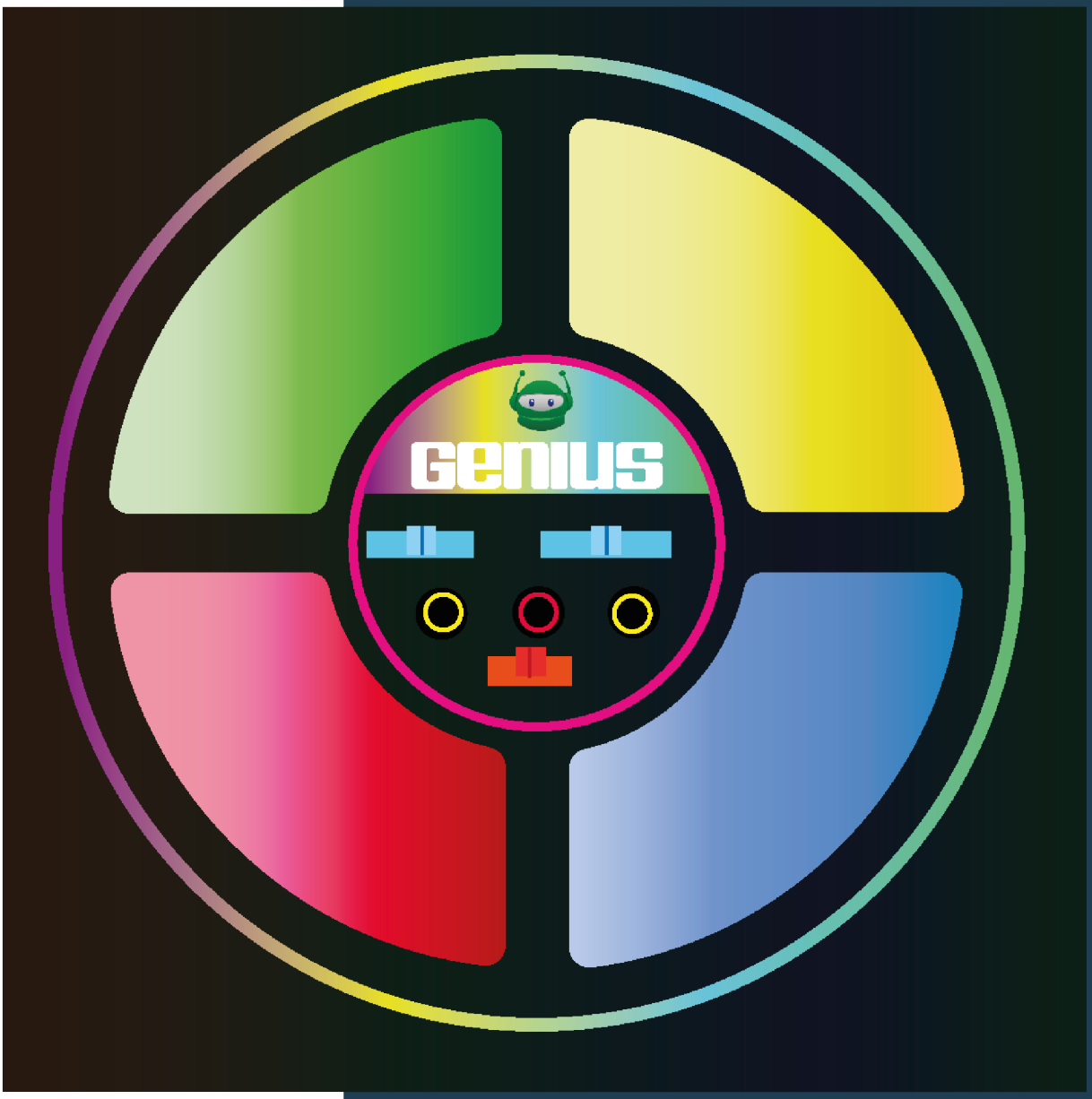
Robótica Educacional

Módulo 3



Aula

24

Genius - II

Diretoria de Tecnologia e Inovação

PARANÁ
GOVERNO DO ESTADO
SECRETARIA DA EDUCAÇÃO

GOVERNADOR DO ESTADO DO PARANÁ

Carlos Massa Ratinho Júnior

SECRETÁRIO DE ESTADO DA EDUCAÇÃO

Roni Miranda Vieira

DIRETOR DE TECNOLOGIA E INOVAÇÃO

Claudio Aparecido de Oliveira

COORDENADOR DE TECNOLOGIAS EDUCACIONAIS

Marcelo Gasparin

Produção de Conteúdo

Cleiton Rosa

Darice Alessandra Deckmann Zanardini

Validação de Conteúdo

Cleiton Rosa

Revisão Textual

Kellen Pricila dos Santos Cochinski

Projeto Gráfico e Diagramação

Edna do Rocio Becker

2024

Sumário

Introdução 2

Objetivos desta aula 3

Roteiro da aula 4

1. Contextualização 4

2. Programação - final 5

3. Feedback e finalização 15

Referências 16

Aula 24 Genius - II

Introdução

Na última aula, conhecemos o clássico jogo Simon, mais chamado no Brasil por Genius, e demos os primeiros passos para programar nossa própria versão! Iniciamos nossa versão para Arduino com as definições, programando seus ajustes para o começo da partida e o comportamento de LEDs e buzzer, desenvolvendo partes do jogo que envolvem a lógica de sequência até o feedback sonoro e visual. Ao final da aula anterior, sugerimos um teste para checarmos como ficou a animação de abertura do jogo pelo protótipo Aula 23 – Genius Parte I, disponível no simulador Wokwi.

Na aula de hoje, avançaremos para a finalização do nosso projeto com a conclusão da programação. Assim, nosso Genius ficará completo e as habilidades dos jogadores serão desafiadas em partidas incríveis e desafiadoras de memorização de cores e sons. Preparado?

Aula 24 Genius - II

Objetivos desta aula

- Finalizar a programação do jogo Genius com base nos componentes utilizados em sua montagem;
- Complementar a programação com foco na complexidade e proposta do jogo.

Lista de materiais

- Notebook ou computador.

Aula 24 Genius - II

Roteiro da aula

1. Contextualização

Com o objetivo de repetir uma sequência de luzes e sons sem errar, Simon – ou Genius – é um jogo com interface simples que se tornou um sucesso, desafiando crianças, jovens e adultos desde seu lançamento, na década de 70.

O legal do jogo é que a cada rodada a dificuldade aumenta, o que estimula cada um a ultrapassar seus recordes e ampliar a habilidade de repetir a sequência de acionamento de cores. E, assim como o próprio jogo, o processo de programá-lo também nos desafia para irmos além!

Figura 1 - Jogo Simon (Genius, no Brasil)

Fonte: Wikimedia Commons.

Aula 24 Genius - II

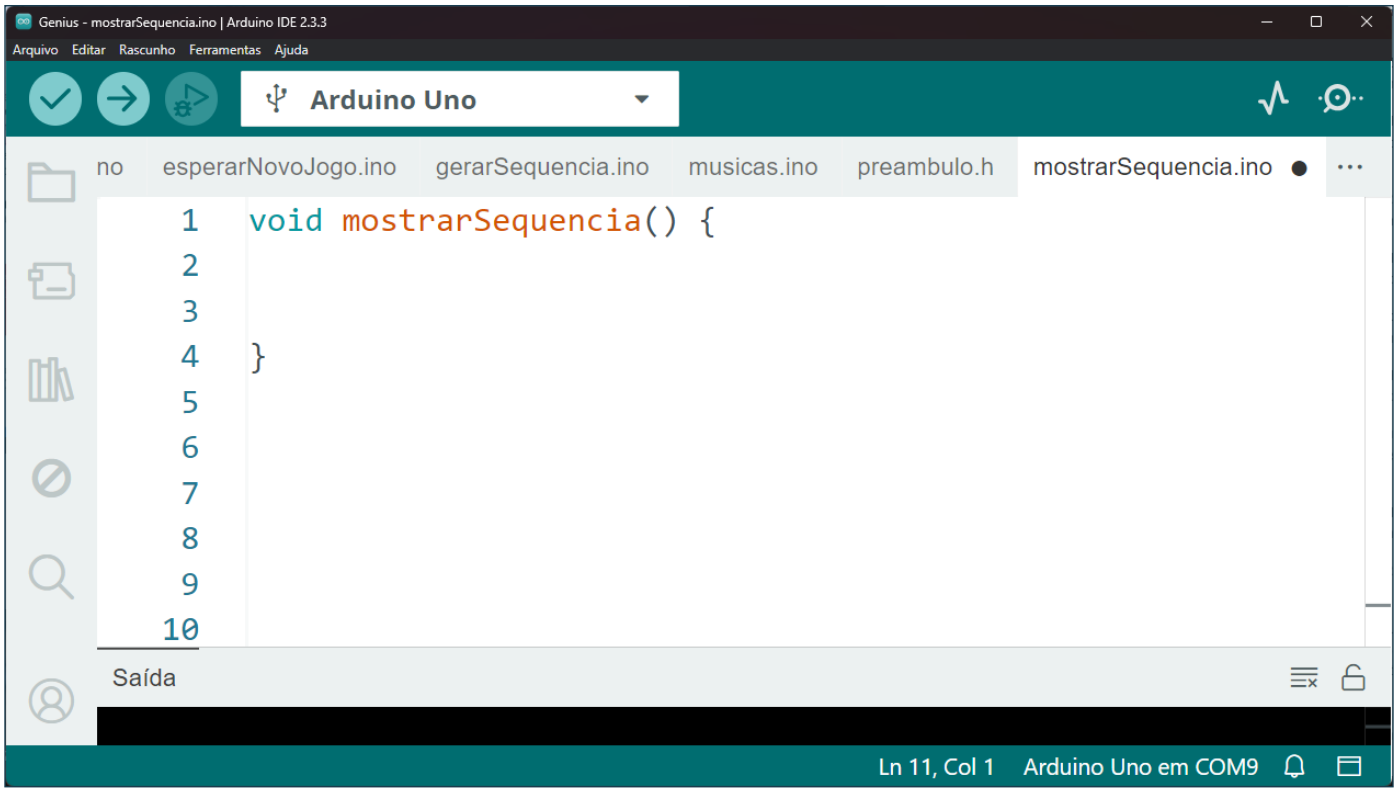
2. Programação - final

Antes de seguirmos para a montagem física do protótipo, finalizaremos a programação! Na última aula, concluímos as etapas iniciais e partiremos da programação da Aula 23 - Genius [Parte I] para dar sequência à conclusão do Genius para Arduino.

Continuando o desenvolvimento de cada parte do jogo, vamos avançar na programação do Genius. Abra o sketch salvo com a programação inicial do jogo para trabalharmos no **void loop()** com novas funções, as quais também criaremos em abas específicas para facilitar toda a organização do nosso jogo. Vamos lá?

Seguindo com a programação iniciada na Aula 23 - Genius [Parte I], adicione uma nova aba para criarmos a função mostrarSequencia().

Figura 2 - Criação da aba mostrarSequencia.ino para a função **void mostrarSequencia()**



Fonte: Arduino IDE.

Aula 24 Genius - II

A função **mostrarSequencia()** é essencial no Genius para apresentar a sequência de LEDs ao jogador, permitindo que ele memorize a ordem correta antes de tentar reproduzi-la.

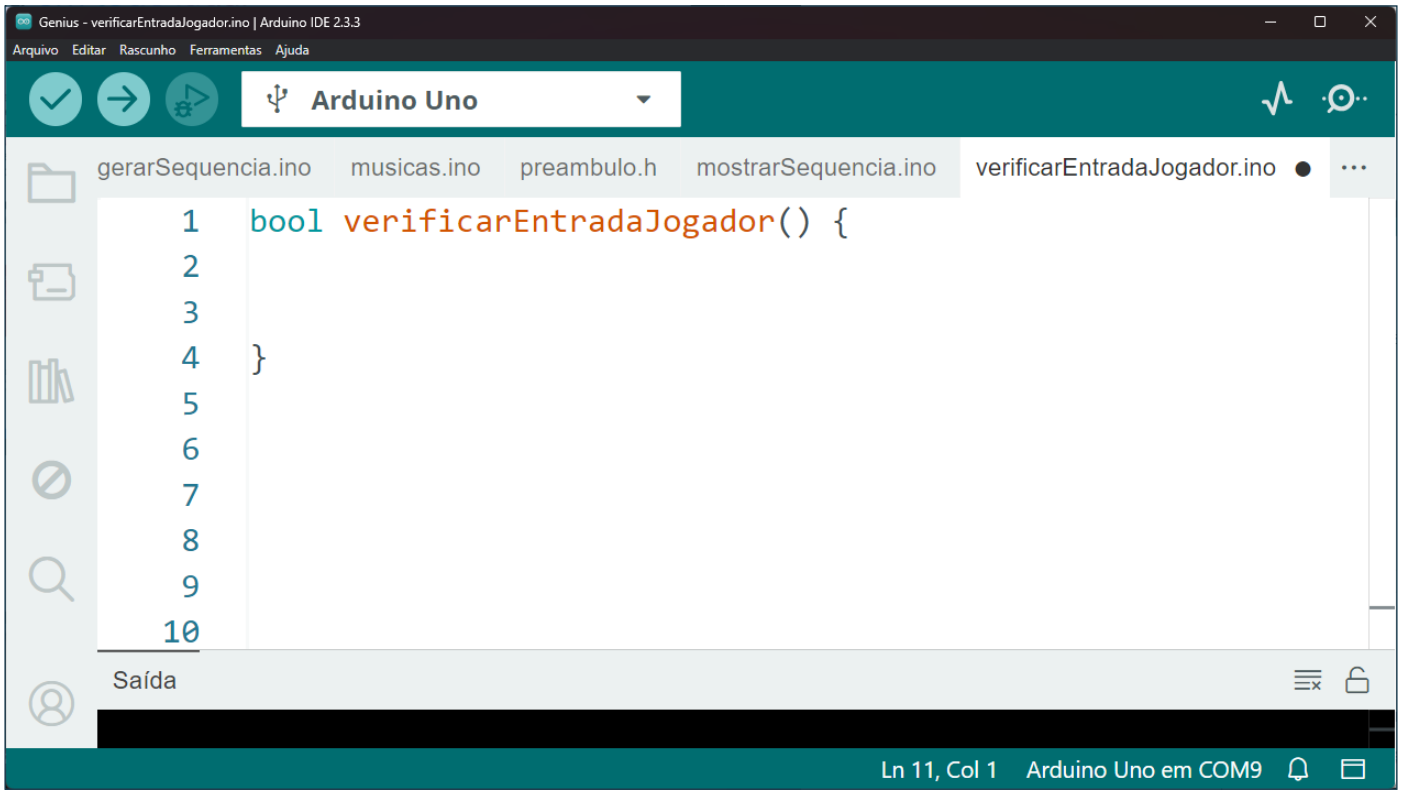
```
/* Função para mostrar a sequência ao jogador. */
void mostrarSequencia() {
  for (int i = 0; i < tamanhoSequencia; i++) {
    acenderLed(sequencia[i]); /* Chama a função que acende o LED da sequência e toca a nota. */
    delay(tempoLedAceso); /* Aguarda pelo tempo definido entre o acionamento de cada LED. */
    apagarLeds(); /* Chama a função que apaga os LEDs e para o som. */
    delay(tempoLedAceso); /* Pausa entre cada LED na sequência. */
  }
}
```

O loop **for()** percorre cada elemento da sequência de LEDs, de **0** até **tamanhoSequencia - 1**, iterando sobre cada elemento da sequência gerada. A função **acenderLed()**, criada na aula anterior, recebe como parâmetro o número do LED que está guardado em **sequência[i]**, acendendo o LED correspondente e tocando a nota musical associada a ele. Então, temos um **delay** que faz o Arduino aguardar pelo tempo definido pela variável **tempoLedAceso**, o que permite ao jogador visualizar o acionamento por tempo suficiente, como definido no preâmbulo do nosso jogo. Após, chama a função que apaga todos os LEDs e para o som, preparando para mostrar o próximo LED da sequência. A segunda pausa com **delay** cria um intervalo entre o desligamento do LED atual e o acendimento do próximo, facilitando a visualização da sequência pelo jogador.

Aula 24 Genius - II

Pensando no funcionamento do jogo Genius, a função que acabamos de criar mostra ao jogador a sequência de LEDs. Quando o jogador a reproduzir, o Arduino precisará saber se a sequência pressionada é correta ou não, o que indicará se o jogador acertou ou errou. Por isso, criaremos uma aba para a próxima função, que agora será do tipo **boolean** para verificar a entrada do jogador.

Figura 3 - Criação da aba verificarEntradaJogador.ino para a função **bool verificarEntradaJogador()**



Fonte: Arduino IDE.

Como estamos na sequência de criação das funções que utilizaremos no **void loop()**, algumas coisas ainda precisarão ser feitas após checar a entrada do jogador. Por enquanto, vamos conferir como fica a nova função booleana **bool verificarEntradaJogador()**.

Aula 24 Genius - II

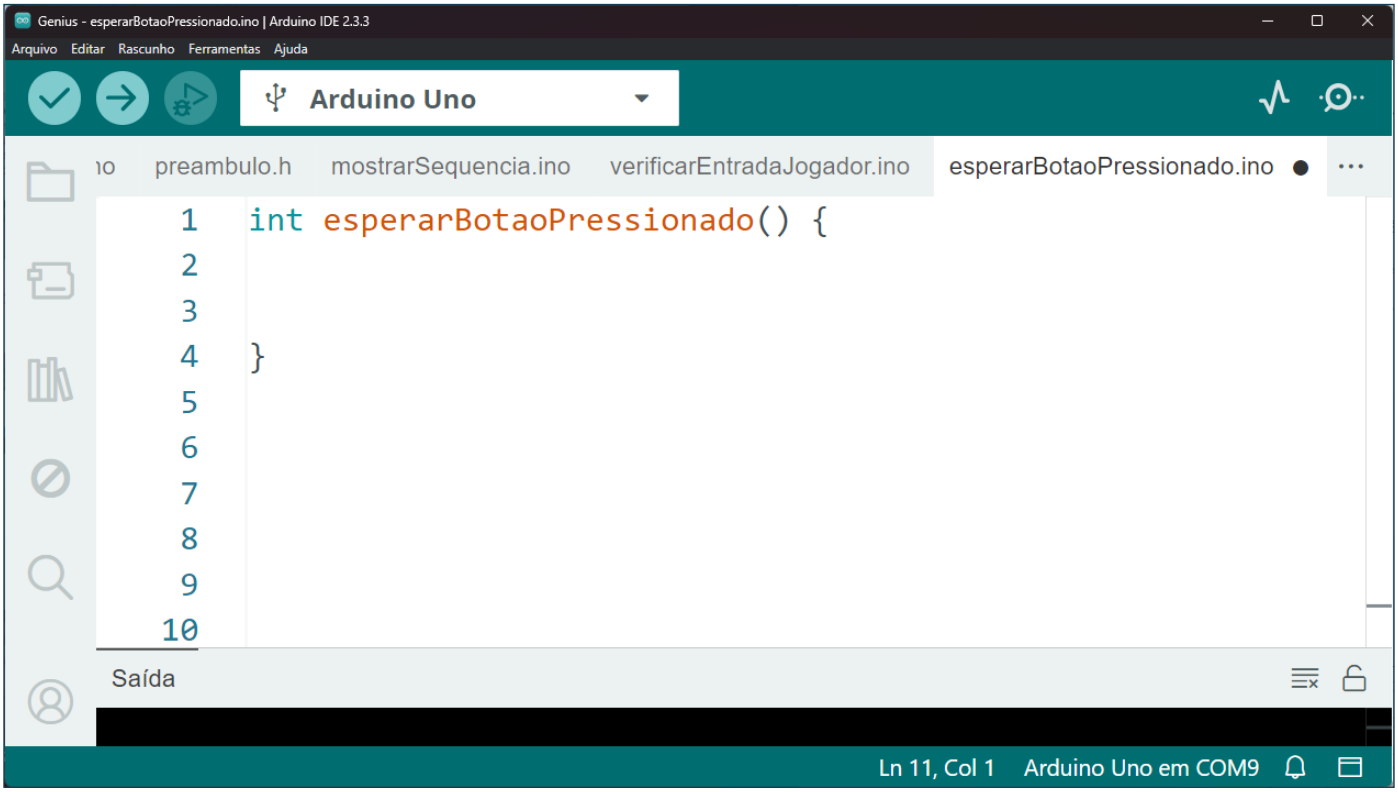
```
/* Função para verificar a entrada do jogador. */
bool verificarEntradaJogador() {
  for (int i = 0; i < tamanhoSequencia; i++) {
    int botaoPressionado = esperarBotaoPressionado();
    if (botaoPressionado != sequencia[i]) {
      return false; /* Entrada incorreta. */
    }
  }
return  true; /* Entrada correta. */
}
```

Essa função é fundamental para validar as ações do jogador e determinar se ele seguiu corretamente a sequência apresentada pelo jogo, permitindo que o jogo continue ou sinalize um erro, ações que definiremos em breve no **void loop()**. A função **verificarEntradaJogador()** verifica se a sequência de botões pressionados pelo jogador corresponde à sequência gerada pelo jogo e inicia pelo loop **for()** para percorrer cada elemento da sequência que foi gerada e mostrada ao jogador. A função **esperarBotaoPressionado()** (que será do tipo inteira e criaremos em outra aba do código) aguarda até que o jogador pressione um botão e retorna o valor correspondente ao botão pressionado. Na sequência, o **if** faz uma comparação pelo operador lógico **!=** se a entrada do jogador, armazenada na variável local **botaoPressionado**, corresponde ao valor de **sequencia[i]**. Se o valor do botão pressionado pelo jogador não corresponder ao valor esperado na sequência, a função retorna **false**, indicando que a entrada está incorreta; se o jogador pressionar todos os botões corretamente, a função retorna **true**, indicando que a entrada correspondeu à sequência gerada pelo jogo.

Aula 24 Genius - II

Para que o Arduino execute a função **verificarEntradaJogador()** corretamente, criaremos a aba **esperarBotaoPressionado** para a função **int esperarBotaoPressionado()** que devolverá um número inteiro conforme mapeamento dos botões do nosso jogo Genius.

Figura 4 - Criação da aba esperarBotaoPressionado.ino para a função **int esperarBotaoPressionado()**



Fonte: Arduino IDE.

Aula 24 Genius - II

Na função, será aplicada a mesma lógica a cada um dos botões do jogo: a função **int esperarBotaoPressionado()** aguarda até que o jogador pressione um dos botões e, em seguida, retorna um valor que representa qual botão foi pressionado.

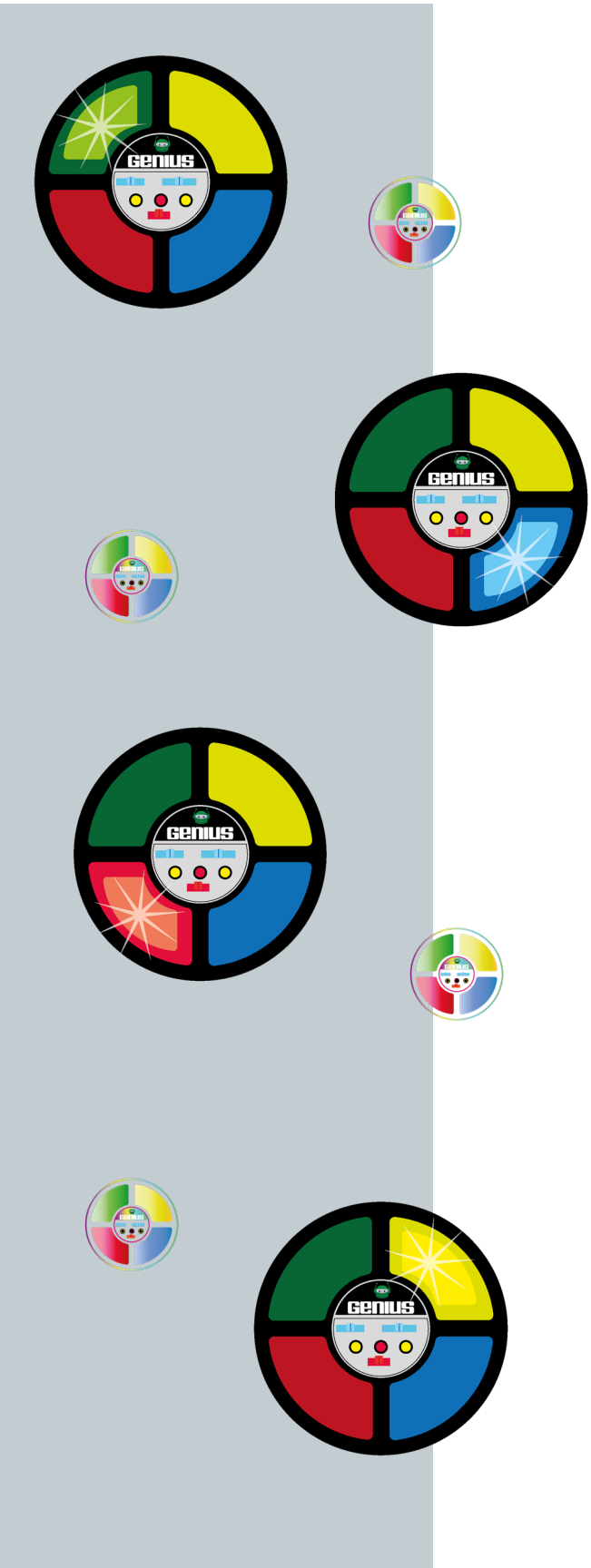
```
/* Função para esperar até que um botão seja pressionado. */
int esperarBotaoPressionado() {
  while (true) {
    if (digitalRead(botaoAmarelo) == LOW) {
      acenderLed(0);
      delay(tempoLedAceso); /* Mantém o LED aceso pelo tempo definido. */
      apagarLeds();
      while (digitalRead(botaoAmarelo) == LOW) {}; /* Espera até o botão ser solto. */
      return 0;
    }
    if (digitalRead(botaoAzul) == LOW) {
      acenderLed(1);
      delay(tempoLedAceso); /* Mantém o LED aceso pelo tempo definido. */
      apagarLeds();
      while (digitalRead(botaoAzul) == LOW) {}; /* Espera até o botão ser solto. */
      return 1;
    }
```

Aula 24 Genius - II

```
  if (digitalRead(botaoVerde) == LOW) {
    acenderLed(2);
    delay(tempoLedAceso); /* Mantém o LED aceso pelo tempo definido. */
    apagarLeds();
    while (digitalRead(botaoVerde) == LOW) {}; /* Espera até o botão ser solto. */
    return 2;
  }
  if (digitalRead(botaoVermelho) == LOW) {
    acenderLed(3);
    delay(tempoLedAceso); /* Mantém o LED aceso pelo tempo definido. */
    apagarLeds();
    while (digitalRead(botaoVermelho) == LOW) {}; /* Espera até o botão ser solto. */
    return 3;
  }
}
    }
```

Nessa função, o **while (true)** cria um loop infinito que mantém o Arduino aguardando até que uma das condições internas seja satisfeita (ou seja, um botão ser pressionado). Vamos analisar a primeira condição? **if (digitalRead(botaoAmarelo) == LOW)** verifica se o botão amarelo foi pressionado. Se sim, sua ação interna é chamar a função **acenderLed()** para acender o LED correspondente ao botão amarelo e tocar a nota associada, manter pelo **delay(tempoLedAceso)** o LED aceso por um tempo definido para então apagar e parar o som. Após, **while (digitalRead(botaoAmarelo) == LOW) {}** aguar-

Aula 24 Genius - II



da até que o botão seja solto (liberado) – esse loop interno garante que a função só continue após o botão ser solto, evitando leituras repetidas pelo Arduino. Por fim, **return 0** retorna o valor **0** à função **verificarEntradaJogador**, indicando que o botão **amarelo** foi pressionado e guardando a informação em sua variável local **botaoPressionado**. Como nosso jogo Genius possui quatro botões conectados, representando cada cor de LED, a função **esperarBotaoPressionado()** repete o mesmo processo de acender o LED, tocar a nota, manter o LED aceso por um tempo, apagar o LED e esperar até que o botão seja solto para os botões **azul**, **verde** e **vermelho**, retornando **1**, **2** e **3**, respectivamente, quando esses botões são pressionados, capturando a entrada do jogador e assegurando que a ação de pressionar um botão seja registrada corretamente no nosso jogo Genius.

Agora, chegou o momento de chamarmos todas as novas funções – criadas nesta e na aula anterior – **no void loop()**!

Finalizada a última função do jogo Genius, vamos ao **void loop()** para definir a lógica geral aplicada ao jogo e controlar o fluxo contínuo do jogo, repetindo-se indefinidamente enquanto o Arduino estiver ligado. E quando desligado, o Arduino físico guardará uma informação importante: os recordes do jogo na memória EEPROM!

Aula 24 Genius - II

```
/* Função loop é executada continuamente. */
void loop() {
  /* Mostra a sequência ao jogador. */
  mostrarSequencia();
  /* Verifica a entrada do jogador. */
  if (verificarEntradaJogador()) { /* Se acertou, faça... */
    Serial.println("Correto! Próxima rodada.");
    delay(1000);
    gerarSequencia();
  } else {
    Serial.println("Game Over!");
    tocarMusicaDerrota();
    /* Atualiza o recorde se o jogador tiver atingido um novo recorde. */
    if (tamanhoSequencia - 1 > recorde) {
      recorde = tamanhoSequencia - 1; /* Atualiza o recorde. */
      EEPROM.put(enderecoRecorde, recorde); /* Grava o recorde na EEPROM. */
      Serial.print("Novo Recorde: ");
      Serial.println(recorde);
    } else {
      Serial.print("Recorde Atual: ");
      Serial.println(recorde);
    }
    esperarNovoJogo();
  }
        }
```

Aula 24 Genius - II

Como já criamos todas as funções para nosso jogo, chegou o momento de aplicarmos as recentes. O **void loop()** inicia com a função **mostrarSequencia()** para exibir a sequência de LEDs que o jogador deve memorizar e reproduzir. Essa função acende e apaga os LEDs conforme a sequência gerada. Depois, aplica-se a estrutura condicional **if com a função verificarEntradaJogador()** para verificar se o jogador reproduziu corretamente a sequência. Se a entrada for correta, a função retorna **true**. Caso contrário, retorna **false**. A partir disso, as condições são aplicadas:

- Se a entrada for correta, o monitor serial exibe o feedback de acerto e, após o **delay** de 1 segundo, a função **gerarSequencia()** é chamada para adicionar um novo passo à sequência, aumentando a dificuldade do jogo.
- Se a entrada for incorreta, o monitor serial exibe o feedback de fim de jogo e a função **tocarMusicaDerrota()** é chamada com a animação de LEDs indicando que o jogo terminou. Então, o Arduino verifica se há um novo recorde e atualiza o valor da variável **recorde**, guardando essa informação na memória EEPROM pela função **EEPROM.put(enderecoRecorde, recorde)**. Por fim, imprime o recorde (novo ou atual) e chama a função **esperarNovoJogo()** para nova partida.

Finalizando toda a programação do nosso Genius, percebemos que a função **void loop()** controla a lógica principal e garante que o jogo continue a partir de onde parou ou reinicie quando necessário. E por ela também exploramos uma novidade para a programação tanto desse quanto de outros projetos que você possa desenvolver: o recurso a memória EEPROM, cujos detalhes conhecemos na aula anterior e podemos explorar no Arduino físico.

Confira pelo **Arduino IDE Online** como ficou o código na íntegra! É ele que carregaremos ao protótipo físico, na próxima aula, para partidas desafiadoras!

Acesse **Aula 24 – Genius [Código Completo]**

Que tal, antes de montarmos o protótipo, testarmos uma partida pelo **simulador Wokwi**? Assim, você já confere se serão precisos ajustes, por exemplo, no tempo de acionamento entre os LEDs.

Vamos jogar?

Teste nosso jogo Genius no simulador Wokwi!

Boa sorte e diversão!!

Aula 24 Genius - II

Desafio

Que tal pensar em outras músicas e controle dos LEDs tanto para o início quanto para a finalização do jogo, criando alternativas com outras notas musicais e tempos?

E se...

O projeto não funcionar?

- Observe as portas conectadas no simulador e declaradas na programação.
- Verifique as funções criadas, com base na programação final do jogo Genius.

Eu quiser um jogo Genius mais rápido para ser finalizado?

- Você poderá redefinir quantos elementos seu jogo terá, alterando o valor inteiro atribuído no array de bytes **int sequencia[100]**.

3. Feedback e finalização

Hoje concluímos a programação do nosso jogo Genius utilizando Arduino e já realizamos os testes antes de chegarmos à versão final com o protótipo físico. Cada etapa do processo é essencial para compreendermos desde a criação das funções específicas para o jogo até os conceitos relacionados à memória EEPROM, a qual testaremos na próxima aula!

REFERÊNCIAS

ARDUINO. **A guide to EEPROM**. Disponível em: https://docs.arduino.cc/learn/programming/eeprom-guide/. Acesso em: 10 out. 2024.

ARDUINO. **Documentação de Referência da Linguagem Arduino**. Disponível em: https://www.arduino.cc/reference/pt/. Acesso em: 27 mai. 2024.

ARDUINO. **EEPROM Library**. Disponível em: https://docs.arduino.cc/learn/built-in-libraries/eeprom/. Acesso em: 10 out. 2024.

V&A. **Simon**. Disponível em: https://collections.vam.ac.uk/item/O1243092/simon-electronic-game--ralph-h-baer/. Acesso em: 05 nov 2024.

DIRETORIA DE TECNOLOGIAS E INOVAÇÃO (DTI)

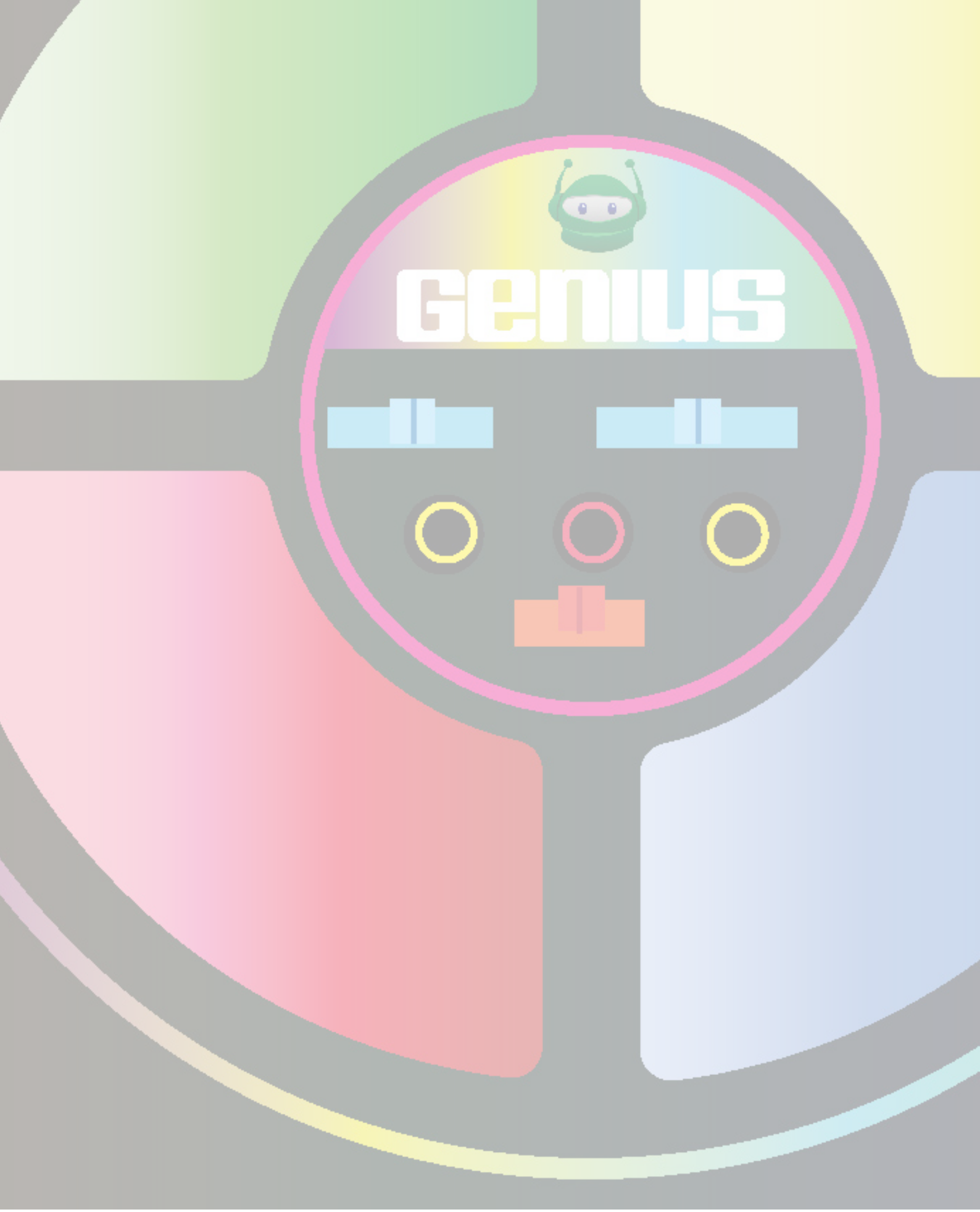
COORDENAÇÃO DE TECNOLOGIAS EDUCACIONAIS (CTE)

EQUIPE ROBÓTICA PARANÁ

- Adilson Carlos Batista
- Ailton Lopes
- Andrea da Silva Castagini Padilha
- Cleiton Rosa
- Darice Alessandra Deckmann Zanardini
- Edna do Rocio Becker
- Kellen Pricila dos Santos Cochinski
- Marcelo Gasparin
- Michele Serpe Fernandes
- Michelle dos Santos
- Roberto Carlos Rodrigues
- Sandra Aguera Alcova Silva
- Viviane Dziubate Pittner

Os materiais, aulas e projetos da "Robótica Paraná", foram produzidos pela Coordenação de Tecnologias Educacionais (CTE), da Diretoria de Tecnologia e Inovação (DTI), da Secretaria de Estado da Educação do Paraná (SEED), com o objetivo de subsidiar as práticas docentes com os estudantes por meio da Robótica.
Este material foi produzido para uso didático-pedagógico exclusivo em sala de aula.

Este trabalho está licenciado com uma Licença
Creative Commons – CC BY-NC-SA
Atribuição - NãoComercial - CompartilhaIgual 4.0



DTI - DIRETORIA DE TECNOLOGIA E INOVAÇÃO

PARANÁ
GOVERNO DO ESTADO
SECRETARIA DA EDUCAÇÃO