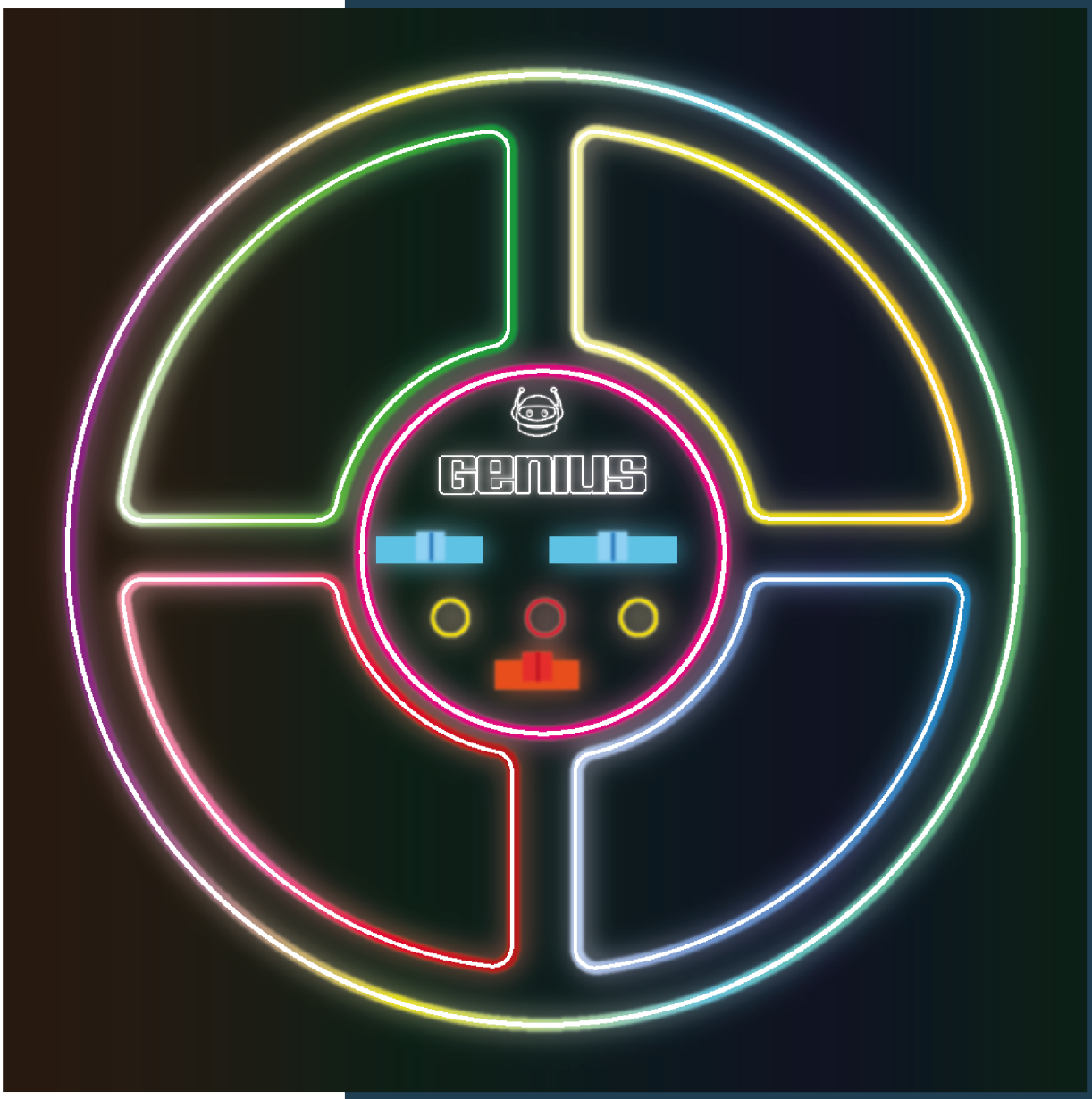
Robótica Educacional

Módulo 3



Aula 23

Genius - I

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
Darice Alessandra Deckmann Zanardini

Validação de Conteúdo
Cleiton Rosa

Revisão Textual
Kellen Pricila dos Santos Cochinski

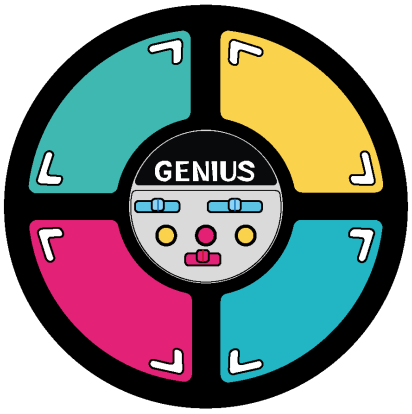
Projeto Gráfico e Diagramação
Edna do Rocio Becker

2024

Sumário

Aula 23 Genius - I

Introdução



Que tal momentos de diversão e estímulo ao raciocínio lógico com um jogo da memória?

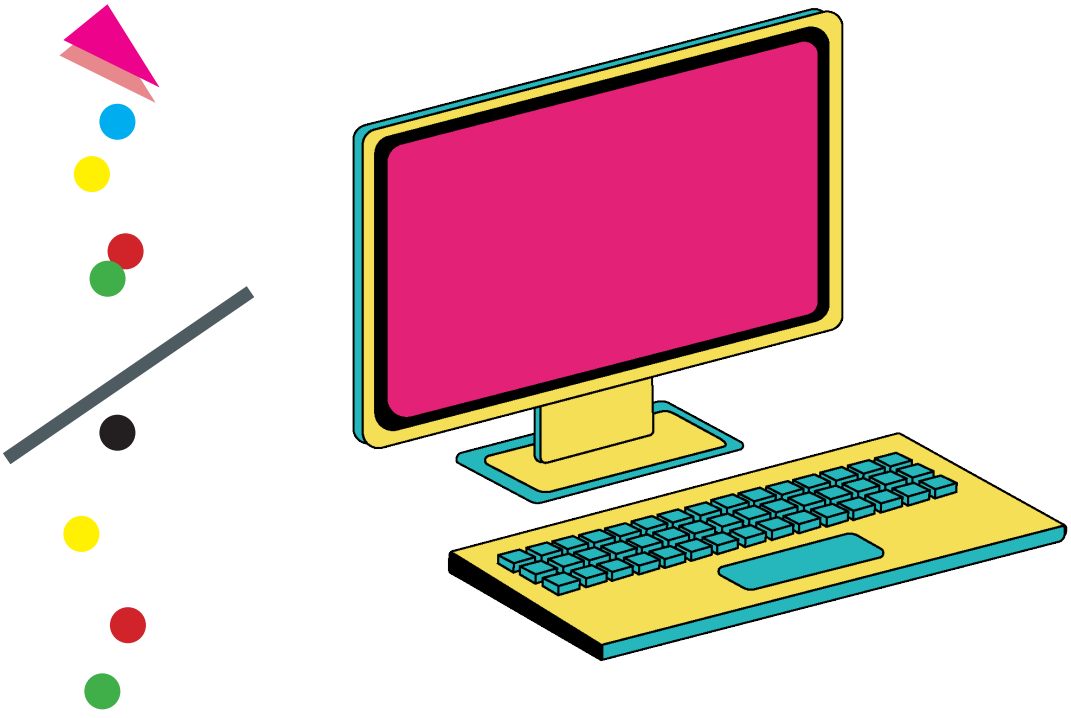
Nesta aula, teremos uma introdução ao clássico jogo Genius e iniciaremos sua programação, desenvolvendo na próxima aula o protótipo de uma versão com Arduino, LEDs, botões e buzzer para criar a nossa versão personalizada!

Objetivos desta aula

- Conhecer o jogo Genius e as habilidades relacionadas;
- Programar parte do jogo 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 23 Genius - I

Roteiro da aula

1. Contextualização

Nesta aula, vamos mergulhar em mais um momento de nostalgia (como fizemos nas aulas inspirados nos filmes "Super Máquina" e "De Volta para o Futuro" deste Módulo 3 de Robótica Educacional), com o clássico jogo Genius, ícone da década de 1980 e que até hoje desafia nossas habilidades de memória e reflexo.

Mais conhecido como "Simon" em outras partes do mundo, justamente seu nome original, o jogo foi lançado nos Estados Unidos em 1978 e rapidamente tornou-se um sucesso! Seus criadores, Ralph H. Baer (1922 - 2014) e Howard J. Morrison (1932), focaram na simplicidade do jogo, isso o tornou genial: repetir uma sequência de luzes e sons sem errar! E o mais bacana é que a cada rodada a dificuldade aumenta, pois são adicionados novos elementos a serem repetidos.

Aula 23 Genius - I

E como jogar? Observando a versão original, o jogo consiste em quatro botões coloridos, cada um associado a uma nota musical. Ao iniciar a partida, Simon (Genius) exibe uma sequência aleatória de luzes e sons e o jogador precisa repetir a sequência corretamente, como em um jogo estilo "siga o líder". Então, Simon (Genius) segue ampliando a sequência e se o jogador errar, o jogo termina. É uma brincadeira que testa e aprimora a memória visual e/ou auditiva, além de proporcionar muita diversão e estimular novas conquistas!

Figura 1 - Jogo Simon (Genius, no Brasil)

Fonte: Wikimedia Commons.

SAIBA MAIS!

"Um símbolo da cultura pop dos anos 1980, 'Simon' foi criado depois que seus inventores viram um jogo de repetição semelhante em uma convenção de jogos de arcade, nos anos 1970. Fabricado pela Atari, 'Touch Me' criou uma sequência de luzes e sons que o jogador teria que repetir pressionando a mesma sequência de botões. Foi um fracasso nos fliperamas, competindo contra nomes como 'Pac Man' e 'Space Invaders'. Os designers de 'Simon', Baer e Morrison, decidiram que poderiam competir com a máquina Atari criando um jogo para jogar em casa, melhorando a aparência e os sons. Eles levaram seu design para a MB, que capitalizou seu estilo disco ao lançá-lo na boate Studio 54, em Nova York. Um grande sucesso e inovação em jogos domésticos, é tão icônico de sua época que agora é invariavelmente usado como uma abreviação para a década de 1980 em filmes e programas de televisão."

Fonte: Victoria and Albert Museum

2. Montagem e programação

Realizaremos toda a montagem do jogo Genius diretamente na protoboard com LEDs, buzzer e botões na **Aula 24 – Genius [Parte II]** e a programação iniciaremos nesta aula 23. Desde já, você e seus colegas poderão pensar em como recriar o protótipo do Genius com Arduino em uma caixa personalizada, por exemplo, personalizando ainda mais suas versões do jogo, assim como ocorre com Simon, que possui diversas versões.

Figura 2 - Simon de bolso

Fonte: Wikimedia Commons.

Aula 23 Genius - I

Vamos começar a programação para criarmos sequências de cores do nosso jogo Genius? E, caso você queira antecipar um protótipo para testes, ao final desta aula compartilharemos a versão pelo simulador Wokwi.

Agora, vamos programar!

Para a programação do jogo Genius, seguiremos a mesma proposta adotada nas aulas anteriores: **programação por abas**. A organização de uma aba para preâmbulo e cada função, facilita para adaptar ou melhorar o código em momentos futuros de revisão, ou mesmo ampliação do projeto, como a inclusão de um display, por exemplo.

Na programação desta aula, utilizaremos uma biblioteca para manipular um recurso que a placa Arduino oferece e ainda não abordamos em nossas aulas de Robótica Educacional: a **memória EEPROM**, que significa algo como "Memória de somente leitura programável e apagável eletronicamente", um tipo de memória não volátil usada para armazenar dados que precisam ser mantidos mesmo quando o dispositivo está desligado.

Saiba mais!

Quando realizarmos o teste do nosso protótipo no Arduino físico, no próximo encontro, exploraremos o recurso da memória **EEPROM** – pelo simulador não conseguiremos, pois ele não possui o recurso.

Quando jogarmos Genius pelo Arduino físico, os dados do último recorde serão armazenados para a próxima jogada. Então, vamos entender como a **EEPROM** funciona?

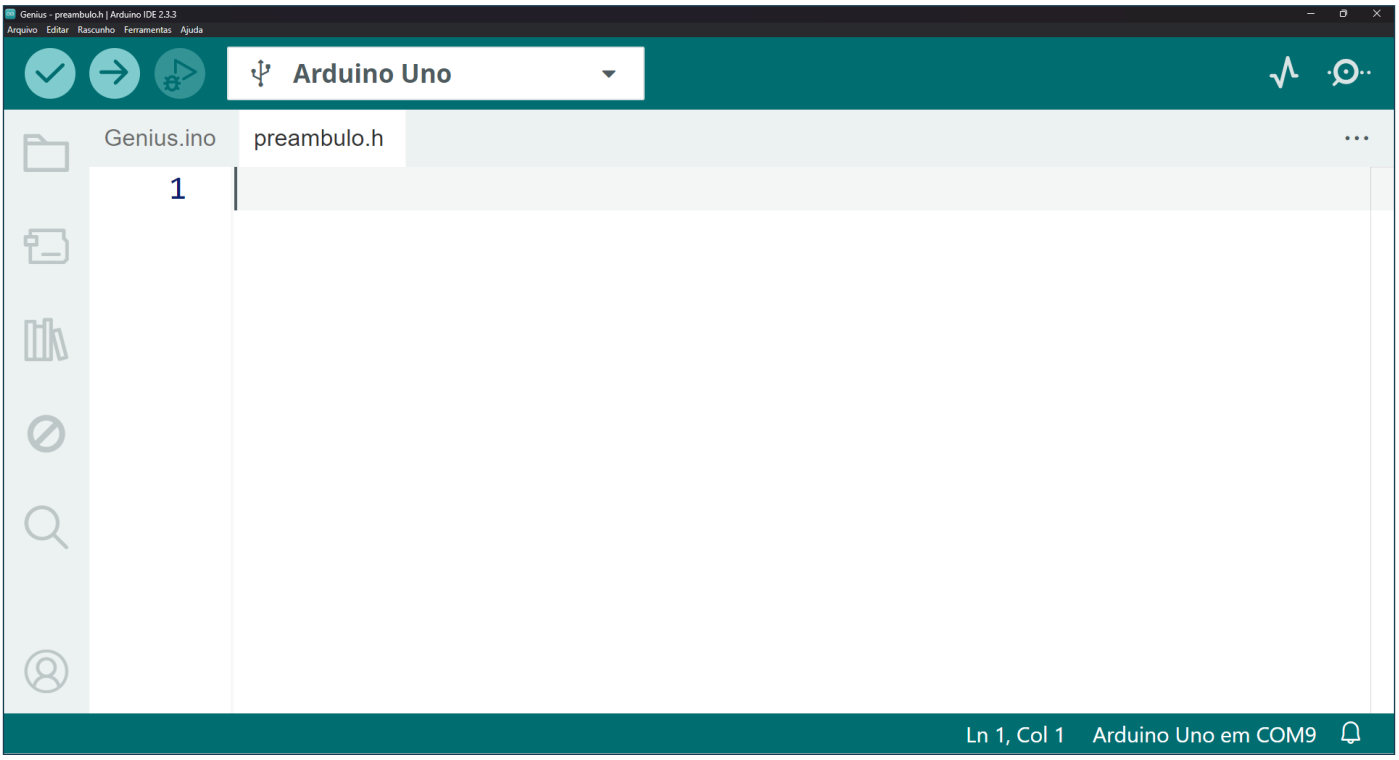
Diferente da memória RAM que perde seus dados quando o Arduino é desligado, a **EEPROM** mantém os dados salvos mesmo sem energia, o que é perfeito para armazenar informações importantes como os recordes do nosso jogo.

Em comparação às memórias de outros dispositivos mais robustos, como computadores, a **EEPROM** no Arduino possui capacidade limitada, mas suficiente para armazenar quantidades menores de dados, como as pontuações do jogo. Portanto, poderemos tanto ler quanto gravar dados na **EEPROM**, ou seja, salvar o novo recorde de jogo e, posteriormente, recuperar essa informação quando nosso jogo Genius for ligado novamente. Assim, quando você e seus colegas jogarem novamente, poderão tentar quebrar o recorde anterior!

Boa sorte!

Em um novo sketch, abriremos a primeira aba (de cabeçalho/header) da programação: **preambulo.h**.

Figura 3 - Criação da aba do tipo header (cabeçalho) **preambulo.h**



Fonte: Arduino IDE.

Como utilizaremos uma biblioteca para manipular a memória **EEPROM**, a adicionaremos à programação sem necessidade de instalá-la, pois é nativa do Arduino.

```
/* Biblioteca que permitirá a gravação de dados na memória EEPROM do Arduino. */

#include <EEPROM.h>
```

Na sequência, definiremos as portas dos componentes que conectaremos ao protótipo: LEDs, botões e buzzer.

Aula 23 Genius - I

```
/* Definição dos pinos dos LEDs. */
#define ledAmarelo 8
#define ledAzul 6
#define ledVerde 4
#define ledVermelho 2

/* Definição dos pinos dos botões. */
#define botaoAmarelo 9
#define botaoAzul 7
#define botaoVerde 5
#define botaoVermelho 3

/* Definição do pino do buzzer. */
#define buzzer 11
```

Como comentamos no início desta aula, a ideia é personalizar o jogo e a próxima definição se refere à primeira personalização: as notas a serem emitidas pelo buzzer.

```
/* Notas musicais correspondentes aos LEDs. */
#define notaAmarelo 262 /* Dó */
#define notaAzul 294 /* Ré */
#define notaVerde 330 /* Mi */
#define notaVermelho 349 /* Fá */
```

Aula 23 Genius - I

Próxima personalização define o "movimento do jogo" e corresponde à velocidade da sequência, ou seja, ao tempo de acionamento de cada LED.

```
/* Tempo, em milissegundos, de acionamento de cada LED. */

#define tempoLedAceso 300
```

Nos primeiros testes do jogo, você e seus colegas perceberão se a velocidade de acionamento dos LEDs estará ok para a jogo de vocês, podendo redefinir depois a velocidade de acionamento da sequência de LEDs como mais lenta ou mais rápida.

A próxima definição refere-se ao endereço de memória EEPROM que guardará o recorde da partida na variável **enderecoRecorde**. A EEPROM do Arduino Uno tem uma capacidade de armazenamento de 1024 bytes. Isso significa que poderemos armazenar até 1024 bytes de dados, o que é suficiente para pequenas quantidades de informações, como os recordes do nosso jogo.

```
/* Endereço na EEPROM para armazenar o recorde (0 a 1023). */

#define enderecoRecorde 0
```

Usar um endereço de memória na programação é essencial porque cada byte da EEPROM tem um endereço único e podemos armazenar diferentes tipos de informações em diferentes intervalos de endereços, o que ajuda a evitar a sobreposição de dados, garantindo que novas informações não substituam as antigas inadvertidamente.

Para o Arduino manipular o recorde, fazer as comparações e guardar esse valor, precisamos de uma variável para guardar o recorde. Como iniciaremos um jogo do zero, a variável terá inicialmente o valor **0** atribuído e, conforme o recorde será quebrado, atualizaremos esse valor, guardando-o na EEPROM do Arduino.

```
/* Variável para armazenar o recorde. */

int recorde = 0;
```

Aula 23 Genius - I

As próximas variáveis referem-se à sequência do jogo com uma estrutura que permitirá ao Arduino armazenar e controlar a sequência de passos que o jogador precisa seguir para avançar no Genius, além de facilitar a comparação com a entrada do jogador para verificar acertos ou erros: **sequencia[100]** é o **array** que armazena até 100 elementos da sequência do jogo Genius e **tamanhoSequencia** é a variável que controla o número atual de elementos na sequência gerada, começando em **0** e aumentando conforme o jogo progride.

```
/* Array para armazenar a sequência gerada pelo jogo. */
/* O tamanho do array é 100, o que significa que ele pode armazenar até 100 elementos. */
int sequencia[100];


/* Variável para controlar o tamanho atual da sequência gerada. */
/* Indica quantos elementos da sequência foram gerados e devem ser reproduzidos pelo jogador. */
int tamanhoSequencia = 0;
```

Dica!

Caso você, após finalizarmos o Genius na próxima aula, decida ampliar seu protótipo com a adição de um display, por exemplo, observe, na compilação do código pelo Arduino IDE, o percentual da memória do Arduino utilizada. Caso ultrapasse 100%, reduza o tamanho da sequência do seu jogo!

Com esses elementos, finalizamos a aba **prembulo.h.** Vamos ver como ficou esse trecho completo?

Aula 23 Genius - I

```
/* Biblioteca que permitirá a gravação de dados na memória eeprom
do Arduino. */
#include <EEPROM.h>

/* Definição dos pinos dos LEDs. */
#define ledAmarelo 10
#define ledAzul 8
#define ledVerde 6
#define ledVermelho 4

/* Definição dos pinos dos botões. */
#define botaoAmarelo 11
#define botaoAzul 9
#define botaoVerde 7
#define botaoVermelho 5

/* Definição do pino do buzzer. */
#define buzzer 2

/* Notas musicais correspondentes aos LEDs. */
#define notaAmarelo 262 /* Dó */
#define notaAzul 294 /* Ré */
#define notaVerde 330 /* Mi */
#define notaVermelho 349 /* Fá */
```

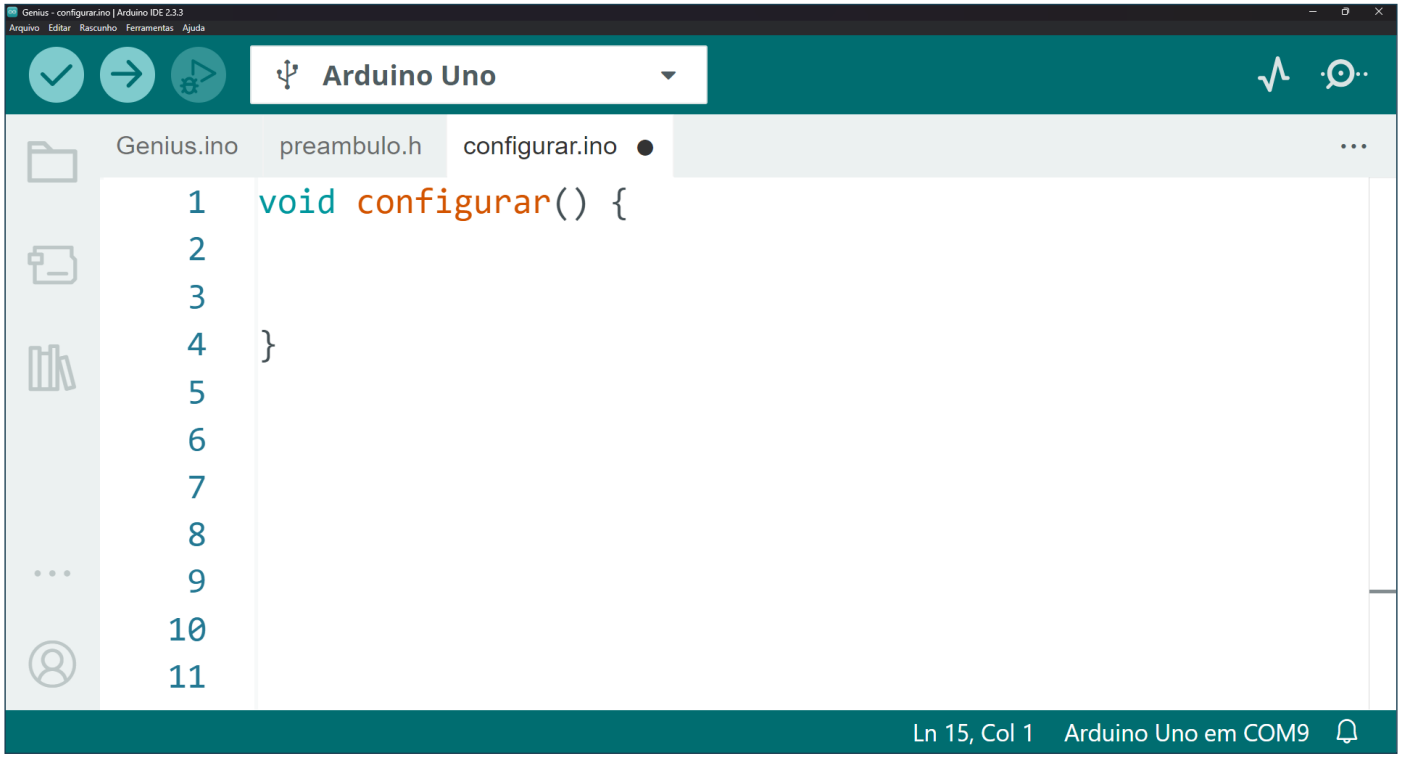
Aula 23 Genius - I

```
/* Tempo que o LED fica aceso em milissegundos. */
#define tempoLedAceso 300

/* Endereço na EEPROM para armazenar o recorde (0 a 1023). */
#define enderecoRecorde 0

/* Variável para armazenar o recorde. */
int recorde = 0;

/* Array para armazenar a sequência gerada pelo jogo. */
/* O tamanho do array é 100, o que significa que ele pode armazenar até 100 elementos. */
int sequencia[100];

/* Variável para controlar o tamanho atual da sequência gerada. */
/* Indica quantos elementos da sequência foram gerados e devem ser reproduzidos pelo jogador. */
int tamanhoSequencia = 0;
```

Aula 23 Genius - I

Seguiremos com a programação do Genius criando novas abas a cada função do jogo. A primeira função, **configurar()**, portanto, será criada como arquivo do tipo **.ino** e receberá o mesmo nome:

Figura 4 - Criação da aba configurar**.ino** para a função **void configurar()**



Fonte: Arduino IDE.

Nessa nova aba, faremos todas as configurações dos LEDs e botões conectados ao Arduino, além de iniciar a comunicação serial e utilizar um recurso do Arduino que será essencial para o nosso jogo ser mais dinâmico quanto à geração de sequências criadas: a função **randomSeed()**, a qual lerá um valor aleatório do ambiente e usará esse valor para iniciar o gerador de números aleatórios, garantindo assim que as sequências do jogo Genius sejam sempre diferentes e imprevisíveis, o que fará com que nosso jogo seja mais emocionante.

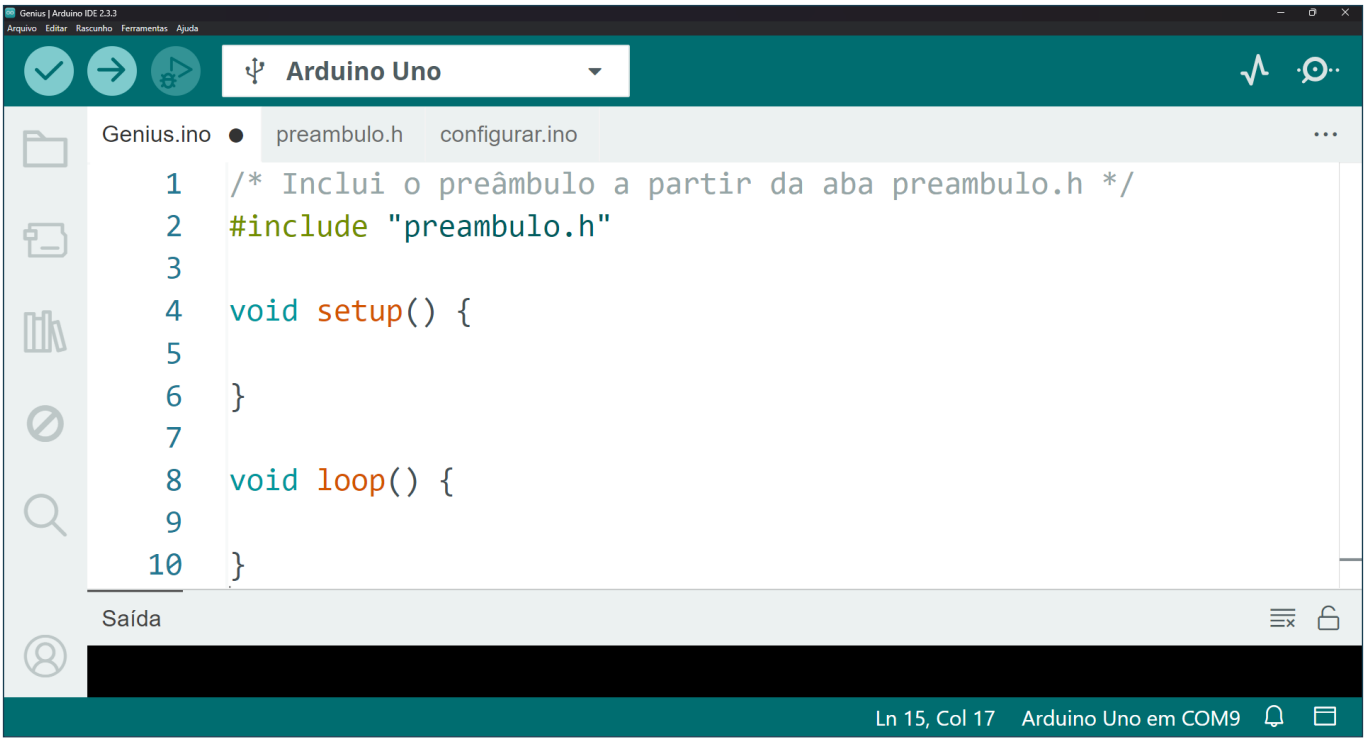
Aula 23 Genius - I

```
void configurar() {
  /* Configuração dos pinos dos LEDs como saída. */
  pinMode(ledAmarelo, OUTPUT);
  pinMode(ledAzul, OUTPUT);
  pinMode(ledVerde, OUTPUT);
  pinMode(ledVermelho, OUTPUT);

  /* Configuração dos pinos dos botões como entrada pull-up. */
  pinMode(botaoAmarelo, INPUT_PULLUP);
  pinMode(botaoAzul, INPUT_PULLUP);
  pinMode(botaoVerde, INPUT_PULLUP);
  pinMode(botaoVermelho, INPUT_PULLUP);

  /* Inicialização da comunicação serial para debug. */
  Serial.begin(9600);

  /* Inicializa o gerador de números aleatórios. */
  randomSeed(analogRead(0));
}
```

Observando a linha da função **randomSeed()**, a função lê, pelo **analogRead(0)** como parâmetro, o valor do pino analógico do Arduino. Esse pino não está conectado em nada específico, então ele faz a coleta de valores aleatórios entre 0 e 1023 porque a porta fica em um estado flutuante, o que gera números aleatórios com um valor diferente toda vez que o programa é iniciado, garantindo assim que as sequências geradas para nosso Genius sejam diferentes a cada jogada.

Partindo para o código principal da programação, adicionaremos o preâmbulo que finalizamos em aba específica na aba do sketch principal da programação do Genius utilizando **#include "preambulo.h"**.

Figura 5 - Sketch principal da programação com inclusão de **preambulo.h**



Fonte: Arduino IDE.

```
/* Inclui o preâmbulo a partir da aba preambulo.h */
#include "preambulo.h"
```

Feita essa inclusão, seguimos direto para a função **void setup()** para chamar a função **configurar()**, a qual acabamos de criar!

```
void setup() {

  /* Chama a função que faz as configurações necessárias. */
  configurar();

}
```

Aula 23 Genius - I

Ainda no **void setup()**, vamos adicionar um comando que carrega o valor do recorde anterior armazenado na EEPROM e o coloca na variável **recorde**, garantindo que o histórico de pontuações seja mantido mesmo após desligamentos. Mas atenção, esse recurso funcionará no protótipo físico com Arduino – nos testes pelo simulador, você verá que o Arduino do Wokwi não possui memória EEPROM!

```
/* Carrega o recorde da EEPROM. */
EEPROM.get(enderecoRecorde, recorde);
```

Essa função é usada para ler dados armazenados na EEPROM e utiliza dois parâmetros, já criados como variáveis: **enderecoRecorde**, que é o endereço na EEPROM onde o recorde foi previamente armazenado, e **recorde**, onde o valor lido da EEPROM será armazenado.

Quando o Arduino físico é iniciado, a função **EEPROM.get** acessa a memória EEPROM no endereço especificado e carrega o valor armazenado nesse endereço, guardando-o na variável **recorde**. Isso permite que o jogo acesse o recorde anterior do Genius, mesmo após o Arduino físico ser desligado e reiniciado, mantendo a persistência dos dados e permitindo aos jogadores ver e tentar bater o recorde anterior, tornando o jogo mais competitivo e divertido!

E como a ideia é ver o **recorde**, seguimos no **void setup()** para imprimir o recorde atual dentro do intervalo de **0** a **100**, o valor atribuído à nossa sequência de jogo pelo **array** do preâmbulo.

```
Serial.print("Recorde Atual: ");
if (recorde < 0 || recorde > 100) {
  Serial.println("0");
} else {
  Serial.println(recorde);
}
```

Aula 23 Genius - I

Por fim, incluiremos no **void setup()** a próxima função a ser criada em aba específica **.ino**.

```
/* Espera pelo jogador para iniciar a partida. */
esperarNovoJogo();
```

Como preparamos o **void setup()** em etapas, para compreender suas linhas, que tal aquela checagem geral no **void setup()** completo?

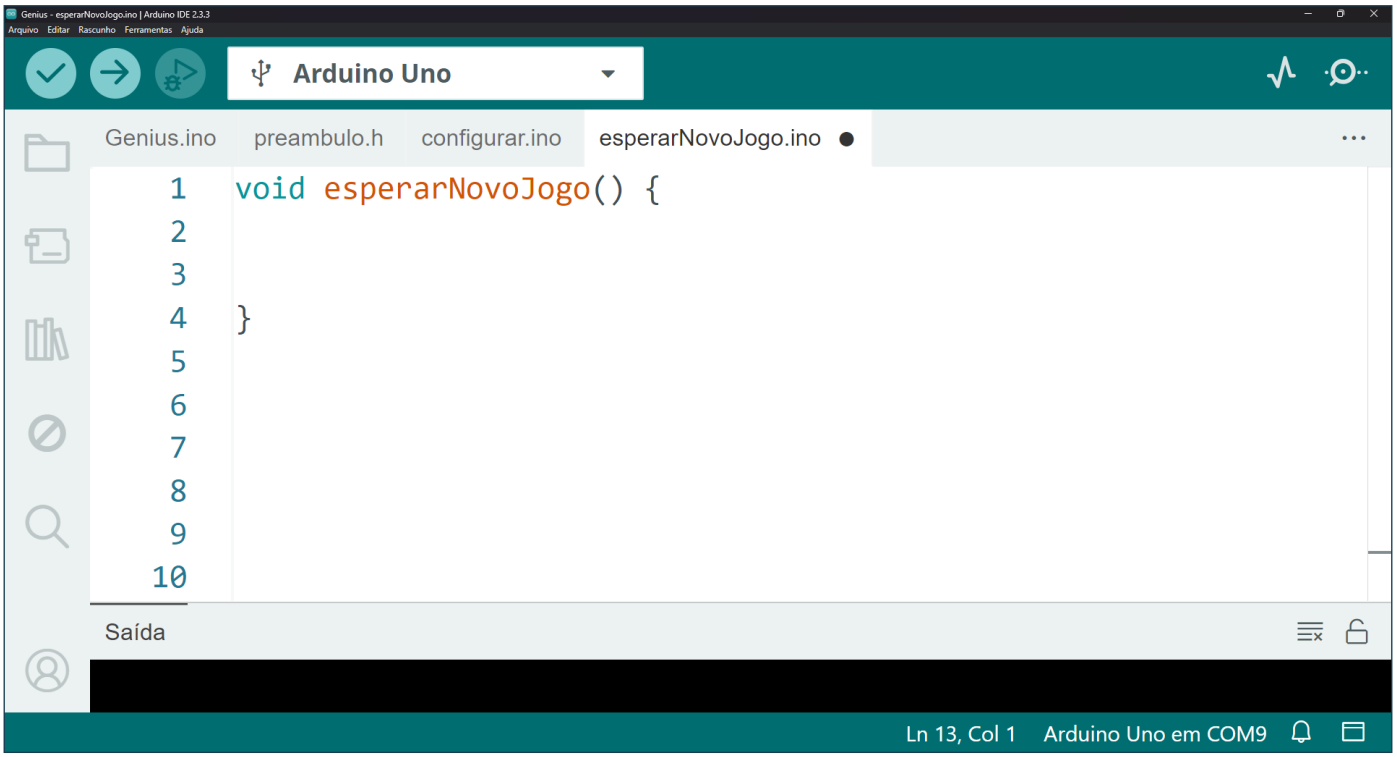
Quadro 1 - Função **void setup()** completa

```
void setup() {
  /* Chama a função que faz as configurações necessárias. */
  configurar();

  /* Carrega o recorde da EEPROM. */
  EEPROM.get(enderecoRecorde, recorde);
  Serial.print("Recorde Atual: ");
  if (recorde < 0 || recorde > 100) {
    Serial.println("0");
  } else {
    Serial.println(recorde);
  }

  /* Espera pelo jogador para iniciar a partida. */
  esperarNovoJogo();
}
```

Aula 23 Genius - I

Finalizada a função **void setup()**, executada uma vez quando o Arduino é ligado, vamos abrir mais uma aba **.ino** para a função **esperarNovoJogo()**.

Figura 6 - Criação da aba esperarNovoJogo.ino para a função **void esperarNovoJogo ()**



Fonte: Arduino IDE.

Nessa nova função, também do tipo **void**, aplica-se a lógica, com instruções pelo monitor serial, de como se dará o início do jogo: via pressionamento de qualquer um dos botões conectados ao Arduino, o que já pediremos para ser impresso pelo Arduino.

```
/* Função para esperar até que um botão seja pressionado para iniciar um novo jogo. */
void esperarNovoJogo() {
  /* Mensagem aguardando a ação do jogador para iniciar uma partida. */
  Serial.println("Pressione qualquer botão para iniciar...");
}
```

Aula 23 Genius - I

Na sequência da programação da função **esperarNovoJogo()**, adicionaremos uma estrutura para aguardar o jogador se manifestar, por um dos botões, para o início da partida e iniciar a partida, gerando uma nova sequência de acionamentos para a partida e tocando a música de abertura com uma animação de LEDs. O **while()** é uma estrutura de controle de fluxo que executa o bloco de código dentro dele repetidamente enquanto a condição especificada for verdadeira. Nesse caso, a condição é **true**, que é sempre verdadeira, criando o loop infinito ou espera infinita que queremos para aguardar o jogador.

```
/* Cria uma espera "infinita" por um botão. */
while (true) {
  /* Verifica se um dos 4 botões foi pressionado. */
  if (digitalRead(botaoAmarelo) == LOW || digitalRead(botaoAzul) == LOW || digitalRead(botaoVerde) == LOW || digitalRead(botaoVermelho) == LOW) {

    /* Aguarda o botão pressionado ser solto. */
    while (digitalRead(botaoAmarelo) == LOW || digitalRead(botaoAzul) == LOW || digitalRead(botaoVerde) == LOW || digitalRead(botaoVermelho) == LOW) {};
    tamanhoSequencia = 0; /* Reinicia o tamanho da sequência. */
    gerarSequencia(); /* Gera uma nova sequência. */
    tocarMusicaAbertura(); /* Toca a música de abertura com animação de LEDs. */
    break;
  }
}
```

Aula 23 Genius - I

No trecho adicionado à função **esperarNovoJogo()**, o primeiro loop cria uma espera infinita, onde o programa fica parado até que uma das quatro condições seja satisfeita: dentro do loop, a função **digitalRead()** lê o estado dos botões e o código verifica constantemente se qualquer um dos quatro botões (amarelo, azul, verde, vermelho) foi pressionado.

O segundo **while** é usado para criar uma espera até que o botão pressionado seja solto, garantindo que o código só continuará depois que o jogador liberar o botão, evitando leituras múltiplas da mesma ação. Enquanto qualquer um dos botões – os indicamos entre **||**, que na linguagem se refere ao "ou" – estiver pressionado (estado == **LOW** por estarem em **INPUT_PULLUP**), o loop continua executando e o segundo **while** mantém o programa "parado" até que todos os botões estejam soltos.

Como a ideia da função **esperarNovoJogo()** é iniciar um novo jogo, após a liberação do botão a variável **tamanhoSequencia = 0** zera o tamanho da sequência, preparando para uma nova rodada do jogo. A seguir chama duas funções que ainda criaremos para gerar uma nova sequência e tocar a música de abertura.

Finalizamos a função **esperarNovoJogo()** com **break** para interromper o primeiro **while (true)**, permitindo que o programa continue por sair do loop infinito. E, nesse passo da programação, entendemos como iniciar um novo jogo: basta apertar qualquer um dos botões conectados ao protótipo.

Quadro 2 - Função **void esperarNovoJogo ()** completa

```
/* Função para esperar até que um botão seja pressionado para iniciar um novo jogo. */
void esperarNovoJogo() {
 /* Mensagem aguardando a ação do jogador para iniciar uma partida. */
 Serial.println("Pressione qualquer botão para iniciar...");
 /* Cria uma espera "infinita" por um botão. */
 while (true) {
  /* Verifica se um dos 4 botões foi pressionado. */
```

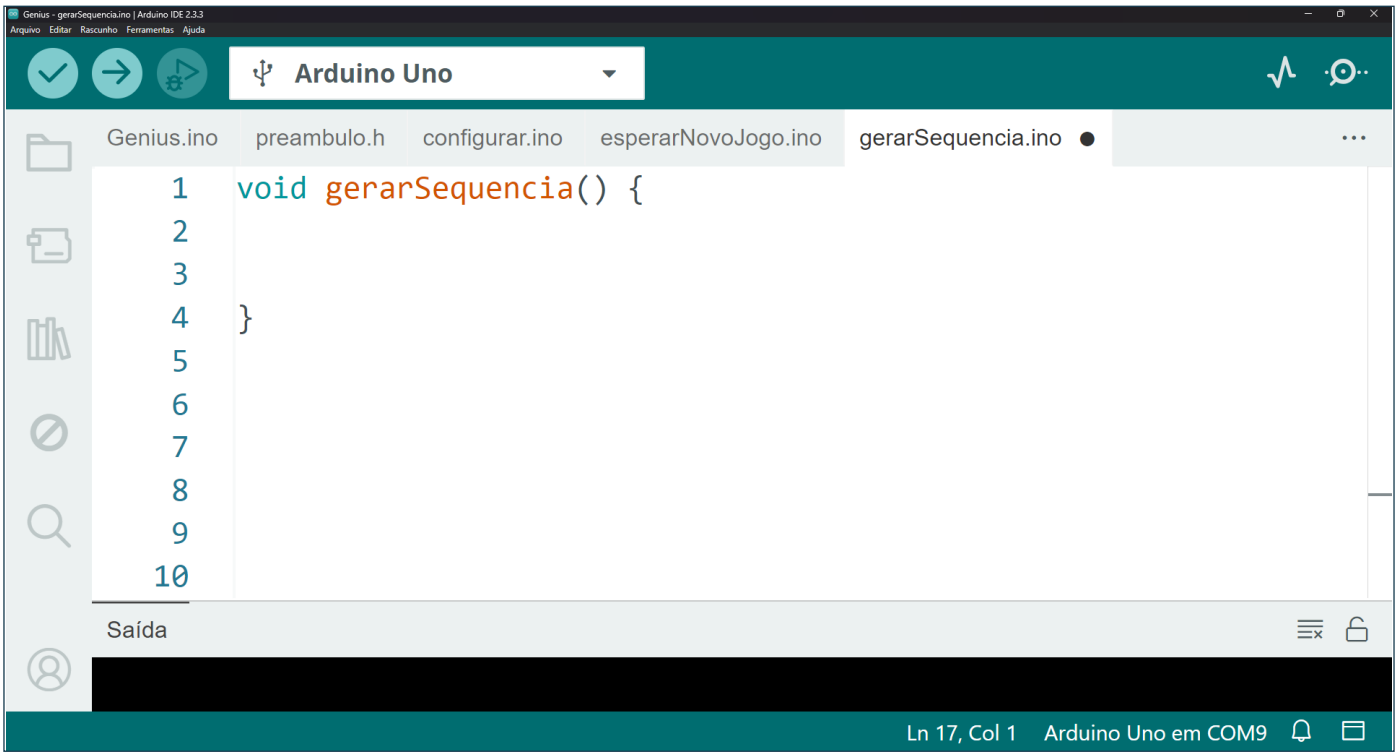
Aula 23 Genius - I

```
    if (digitalRead(botaoAmarelo) == LOW || digitalRead(botaoAzul) == LOW || digitalRead(botaoVerde) == LOW || digitalRead(botaoVermelho) == LOW) {
      /* Aguarda o botão pressionado ser solto. */
      while (digitalRead(botaoAmarelo) == LOW || digitalRead(botaoAzul) == LOW || digitalRead(botaoVerde) == LOW || digitalRead(botaoVermelho) == LOW) {};
      tamanhoSequencia = 0;  /* Reinicia o tamanho da sequência. */
      gerarSequencia();     /* Gera uma nova sequência. */
      tocarMusicaAbertura(); /* Toca a música de abertura com animação de LEDs. */
      break;
    }
  }
}
```

Porém, para termos de fato um jogo, precisamos seguir com a programação para criar, também em abas distintas, as novas funções chamadas na função do tipo void **esperarNovoJogo()**.

Abra uma nova aba do tipo **.ino** para criarmos a próxima função, **void gerarSequencia()**.

Figura 7 - Criação da aba gerarSequencia.ino para a função **void gerarSequencia()**



Fonte: Arduino IDE.

Essa função não possui muitos elementos em sua sintaxe e por ela nosso jogo Genius pode gerar e expandir a sequência de luzes e sons que o jogador deverá seguir, aumentando a dificuldade a cada novo nível.

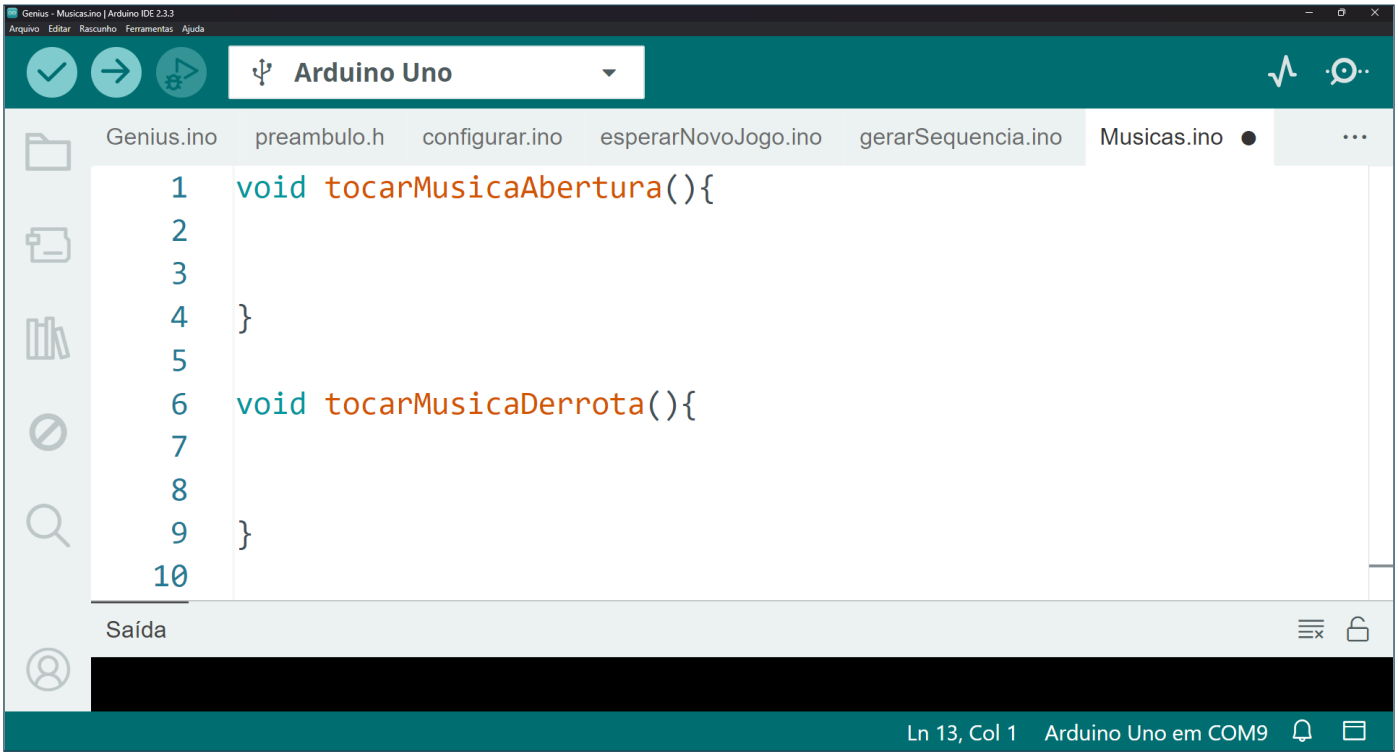
```
/* Função para gerar uma nova sequência. */
void gerarSequencia() {
  sequencia[tamanhoSequencia] = random(4); /* Gera um número aleatório entre 0 e 3.
*/
  tamanhoSequencia++;
  Serial.print("Nível: ");
  Serial.println(tamanhoSequencia); /* Imprime o nível atual no monitor serial. */
}
```

Aula 23 Genius - I

Na estrutura interna de **gerarSequencia()**, a função **random()** gera um número aleatório entre **0** e **3**, correspondendo, no contexto do Genius, aos quatro LEDs conectados em nosso protótipo (exemplo: 0 para amarelo, 1 para azul, 2 para verde, 3 para vermelho). O número aleatório gerado é armazenado no array **sequencia**, na posição **tamanhoSequencia**, a qual é incrementada, e durante o jogo esses números no array serão usados para determinar de forma randômica quais LEDs acenderão em cada passo do jogo. Por fim, com base no tamanho da sequência usada no momento, imprime no monitor serial para feedback ao jogador sobre seu nível atual no jogo.

Além dessa, a função **esperarNovoJogo()** possui internamente mais uma, que também criaremos a seguir, porém, em uma aba específica para as músicas do jogo.

Figura 8 - Criação da aba Musicas.ino para as funções **void tocarMusicaAbertura() e void tocarMusicaDerrota()**



Fonte: Arduino IDE.

Geralmente temos criado cada função em uma aba específica, porém agora veremos que é possível, em uma mesma aba **.ino**, criar duas ou mais funções, basta indicar seu tipo e seguir a mesma estrutura que adotamos nas criações anteriores, com a ação da função especificada entre **{ }**, como veremos a seguir – vamos aproveitar e já adicionar à aba **Musicas.ino**, além da função **void tocarMusicaAbertura()**, a função **void tocarMusicaDerrota()** que será usada na próxima aula.

```
/* Função para tocar a música de abertura com animação de LEDs. */
void tocarMusicaAbertura() {
  acenderLed(0);
  delay(300);
  apagarLeds();
  delay(100);
  acenderLed(1);
  delay(300);
  apagarLeds();
  delay(100);
  acenderLed(2);
  delay(300);
  apagarLeds();
  delay(100);
  acenderLed(3);
  delay(300);
  apagarLeds();
  delay(1000); /* Espera um segundo antes de iniciar a partida. */
}
```

Aula 23 Genius - I

```
/* Função para tocar a música de derrota com animação de LEDs. */
void tocarMusicaDerrota() {
  int notas[] = {330, 294, 262, 196, 150, 130, 110, 98}; /* Sequência das notas da derrota */
  int duracoes[] = {150, 150, 150, 150, 150, 150, 150, 600}; /* Duração de cada nota. */
  int leds[] = {2, 4, 6, 8}; /* Pinos dos LEDs. */
  int numLeds = sizeof(leds) / sizeof(leds[0]); /* Número de LEDs disponíveis. */
  for (int i = 0; i < 8; i++) {
    /* Tocar a nota. */
    tone(buzzer, notas[i]);
    /* Acender um LED aleatório. */
    int ledAleatorio = leds[random(numLeds)]; // Escolhe um LED aleatório
    digitalWrite(ledAleatorio, HIGH); // Acende o LED escolhido
    delay(duracoes[i]); /* Espera a duração da nota. */
    noTone(buzzer); /* Para o som da nota. */
    apagarLeds(); /* Apaga todos os LEDs. */
    delay(100); /* Pequena pausa entre as notas. */
  }
  delay(1000); /* Espera um segundo antes de permitir iniciar uma nova partida. */
}
```

Aula 23 Genius - I

O que caracterizam as duas funções na aba Musicas.ino? Ao criar os comandos para tocar as músicas, contemplaremos também o acionamento dos LEDs para acompanhar a execução.

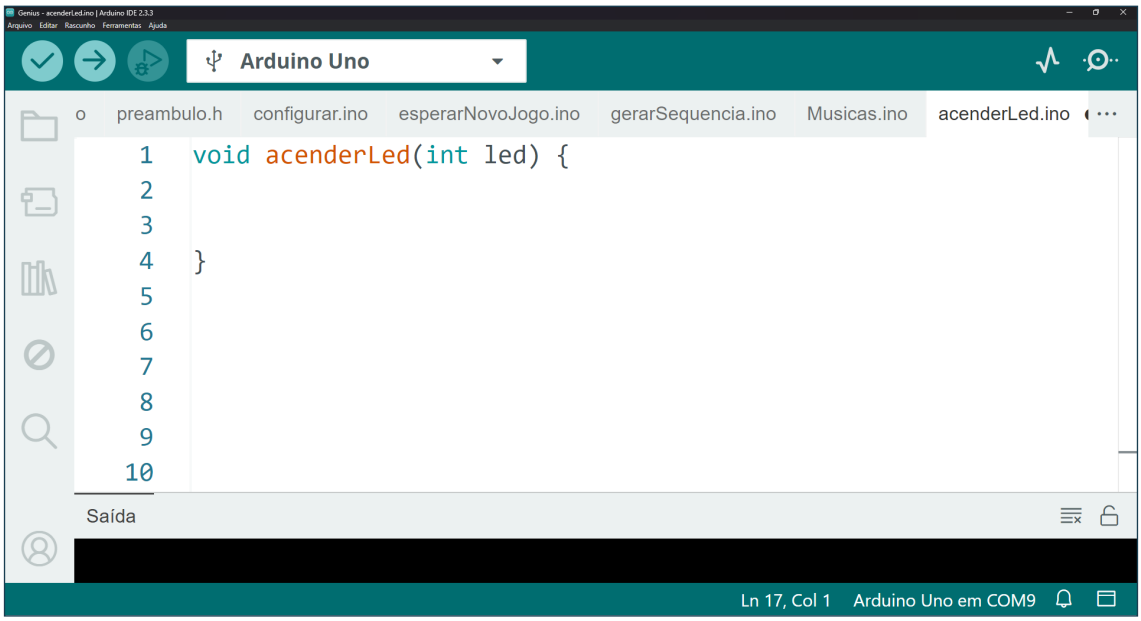
A função **void tocarMusicaAbertura()** é responsável por tocar a música de abertura e realizar uma animação com os LEDs antes do início da partida do jogo Genius. Sua estrutura possui outras duas funções também para serem criadas no nosso projeto: **acenderLed()** e **apagarLeds()**, nas quais cada LED é aceso sequencialmente pelo parâmetro indicado e, após um curto período definido pelo **delay(300)**, todos os LEDs são apagados. Por fim, após a sequência de LEDs, o código espera um segundo antes de iniciar a partida, proporcionando um breve momento de transição.

A segunda função da aba **musicas.ino**, **tocarMusicaDerrota()**, é responsável por tocar uma música específica e exibir uma animação com LEDs quando o jogador perde.

Sua estrutura difere da função anterior e possui as arrays **int notas[] = {...}** para definir a sequência das notas musicais para a música de derrota, **int duracoes[] = {...}** para definir a duração de cada nota e **int leds[] = {...}** para indicar os pinos dos LEDs que serão usados na animação. A função contém também a variável **numLeds** para calcular, pela função **sizeof()**, o número de LEDs disponíveis. Então, aplica-se o loop para tocar as notas pela estrutura de controle **for (int i = 0; i < 8; i++) { ... }**, a qual itera sobre os sons e suas respectivas durações: a função **tone(buzzer, notas[i])** toca a nota atual usando o mesmo buzzer conectado ao protótipo, enquanto a função **digitalWrite()** aciona um LED aleatório, selecionado pela função **random()**. Por fim, as funções **apagarLeds()** e **noTone()** para controle dos acionamentos. Nessa função **tocarMusicaDerrota()**, temos três delays: **delay(duracoes[i])**, para manter a nota e o LED acesos pela duração especificada, **delay(100)**, para pausa entre notas, e **delay(1000)** para, após tocar todas as notas, esperar um segundo para o jogador se preparar para a próxima tentativa, o que indica o fim do feedback de derrota.

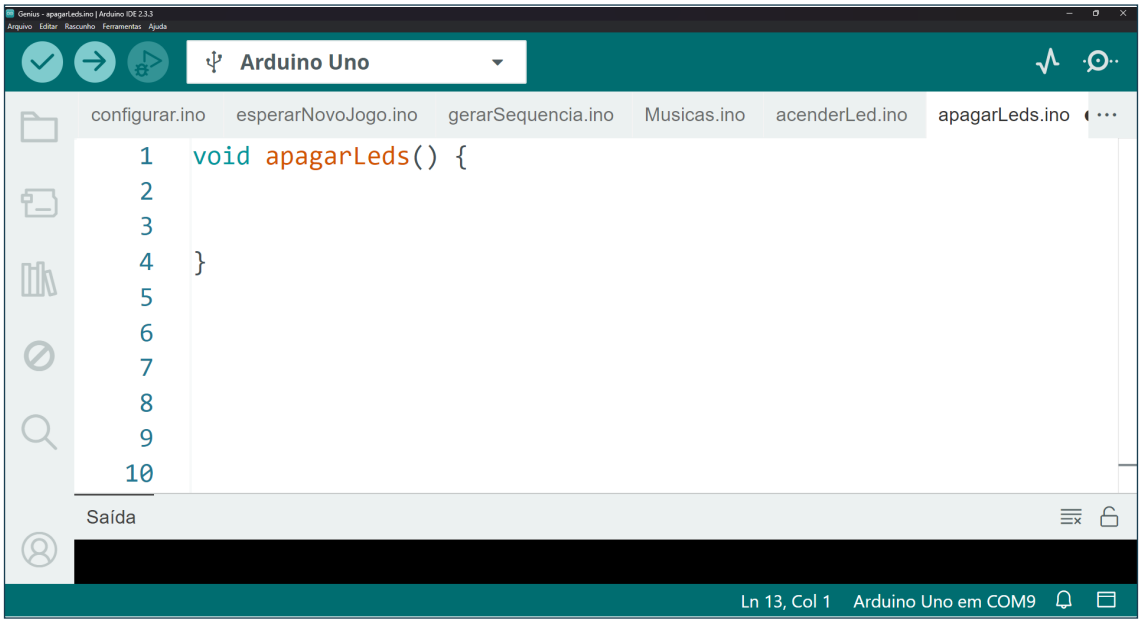
Finalizaremos a primeira parte da programação do jogo Genius com as duas últimas funções a serem criadas nesta aula, também em suas próprias abas: **acenderLed()**, com definição de seus parâmetros, e **apagarLeds()**.

Figura 9 - Criação da aba acenderLed.ino para a função **acenderLed()**



Fonte: Arduino IDE.

Figura 10 - Criação da aba apagarLeds.ino para a função **apagarLeds()**



Fonte: Arduino IDE.

Apesar do nome, a função **acenderLed(int led)** é utilizada para acender um LED específico e tocar uma nota musical correspondente, conforme definições no preâmbulo para os pinos dos LEDs e as notas. Isso é fundamental para a interface do jogo, que usa luzes e sons para sinalizar ao jogador.

```
/* Função com parâmetro para acender o LED correspondente e tocar a nota musical. */
void acenderLed(int led) {
  switch (led) {
    case 0:
      digitalWrite(ledAmarelo, HIGH);
      tone(buzzer, notaAmarelo);
      break;
    case 1:
      digitalWrite(ledAzul, HIGH);
      tone(buzzer, notaAzul);
      break;
    case 2:
      digitalWrite(ledVerde, HIGH);
      tone(buzzer, notaVerde);
      break;
    case 3:
      digitalWrite(ledVermelho, HIGH);
      tone(buzzer, notaVermelho);
      break;
  }
}
```

Aula 23 Genius - I

A função, do tipo **void**, utiliza um parâmetro do tipo **int** para determinar qual LED deve ser aceso. Os valores possíveis são **0**, **1**, **2**, e **3**, cada um correspondendo a um LED e uma nota diferente. Observe, internamente, que a função **acenderLed()** possui a estrutura **switch** que verifica o valor do parâmetro **led** e executa o bloco de código correspondente. Cada caso, de 0 a 3, acende um LED específico e toca uma nota musical correspondente usando **digitalWrite()** e **tone()**, combinando, assim, efeitos visuais e sonoros para melhorar a experiência de jogo e fornecer feedback ao jogador.

Pensando em toda estrutura do jogo, essa função é uma parte essencial do funcionamento do nosso jogo Genius, proporcionando as indicações visuais e auditivas necessárias para que os jogadores sigam a sequência correta.

E, como tudo o que acende precisa, em algum momento, ser apagado, programaremos a aba para a última função dessa aula: **apagarLeds()**.

A função **apagarLeds()** é utilizada para apagar todos os LEDs e parar qualquer som que esteja sendo emitido pelo buzzer. Isso é importante para reiniciar o estado do jogo e garantir que não haja LEDs acesos ou som contínuo após uma ação específica, como o fim de uma sequência de jogo ou quando o jogador errar.

```
/* Função para apagar todos os LEDs e parar o som. */
void apagarLeds() {
  digitalWrite(ledAmarelo, LOW);
  digitalWrite(ledAzul, LOW);
  digitalWrite(ledVerde, LOW);
  digitalWrite(ledVermelho, LOW);
  noTone(buzzer);
}
```

Confira, no sketch online da **Aula 23 – Genius [Parte I]**, como ficou o código desta aula. Na próxima, finalizaremos a programação do jogo Genius, compreendendo cada um de seus elementos para que, a cada novo projeto, você possa ter mais autonomia na criação de tantos outros!

Aula 23 Genius - I

Finalizada a programação, realizaremos também a montagem do protótipo físico para checar todas as funcionalidades dos recursos aplicados ao jogo, especialmente quanto à gravação do recorde na memória EEPROM. Assim, você e seus colegas poderão iniciar uma série com o jogo Genius na escola para verificar qual estudante, funcionário ou professor possui mais habilidades de memória para as sequências do jogo.

Dica!

Podemos experimentar uma prévia do nosso jogo pelo simulador Wokwi, porém sem o recurso da EEPROM. Acesse o protótipo Aula 23 – Genius Parte I para os primeiros testes, verificando a impressão das instruções e a execução da música de abertura com acionamento dos LEDs correspondentes às notas definidas!

Aula 23 Genius - I

Desafio:

Que tal pensar em novas melodias para o início do seu jogo, criando alternativas com outras notas musicais e tempos?

E se...

O projeto não funcionar?

- Verifique as funções criadas, com base na programação dessa aula, lembrando que por enquanto programamos o início do jogo! A sequência final será na **Aula 24 – Genius [Parte II]**.

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]**.

Eu quiser resetar o recorde armazenado no jogo Genius?

- Na aba **configurar.ino**, na qual criamos a função **configurar()**, adicione a linha a função **EEPROM.put(enderecoRecorde, 0);** e carregue-a uma vez para resetar o recorde da EEPROM.

3. Feedback e finalização

O processo de criação e desenvolvimento de jogos pode envolver diversas etapas e, assim como em uma partida, quando avançamos pelas fases e níveis, sua construção avança também!

Nesse primeiro momento do jogo Genius, fizemos as primeiras definições e programamos ajustes para início da partida e comportamento de LEDs e buzzer... confira, compartilhando seu projeto com os demais colegas, se o objetivo de programar a abertura do jogo foi alcançado e se vocês chegaram a testá-lo pelo protótipo **Aula 23 – Genius Parte I**, disponível no simulador Wokwi.

Na próxima aula, avançaremos na finalização do nosso projeto para o jogo ficar completo e suas habilidades serem desafiadas! Até lá!

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)

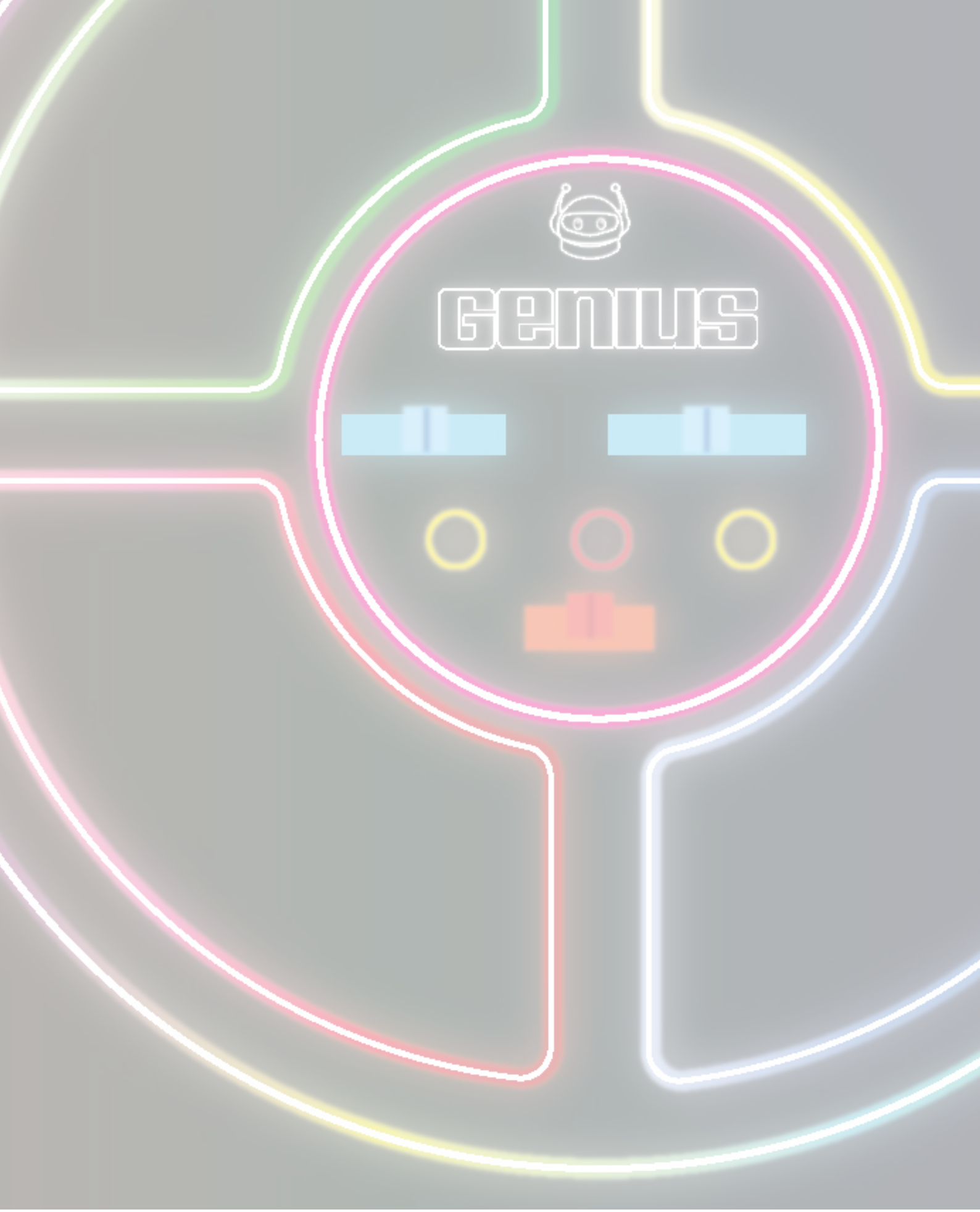
COORDENAÇÃO DE TECNOLOGIAS EDUCACIONAIS (CTE)

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