.
- Se os sons não estiverem audíveis: alguns buzzers têm volume baixo. Geralmente está tudo bem. Também pode-se tentar frequências de tom diferentes se algumas não forem audíveis, mas as atuais estão na faixa média, então devem funcionar.
- Se um botão específico não estiver funcionando, provavelmente é um problema de fiação (talvez a leitura analógica não esteja chegando a 1023 porque aquele botão não está corretamente conectado ao 5V quando pressionado? Verifique a fiação).
- Se todos ou vários botões falharem, verifique se os pinos analógicos estão lendo valores estáveis (talvez seja necessário um resistor pull-down, afinal? Mas geralmente, se estiverem flutuando, eles leem valores aleatórios entre ~0 e 20, o que não causaria falsos positivos, já que precisamos de 1023).
- O estado de espera ao pressionar o botão verde acende todos os quatro LEDs porque o código faz isso. Isso pode aumentar o consumo de corrente ($4 \times 15\text{mA} = 60\text{mA}$, o que é aceitável).
- O consumo total do circuito:
 - Com os 4 LEDs acesos: cerca de 60mA
 - Buzzer ativo: cerca de ??? talvez 30mA

- Backlight do LCD: cerca de 20mA
- Em funcionamento total: aproximadamente 100mA
- A porta USB pode fornecer essa corrente, e a bateria também (embora com duração limitada).
- Uso com bateria: teste após verificar tudo via USB. Em seguida, desconecte o USB e conecte a bateria. O sistema deve iniciar normalmente (LCD ligado etc.). Pressione o botão de início, etc. Se a bateria estiver nova, tudo deverá funcionar da mesma forma. Pode haver uma leve queda de tensão (de 9V para 5V via regulador), mas isso é aceitável.

5.6 Expandindo e Melhorando

Ao entender o código, você pode discutir ou pensar em expansões:

- **Modo "Dificuldade Progressiva":** Podia-se aumentar a velocidade de exibição (reduzir delays) conforme pontuação aumenta, tornando memorizar mais difícil.
- **Limite de Tempo para resposta:** Podia-se, por exemplo, se jogador demora mais que X sec para apertar, considerar erro. (Teria que monitorar no loop do input, ex: reset a timer after each button, if goes beyond threshold, break as fail).
- **Indicador visual de final ou recordes:** Podia-se guardar um highscore e mostrá-lo. Ou se completou 5 rodadas, flash something special.
- **Suporte a repetição ou correção de input:** O Simon original comparava a sequência só após o input completo, mas aqui estamos comparando a cada input (o que na prática é o mesmo, mas por ex. no Simon, se você pressiona alguma fora de sequência, ele para. Sim).
- **Mais jogadores ou competições:** Provavelmente fora do escopo, mas poder hooking to scoreboard etc.

Essas são ideias que se poderiam implementar discutindo com alunos que entendem o básico.

5.7 Conclusão da Programação

Programar um jogo clássico como Genius nos permite aplicar vários conceitos:

- Manipulação de arrays e geração aleatória,
- Lógica de controle com loops aninhados e condicionais,
- Interação com hardware (botões, LEDs, buzzer, LCD),
- Pensar em usabilidade (feedback imediato, pausas adequadas, reiniciar limpando estado).

A explicação acima tocou cada parte para que um aluno 8-15 possa seguir o raciocínio, principalmente se já introduzimos noções de variáveis, if, loops antes. Ver isso aplicado concretamente torna mais palpável.

No fim desta unidade, o estudante deve conseguir responder questões como:

- "Como o Arduino sabe se eu apertei a cor certa?" (resposta: comparando a posição na sequência com a cor esperada guardada no array).

- "Como a sequência cresce?" (resposta: cada rodada adiciona um número aleatório ao array).
- "O que acontece quando eu erro?" (resposta: o flag acertou muda, sai do loop e toca som de erro e mostra mensagem final).
- E assim por diante.

Nos exercícios a seguir, vamos consolidar esse entendimento, e preparar para, quem sabe, modificar ou criar novos desafios baseados nesse código.

Exercícios – Unidade 05

1. **Variáveis:** Qual o papel do array `luzes[]` no código? O que ele armazena e como é utilizado durante o jogo?

2. **Gerando Aleatórios:** Explique o que faz a linha `randomSeed(analogRead(0));` seguida de `random(1, 5)`. Por que isso é importante para o jogo?

3. **Fluxo do Jogo:** Descreva resumidamente o que acontece quando o jogador inicia uma nova rodada (desde adicionar uma cor até ele terminar de repetir a sequência).

4. **Comparação:** No código, por que usamos `if(luzes[aux] == 3)` dentro do bloco do botão amarelo, por exemplo? O que representam o número 3 e a variável aux ali?

5. **Feedback Sonoro:** Cada cor toca uma frequência diferente (262 Hz, 294 Hz, etc.). Por que você acha que escolheram frequências diferentes para cada LED? O que aconteceria se todas tocassem a mesma?

6. **Controle de Fluxo:** O código usa `while` para esperar o botão ser solto (`while(pg == 1023) { pg = analogRead(pushG); }`). Explique com suas palavras por que isso foi feito. O que poderia dar errado se não tivéssemos esse `while` e o jogador mantivesse o botão pressionado?

7. **Pontuação:** Em que momento do código a pontuação (pont) é incrementada? O que isso representa no jogo?

8. **Condição de Saída:** Identifique onde no código o loop principal do jogo (while(acertou == 1)) é quebrado. O que causa a saída desse loop e como o código reage a isso?

9. **Fluxograma:** Baseado na explicação, desenhe (em papel mesmo) um fluxograma simples do processo do jogo: Início -> esperar start -> loop de rodada (mostrar seq -> ler seq) -> fim -> reiniciar. (Não precisa entregar o desenho aqui, mas faça para você compreender).

10. **Modificação:** Se quiséssemos deixar o jogo mais fácil para iniciantes, qual parte do código poderíamos ajustar? (Dica: pense nos delays ou nas mensagens no LCD). Como isso afetaria a experiência do jogador?

UNIDADE 06: Funcionamento e Manual do Usuário

Parabéns! Chegamos à última etapa do nosso projeto: entender como **operar o jogo da memória pronto** e discutir orientações de uso, resoluções de problemas comuns e ideias para expandir o projeto. Nesta unidade, vamos elaborar um pequeno **manual do usuário**, ou seja, instruções simples para alguém (criança ou adulto) que for brincar com o jogo da memória que construímos. Também veremos qual o comportamento esperado do circuito em cada situação, e como diagnosticar e resolver situações comuns (por exemplo, "um LED não acende", ou "o botão não funciona", etc.).

Por fim, exploraremos sugestões de melhoria ou expansão do projeto: agora que você dominou o básico, que tal incrementar o jogo? Apresentaremos algumas ideias para inspirar você a continuar aprendendo e se divertindo com robótica e programação mesmo após finalizar este eBook.

6.1 Manual de Uso do Jogo da Memória

Ligando o jogo:

Para ligar o seu jogo da memória Arduino, basta conectar a bateria de 9V ao conector (ou conectar o cabo USB a uma fonte de 5V, como um carregador de celular ou porta do computador). O circuito não tem um botão de liga/desliga dedicado, então ele inicia assim que recebe energia.

Estado inicial:

Ao ligar, você verá os **4 LEDs coloridos acesos** simultaneamente e a tela LCD exibindo a mensagem "Vamos jogar?". Isso indica que o jogo está pronto para começar uma partida. O buzzer não emite som nesse momento.

Iniciando uma partida:

Pressione o **botão verde** (correspondente ao LED verde). Ele funciona como o "start". Assim que você apertá-lo:

- Os LEDs coloridos que estavam acesos se apagam.
- O LCD limpa a mensagem de espera.
- Após um breve instante, o jogo inicia de fato a sequência de memória.

Como jogar:

O jogo consiste em uma sequência cada vez maior de cores e sons que você deve repetir:

1. O Arduino vai piscar uma ou mais luzes em uma certa ordem, tocando um tom para cada cor (cada LED tem um som diferente). Preste muita atenção e **memorize a sequência** de cores exibida.

2. Assim que as luzes terminarem de piscar, **é sua vez**: você deve reproduzir a sequência na mesma ordem, pressionando os botões correspondentes às cores.
 - o Ao apertar cada botão, a luz daquela cor acenderá e o buzzer tocará o mesmo som da fase de demonstração, confirmando seu input.
 - o **Importante:** aperte os botões **um de cada vez**, aguardando cerca de meio segundo entre um e outro. Não tente apertar dois simultaneamente.
3. Se você reproduzir corretamente toda a sequência, **você ganha um ponto!** O LCD irá atualizar mostrando "Pontuacao: X" (sendo X sua nova pontuação). Então o jogo automaticamente passará para a **próxima rodada**, adicionando mais uma cor no final da sequência, tornando-a mais longa e desafiadora.
4. Agora você terá que assistir e memorizar uma sequência mais longa e repeti-la novamente. A cada rodada vencida, a sequência fica maior e sua pontuação aumenta.
5. Isso continua até que você cometa um erro ao reproduzir a sequência.

Final de jogo:

Se você apertar **algum botão errado**, ou seja, reproduzir a cor equivocada na ordem, imediatamente:

- O jogo soará um som de "**erro**" prolongado (um buzzer contínuo diferente dos tons das cores).
- O LCD exibirá "**ERROU!!!**" na primeira linha e "**Resultado: Y**" na segunda linha, onde Y é sua pontuação final (quantas sequências você acertou naquela partida).
- Todos os LEDs permanecem apagados durante o som de erro.
- Após aproximadamente 3 segundos, o jogo reinicia para o estado inicial: os LEDs acendem novamente todos juntos e aparece "**Vamos jogar?**" na tela, pronto para você tentar de novo e talvez bater seu recorde!

Exemplo de jogo:

Suponha que na primeira rodada a cor sorteada foi **Vermelho**. O Arduino pisca o LED vermelho e toca o som correspondente. Você então deve apertar o botão vermelho. Se fizer certo, pontuação vai a 1. Na rodada 2, digamos que a nova sequência sorteada seja **Vermelho -> Azul**. O Arduino pisca vermelho (som A) depois azul (som B). Agora você pressiona em ordem: botão vermelho, depois botão azul. Acertou de novo! Pontuação 2. Na rodada 3, sequência pode ser **Vermelho -> Azul -> Verde**, e assim por diante... Se em algum momento você pressionar um botão fora da ordem (por exemplo, apertar verde quando deveria ser amarelo), o jogo detecta o erro e termina.

Jogando novamente:

Não é necessário desligar e ligar para jogar de novo. Após um fim de jogo (quando aparece "**ERROU!!! Resultado: ...**"), basta pressionar novamente o botão verde de Start. A mensagem "**Vamos jogar?**" já deve estar na tela (ou aparecerá logo após), e o sistema iniciará uma nova sequência do zero ao você apertar Start. Tente superar sua pontuação anterior!

6.2 Comportamento Esperado do Circuito

- **LEDs em espera:**

4 LEDs acesos fixo significam "jogo parado/esperando início". Eles não piscam ou mudam até alguém iniciar.

- **Durante sequência do Arduino:**

Os LEDs piscam um de cada vez, de forma clara. Nunca dois simultâneos. Há uma breve pausa escura entre flashes. O buzzer deve emitir um tom breve exatamente quando a luz acende. O LCD exibe a pontuação atual e uma mensagem motivadora ("Taca-lhe pau!!" ou algo configurado).

- **Durante resposta do jogador:**

Quando você pressiona um botão:

- O LED correspondente acende imediatamente e apaga após cerca de meio segundo.
- O buzzer toca o tom correspondente durante esse meio segundo.
- Os outros LEDs permanecem apagados enquanto você não os aperta. Ou seja, só acende a cor que você apertou, servindo de feedback.
- O LCD permanece mostrando a pontuação durante sua resposta (ele não muda a cada botão apertado, apenas é atualizado no início de cada rodada).

- **Após sequência correta:**

Se você concluir a série certa, o Arduino não faz nenhum som especial (além do beep do último botão). Ele simplesmente inicia a próxima rodada depois de um curto intervalo. Você notará a próxima rodada porque:

- O LCD continuará mostrando "Pontuacao: [atual]" (que já havia sido atualizada no fim da rodada anterior) e possivelmente a mesma frase motivacional na linha de cima.
- Após ~1 segundo, começará a piscar a nova sequência (que será tudo que piscou antes + uma cor nova no final). Este 1 segundo de espera é útil, mas dá pra perceber um respiro entre rodadas.

- **Após erro:**

- LED nenhum fica aceso durante o som de erro (todos apagados).
- Som de erro: é longo (~3 segundos) e de tom diferente (mais grave ou estranho) comparado aos beeps normais. Isso deixa claro que algo deu errado.
- LCD exibe "ERROU!!!!" bem grande e resultado final.
- Depois desses 3 segundos, os 4 LEDs acendem e "Vamos jogar?" volta à tela, sem nenhum som (a não ser que decida inserir talvez um jingle de reset, mas nosso código não tem).

- **Se nenhum botão for pressionado na vez do jogador:**

O jogo essencialmente fica aguardando. Não há limite de tempo implementado, então se o jogador demorar 10 segundos ou 10 minutos para apertar, o estado ficará parado esperando input. Os LEDs permanecerão apagados nesse meio-tempo (pois a sequência já foi mostrada e agora está no loop de input esperando). O LCD continua mostrando a pontuação. Isso é normal. Se quiséssemos, poderíamos impor um timer, mas por padrão não há. Então, jogadores podem pensar calmamente antes de responder (o que ajuda a memorizar, porém o desafio usual do Simon é de memória, não de tempo).

- **Reset do Arduino:**

Se por algum motivo você resetar ou desligar/ligar o Arduino, ele vai reiniciar no estado inicial (LEDs de espera, LCD "Vamos jogar?"). A pontuação é zerada porque não fica armazenada após desligar.

6.3 Situações Comuns e Soluções

Mesmo com o circuito e código funcionando, podem surgir dúvidas ou pequenos problemas ao operar. Vamos listar alguns e suas possíveis causas/soluções:

- **"Um dos LEDs não acende quando deveria":**

- *Durante sequência do Arduino:* Se, por exemplo, a sequência tem azul mas o LED azul não pisca, pode ser que o LED azul esteja invertido ou mal conectado, ou o fio do pino 13 solto. Cheque a fiação do LED correspondente e suas conexões.
- *Durante aperto do jogador:* Se você aperta o botão e a respectiva luz não acende, pode ser problema no botão não sendo reconhecido (veja próxima situação). Em geral, LED não acender quando deveria indica problema de hardware (conexão) mais do que software, já que o código trata todos igualmente.

- **"Apertar um botão não está sendo reconhecido":**

(você pressiona mas nada acontece, o jogo não avança)

- Certifique-se de estar pressionando completamente o botão (botões tácteis pequenos fazem "clic" quando acionados corretamente).
- Pode ser que o botão esteja mal posicionado (pinos numa mesma linha). Verifique a montagem do botão.
- Teste o botão isoladamente: por exemplo, no estado de espera (LEDs acesos), aperte o botão que suspeita. Em nosso design, nada deveria acontecer visualmente nesse estágio, mas se o LED de Start (verde) não inicia, sabemos que ele não estava sendo lido.
- Se for um dos botões de sequência, talvez o jogador esteja apertando errado fora de hora. Lembre: Só pressione depois que o Arduino terminar de piscar a sequência inteira.
- *Solução:* Revisar conexões do botão ao Arduino (fio no pino correto? contato firme?), e se necessário, adicionar um resistor de pull-down (por via das dúvidas) no pino analógico para GND ($10k \sim 100k\Omega$) para

estabilizar. Geralmente não precisa, mas é uma opção se ruído for problema.

- "**O LCD não mostra nada (ou mostra caracteres estranhos)**":
 - Ajuste o potenciômetro de contraste no módulo I2C do LCD. Gire devagar até ver nitidamente os caracteres.
 - Se ele mostra símbolos incorretos, pode ser inicialização errada - verifique se lcd.begin ou lcd.init foi chamado. Mas isso entra mais em debug de código. Em uso normal, deve mostrar texto legível se contraste ok.
 - Às vezes no primeiro segundo ao ligar pode aparecer uns caracteres malucos, mas logo lcd.begin limpa. Isso é normal.
- "**O som do buzzer está muito baixo ou ausente**":
 - Lembre que nosso buzzer não é muito alto. Em ambiente silencioso, deve dar pra ouvir bem. Se houver muito ruído ambiente, pode parecer baixo.
 - Certifique-se que o buzzer está conectado no pino certo e orientado (se for ativo, polaridade; se passivo, tanto faz polaridade).
 - Se realmente não sai som, veja se o LED correspondente acende (se LED acende mas sem som, talvez buzzer problema). Tente ouvir bem próximo.
 - Poderia testar buzzer isolado conectando por 1 seg entre 5V e GND (se ativo, vai apitar constante; se passivo, isso não faz som audível).
 - Se for um buzzer passivo e não ouve nada, será que tone() não funcionando? Deve, mas se fosse active buzzer, tone() curtos vão fazê-lo tic, possivelmente inaudível.
 - **Solução:** No pior caso, troque o buzzer (se tiver outro) ou use fones de ouvido no pino (melhor não, não arrisque ouvia direct).
 - Mas normalmente, está ok.
- "**O jogo não reinicia após erro**":
 - Após "ERROU" e som, ele deve voltar a espera. Se não acendeu LED de espera e "Vamos jogar?" sumiu, talvez ele travou. Pode ser ruído ou bug. Em nosso código, loop() repete e retoma.
 - Tente apertar reset no Arduino ou desligar/ligar. Isso sempre recomeça.
 - Esse projeto é robusto a travamentos, mas se loop não repete, suspeite de travamento possivelmente se tone() ou lcd não liberou recursos (mas deveria).
 - Sem alterações, ele se reinicia bem.
- "**Quero desligar para guardar, como faço?**":
 - Simplesmente desconecte a bateria 9V. O circuito descarrega e apaga. Se alimentado via USB, desconecte o cabo USB. Se quiser um botão on/off, pode ser adicionado entre bateria e Arduino para conveniência.

- **"A bateria acaba muito rápido":**
 - Uma bateria 9V alcalina pode fornecer algumas horas de jogo contínuo (talvez 2 a 5 horas, dependendo). Se usar intensivamente, leve em conta que 4 LEDs e buzzer consomem consideravelmente.
 - Se notar os LED ficando fracos ou falhas estranhas no LCD (caracteres sumindo), a bateria pode estar fraca (<7V). Solução: trocar a bateria. Ou usar fonte USB se disponível.
 - Alternativamente, use pilhas AA (6x1.5V =9V) com maior capacidade ou uma bateria recarregável se tiver circuitos adequados.
- **"Quero alterar a dificuldade":**
 - Se o jogo está muito fácil/difícil, você pode ajustá-lo:
 - Mais fácil: pausar mais entre as cores (aumentar delays), ou repetir a sequência duas vezes antes de esperar resposta (exibir duas vezes).
 - Mais difícil: diminuir esses delays, ou introduzir um limite de tempo para apertar, ou começar já com duas cores na sequência inicial.
 - Essas alterações exigem modificar o código e reprogramar o Arduino, o que é encorajado se você já entende bem. Tente fazer pequenos passos para não bagunçar.
- **"Recorde/High Score":**

O jogo não guarda recorde entre resets, mas você pode anotar manualmente sua maior pontuação e tentar vencê-la. Se quiser, pode programar o Arduino para exibir um "Recorde: Z" no LCD, mas exigiria usar EEPROM ou manter valor até desligado (um recurso avançado). Fora do escopo deste eBook, mas possível de implementar.

6.4 Sugestões de Melhoria ou Expansão do Projeto

Agora que você tem um jogo da memória funcional, as possibilidades de expansão são muitas. Aqui vão algumas ideias para inspirar:

- **Modo Versus/Multijogador:**

Adaptar o jogo para 2 jogadores ou mais turnos. Por exemplo, cada jogador tenta superar a sequência e passa ao próximo, quem errar sai. Isso precisaria de indicações de qual jogador deve jogar (talvez mostrar no LCD "Jogador 1" / "Jogador 2"). Poderia incluir um botão extra para "novo jogador" ou utilizar os mesmos de cor para input do nome do jogador.
- **Aumento de Velocidade:**

Programar para que a cada rodada a velocidade de piscar as luzes aumente, tornando mais desafiador quanto mais longe se chega (o Simon original não faz isso, mas seria uma variante bacana).
- **Adicionar Feedback Visual no LCD:**

Por exemplo, quando acertar uma rodada, mostrar "Boa!" ou algum emoji/símbolo feliz brevemente, e quando errar, além de "ERROU!!!", talvez carinhas tristes. O LCD 16x2 permite criar caracteres customizados (como smileys) – um tópico avançado, mas interessante.

- **Som de Vitória:**

Se o jogador alcançar um certo número de pontos (ex: 10), tocar uma musiquinha especial de parabéns e piscar todos LEDs festivamente. Pode ser um objetivo/metas.

- **Registrar High Score:**

Como mencionado, usar a memória EEPROM do Arduino para salvar o maior recorde atingido e exibi-lo, assim cada novo jogador tenta bater esse recorde. Isso envolve conceitos extras de armazenamento permanente.

- **Modos de Jogo Diferentes:**

Por exemplo, um modo "contrário" onde o jogador deve repetir a sequência de trás para frente. Ou um modo "só som" em que os LEDs não acendem, apenas beep, forçando a pessoa a reconhecer pela nota musical (difícil!).

- **Mais Botões/Cores:**

Com hardware adicional, poderíamos expandir para um Genius de 6 cores, por exemplo. O código ficaria maior, mas escalaria de forma semelhante (ex: random(1,7) e mais ifs ou um array de pinos).

- **Placar e Timer via Serial/Web:**

Se conectar o Arduino ao PC enquanto joga, ele poderia mandar dados via Serial (USB) para um programa registrar pontuações, ou até usar um módulo Bluetooth para scoreboard via celular. Bem além do básico, mas abre portas para IoT.

- **Caixa e Design:**

Fora da programação/eletrônica, você pode construir uma **carcaça** para seu jogo! Um pedaço de papelão ou acrílico para fixar os LEDs e botões nas posições arranjadas como um brinquedo real (talvez redondo com cada cor em um quadrante, imitando o clássico Genius). Personalize com adesivos, pinte as cores, etc. Só cuidado para não desconectar fios ao manusear – fixe bem a protoboard ou solde tudo em uma placa mais compacta se for avançar nesse aspecto.

Lembre-se: cada melhoria grande pode requerer aprender novos conceitos de programação e eletrônica. Isso é ótimo! Significa que esse projeto tem espaço para evoluir junto com suas habilidades. Mesmo que você não implemente todas as ideias, pensar sobre elas já exercita sua criatividade e reforça o entendimento do que fez até agora.

6.5 Encerrando o Projeto

Ao concluir este eBook, esperamos que você tenha alcançado as seguintes coisas:

- Montou um circuito Arduino do zero, aprendendo sobre componentes básicos (LEDs, resistores, botões, buzzer, LCD).
- Desenvolveu (ou pelo menos entendeu profundamente) um código completo de jogo, aplicando lógica de programação, controle de fluxo e manipulação de hardware com software.
- Percebeu na prática as competências de raciocínio lógico, resolução de problemas e interdisciplinaridade (trabalhamos com conceitos de física, matemática e engenharia, tudo enquanto nos divertíamos com um jogo).
- Ganhou confiança para explorar modificações e projetos futuros: se você conseguiu fazer um "Genius" eletrônico, quem sabe qual será seu próximo invento?

No próximo e último trecho deste eBook (Conclusão), vamos resumir o que você aprendeu e incentivá-lo a continuar nessa jornada tecnológica. A robótica e programação são áreas ilimitadas – e você deu um grande passo ao completar este projeto.

Exercícios – Unidade 06

1. **Iniciando o Jogo:** O que você deve observar no circuito para saber que o jogo está pronto para começar? E o que precisa fazer para iniciar uma partida?

2. **Sequência de Jogo:** Descreva o que ocorre quando o Arduino está mostrando a sequência. O que você, como jogador, deve (ou não deve) fazer nesse momento?

3. **Resposta do Jogador:** Se a sequência for Verde -> Vermelho -> Azul, quais botões e em que ordem você deve apertar? O que acontece se você apertar um botão errado no meio?

4. **Indicadores de Erro:** Liste dois sinais que o projeto dá quando você erra a sequência.

5. **Reiniciando:** Depois de um erro e do fim de jogo, como fazer para jogar novamente? É preciso desligar o Arduino?

6. **Problemas e Soluções:** Se você perceber que ao apertar o botão amarelo nada acontece, qual pode ser a causa e como poderia solucionar?

7. **Manutenção:** Como você preservaria a bateria do projeto quando não estiver jogando? O que pode indicar que a bateria está fraca?

8. **Expansão Criativa:** Qual das sugestões de melhoria (ou outra ideia sua) você achou mais interessante? Como você a implementaria de maneira geral (não precisa código, mas o conceito)?

9. **Uso de Competências:** Cite uma habilidade ou conhecimento de outra disciplina escolar que você acabou utilizando ou desenvolvendo ao fazer este projeto (por exemplo: matemática ao calcular resistência, música ao lidar com sons, etc.).

10. **Experiência Pessoal:** Como você se sentiu ao ver o projeto funcionando? Qual foi a parte mais desafiadora e qual foi a mais divertida ao longo do processo?

CONCLUSÃO

Chegamos ao fim do nosso eBook educacional de robótica com Arduino! Esperamos que você tenha aproveitado essa jornada de aprendizado "mão na massa" e se orgulhe do que conquistou. Vamos recapitular as **aprendizagens e competências desenvolvidas** ao longo do projeto, e encerrar com palavras de incentivo para que você continue se aventurando no mundo da robótica e programação.

Resumo do Projeto:

Você construiu, do zero, um jogo eletrônico de memória (estilo Genius/Simon) usando um microcontrolador Arduino. Desde a **montagem do hardware** – conectando LEDs, resistores, botões, buzzer e display LCD – até a **programação do software** – escrevendo (ou entendendo profundamente) um código que gera sequências aleatórias e interage com o jogador –, você percorreu todas as etapas de desenvolvimento de um sistema eletrônico completo. Isso não é pouca coisa!

Ao longo do caminho, reforçamos vários conceitos:

- **Eletrônica básica:** tensão, corrente, resistência (Lei de Ohm), montagem em protoboard, uso de componentes como LEDs (incluindo polaridade), resistores (e cálculo de valores), botões (e configurações de entrada), buzzer (som) e display LCD (com comunicação I2C). Você viu esses elementos isoladamente e depois integrados no projeto final.
- **Programação em Arduino (C/C++):** estrutura de um programa (funções setup() e loop()), declarações de variáveis e constantes, uso de bibliotecas (LiquidCrystal), funções para entradas e saídas (pinMode, digitalWrite, analogRead, tone, etc.), lógica condicional (if/else) e loops (for, while). Mais importante, aprendeu a pensar logicamente para **resolver um problema** (no caso, replicar a lógica de um jogo de memória).
- **Depuração e melhoria contínua:** viu que erros podem acontecer (um LED que não acende, um fio solto, um bug no código) e aprendeu a abordá-los sistematicamente – verificando conexões, usando o serial monitor para debug de código, ajustando componentes (como contraste do LCD) e assim por diante. Essa habilidade de identificar e solucionar problemas é valiosa não só em robótica, mas em qualquer projeto ou situação da vida.
- **Interdisciplinaridade:** embora fosse um projeto de robótica, envolveu matemática (contagem de pontuação, uso de valores numéricos de entradas, frequência de som que se relaciona com escalas musicais), conceitos de física (circuitos elétricos, som – frequências em Hz, tempo – ms nos delays), arte e design (escolha de cores e sons, layout do dispositivo, criatividade nas mensagens de

feedback), e até habilidades linguísticas e de comunicação (lendo e seguindo instruções, escrevendo pequenos relatórios ou compartilhando resultados com colegas).

As **competências gerais** que a BNCC e as DCNs propõem também foram vivenciadas na prática:

- **Pensamento científico e crítico:**
você formulou hipóteses ("Por que esse LED não acende? O que acontece se eu mudar tal linha do código?") e testou na prática, analisando resultados e tirando conclusões. Isso é método científico puro!
- **Cultura digital:**
ao programar um dispositivo físico e entender seu funcionamento, você deixou de ser apenas consumidor para se tornar **criador de tecnologia**, exatamente como preconiza a competência 5 da BNCC. Agora, quando você ver um brinquedo eletrônico ou um semáforo de pedestres piscando, terá uma noção bem melhor do que pode estar acontecendo "por dentro".
- **Trabalho em equipe e comunicação:**
se você realizou este projeto em grupo ou com auxílio de um professor/pai, provavelmente discutiu ideias, explicou suas dúvidas, ouviu sugestões – enfim, **colaborou** para atingir um objetivo comum. Mesmo sozinho, ao ler este eBook, você se comunicou conosco através do texto e aplicou instruções, o que exige interpretação e foco.
- **Criatividade e autonomia:**
montar e programar envolve muito de tentativa e erro e ajustes pessoais. Talvez você tenha mudado a mensagem no LCD para algo divertido, ou decidiu usar outros sons – isso já é seu toque criativo. E esperamos que tenha ganhado **autoconfiança** para explorar ainda mais sem medo. Onde antes você via um emaranhado de fios e código, agora você enxerga possibilidades e sente: "Eu sou capaz de fazer isso!"

Próximos Passos:

Este projeto é apenas o começo. Considere:

- Aprender sobre outros sensores e atuadores: que tal adicionar um sensor de luz para o jogo só funcionar no escuro? Ou usar LEDs RGB multi-cores para novas cores?
- Mergulhar em programação mais avançada: por exemplo, aprender a usar funções personalizadas, estruturas de dados diferentes, ou até migrar para uma plataforma de blocos (como Scratch for Arduino) se quiser visualizar de outra forma.
- Participar de comunidades: existem clubes de robótica, feiras de ciências, fóruns online (como o Arduino Community) onde você pode mostrar seu projeto, ver projetos de outros e trocar conhecimento. Lembre-se, você agora faz parte da comunidade **maker** – pessoas que adoram criar e compartilhar invenções.

- Projeto integrador na escola: que tal apresentar esse jogo na aula de matemática ou física para ilustrar conceitos? Ou mesmo propor um desafio para colegas jogarem, estimulando lógica e memória deles?
- Continuar estudando: se isso te apaixonou, há muito a aprender – desde eletrônica mais profunda (entender transistores, quem sabe construir um circuito impresso para seu jogo) até programação de alto nível (talvez criar um jogo parecido no computador, ou um app móvel). O importante é manter a chama da curiosidade acesa.

Agradecimento e Encerramento:

Parabenizamos você por ter chegado até o fim. Não importa se você teve muita ajuda ou fez tudo sozinho – o que vale é que você persistiu e concretizou um projeto. Lembre-se: **errar faz parte** do processo de aprendizagem. Cada erro é uma oportunidade de aprender algo novo. Na robótica e na programação, a frase "não funcionou... ainda" é mais correta do que "não funcionou e pronto" – porque com paciência, eventualmente, funciona!

Esperamos que a experiência tenha sido divertida e enriquecedora. Que este seja o primeiro de muitos projetos. Continue explorando, programando, montando, desmontando (com cuidado!) e imaginando. O futuro precisa de inventores e solucionadores de problemas como você.

Como dizemos no mundo maker: "**Mão na massa e bora criar!**" Muito obrigado por participar deste eBook e... **até a próxima invenção!**

REFERÊNCIAS

- Papert, Seymour. *A Máquina das Crianças: repensando a escola na era da informática*. Porto Alegre: Artes Médicas, 1994. (Conceitos de construcionismo e aprendizagem por projetos).
- Brasil. Ministério da Educação. **Base Nacional Comum Curricular (BNCC)** – Educação Básica, 2017. Disponível em: basenacionalcomum.mec.gov.br. (Competências gerais e integração de tecnologia na educação).
- Tutorial Blog Eletrogate. "**Genius – Jogo da Memória no Arduino**". Eletrogate, 06/10/2020. (Projeto semelhante usado como referência de código e montagem).
- Documentação Oficial Arduino. "Arduino Uno Rev3 – Pinout" e "Arduino Language Reference". Disponível em arduino.cc. (Referência para uso de pinMode, digitalWrite, analogRead, tone, random etc.).
- Khan Academy (Pt-Br). **Artigo: Divisor de tensão**. (Explicação simples sobre como resistores em série dividem tensão).
- Mundo Educação (UOL). "*Primeira Lei de Ohm: o que diz, fórmula, cálculo*" (Conceitos de $U=R \cdot I$ e relação entre grandezas elétricas).
- Educacional (Blog). "*Robótica educacional: tudo que você precisa saber sobre a metodologia*". Educacional, 20/02/2025. (Discute vantagens da robótica na educação, interdisciplinaridade e competências desenvolvidas).
- Projeto Arduino no GitHub. "**Simon Says Game**" – Exemplo de implementação comunitária (inspiração para variações de código, sons e dificuldades).
- **Datasheet ATMega328P** (Microcontrolador do Arduino Uno). Microchip. (Para aprofundamento em portas ADC, timers usados em tone(), etc., conteúdo avançado além do eBook).
- Sandro Mesquita – Instagram [@profsandromesquita](https://www.instagram.com/@profsandromesquita) e site www.profsandromesquita.com. (Para contato, novidades em projetos de robótica e iniciativas educacionais em Fortaleza-CE).

(As referências acima englobam fontes de conceitos teóricos, inspiração de projetos e documentação utilizada na elaboração deste eBook.)