Júlia Wotzasek Pereira

Desenvolvimento e Descrição do Jogo *Mastermind* utilizando Arduino

São José dos Campos - Brasil

Maio de 2019

Júlia Wotzasek Pereira

Desenvolvimento e Descrição do Jogo *Mastermind* utilizando Arduino

Relatório apresentado à Universidade Federal de São Paulo como parte dos requisitos para aprovação na disciplina de Sistemas Embarcados.

Docente: Prof. Dr. Sérgio Ronaldo Barros dos Santos

Universidade Federal de São Paulo - UNIFESP

Instituto de Ciência e Tecnologia - Campus São José dos Campos

São José dos Campos - Brasil

Maio de 2019

Sumário

1 Descrição do Projeto

1.1 Descrição

O jogo *Mastermind*, conhecido no Brasil como jogo de Senha, é um jogo de lógica simples, e desenvolvê-lo foi o que visou esse projeto. A ideia do jogo é que o usuário deve descobrir uma senha de 4 dígitos, com os dígitos indo de 1 a 6, e que tem 10 chances para isso. A cada jogada, para saber o quão próximo se encontra do resultado correto, o jogo retorna ao jogador a resposta da seguinte forma:

- Se o jogador acertou que aquele dígito deve estar na senha, mas errou a sua posição, um *led* branco se acenderá para cada dígito nessa situação, sendo no máximo 4 *leds* brancos (ocorrendo quando o jogador acerta todos os dígitos na posição errada);
- Se o jogador acertou o dígito na posição correta, um *led* verde se acenderá para cada dígito nessa situação, com 4 deles acendendo indicando que o jogador ganhou o jogo.

O projeto foi feito para ser possível jogar sozinho ou com um desafiante. Para a versão do jogo *single player*, desenvolveu-se um algoritmo para gerar uma senha aleatória. Para a versão do jogo *multi player*, o desafiante entra com a senha no teclado.

Para a implementação, foi utilizada a placa *Arduino Mega 2560* para integração dos periféricos e para controlar a lógica do jogo no microcontrolador. Para a entrada do "chute"da senha correta pelo jogador bem como da senha do desafiante, foi utilizado um teclado matricial 4×4 de membrana. Para o retorno da quantidade de acertos utilizou-se *leds* de alto brilho e para a comunicação com o jogador, utilizou-se um *display* LCD 16×2 com módulo de comunicação I2C acoplado.

1.2 Objetivos

1.2.1 Objetivos Gerais

Por objetivo geral desse projeto tem-se assimilar os conceitos básicos de sistemas embarcados por meio da definição, desenvolvimento e implementação de um projeto utilizando o Arduino Mega 2560.

1.2.2 Objetivos Específicos

Por objetivos específicos temos:

- Desenvolvimento da lógica do jogo *Mastermind*
- Desenvolvimento de um algoritmo de senha aleatória
- Implementação do teclado matricial
- implementação dos *leds*
- implementação do *display* LCD
- implementação do *buzzer*
- implementação de modos de jogo

2 Funcionamento do Sistema

2.1 *Hardware*

2.1.1 Descrição dos Componentes

2.1.1.1 Teclado Matricial 4×4 de Membrana

O teclado matricial 4×4 de membrana consiste em 16 teclas *push-buttons* do tipo membrana dispostos na configuração da figura 1. Como podemos observar, botões de uma mesma linha são conectados entre si, correspondendo aos canais de 1 a 4. Da mesma maneira, botões de uma mesma coluna são conectados entre si, correspondendo aos canais de 5 a 8.

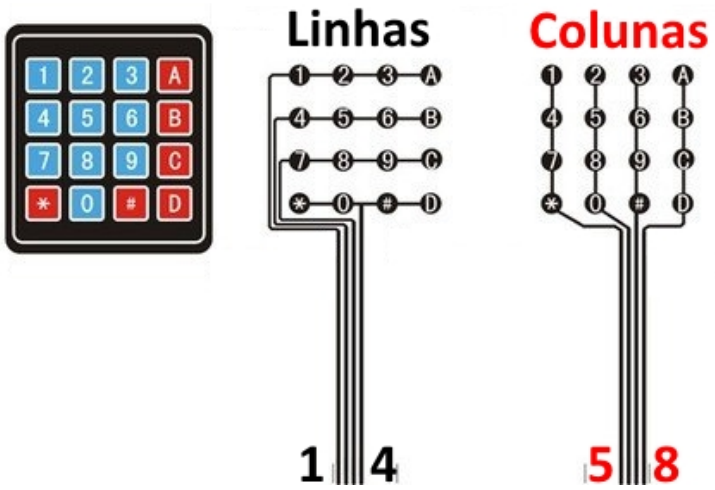

Figura 1 – Teclado Matricial de Membrana. Fonte: https://www.filipeflop.com/blog/teclado-matricial-4x4-arduino/

Para os botões do teclado, 4 estados possíveis são considerados:

1. **IDLE:** Esse é o estado de livre, sem valor. É o estado para quando o botão não está pressionado.
2. **PRESSED:** Esse é o estado para quando o botão é apertado.
3. **RELEASED:** Esse é o estado para quando o botão é solto.
4. **HOLD:** Esse é o estado para quando o botão é segurado pressionado. É interessante destacar de antemão que o que faz considerar que é nesse estado que o botão se encontra depende de quanto tempo se define como tempo de *holding*. Na biblioteca *keypad.h* esse *debounce* é feito com a função millis().

Destacamos estes estados do teclado pois eles são bastante relevantes para a explicação de como ocorre a leitura de qual botão foi apertado. O processo de leitura ocorre por varredura. A ideia do processo é a seguinte: Pelas quatro colunas de entrada, o Arduino envia um sinal digital alto ($HIGH$). Se nenhuma das teclas estiver pressionada, o sinal das quatro linhas estará baixo (LOW).

Suponha agora que apertemos, por exemplo, a tecla 8. Nesse instante, a corrente fluirá entre a coluna 2 e a linha 3, fazendo o sinal digital da coluna 2 ir para LOW. Essa mudança de estado - $(HIGH, LOW) \rightarrow (LOW, LOW)$ - faz com que se comece a fazer a varredura nas linhas, enviando um sinal digital alto ($HIGH$) linha por linha. Ainda no exemplo em que apertamos a tecla 8, quando o sinal alto é enviado para as linhas 1 e 2, nenhuma mudança de estado é alterada pois, como podemos notar em 2, os caminhos não estão conectados. Porém, quando é enviado um sinal alto para a linha 3, o sinal da coluna 2 retorna para $HIGH$ e consegue, dessa forma, determinar que o botão apertado fora o 8.

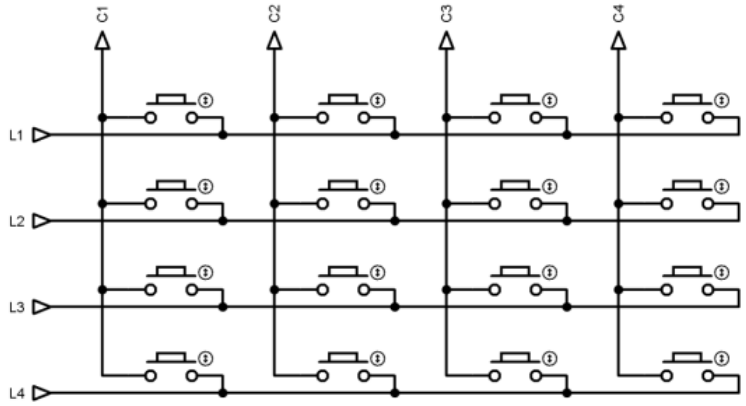

Figura 2 – Circuito de multiplexação do teclado matricial. Fonte: https://portal.vidadesilicio.com.br/teclado-matricial-e-multiplexacao/

2.1.1.2 *Display* LCD

O *display* LCD utilizado possui alimentação de $5V$, 1 pino para *backlight*, 6 pinos de dados. O *display* em questão possui 16×2. Cada célula do *display é de* 8×5 *bits*.

2.1.1.3 Módulo I2C

Como descrito em (1), o protocolo de comunicação I2C é um protocolo de comunicação serial síncrona, que utiliza apenas dois fios. É um protocolo utilizado para conectar a placa mãe a periféricos de baixa-velocidade.

O barramento I2C é composto por dois fios de comunicação bidirecional: o ***serial clock (SCL)*** e o ***serial data (SDA)***. Além disso, precisamos fornecer alguma tensão, geralmente $3,3V$ ou $5V$, e o GND. O SCL é responsável pelo *clock* do barramento, e o SDA pelos dados.

O protocolo I2C tem dois tipos de dispositivo: ***master*** e ***slave***, com o master sendo responsável pelo controle dos slaves. No caso, quando colocamos o módulo I2C acoplado ao *display* LCD, estamos tomando o LCD como slave, controlado pelo Arduino.

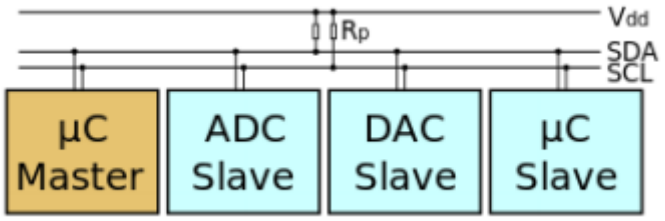

Figura 3 – Master-slave comunicação I2C. Fonte: (1)

Como se pode perceber pela figura, quando o barramento I2C está no estado neutro é mantido valor alto em ambas as linhas de comunicação. Para gerar comunicação, o SDA deve ser trazido para o valor 0 pelo mestre. Assim, para cada vez que o SCL pulsa, o SDA é lido.

O fato de termos o LCD conectado a um módulo I2C é bastante útil, pois minimiza o número de portas necessárias para conectar o *display* LCD.

2.1.1.4 *Buzzer*

O *buzzer* é um transdutor piezoelétrico, isto é, um componente que de acordo com a tensão recebida gera uma tensão mecânica. Quando enviamos essa tensão com certa frequência, fazemos com que ele vibre, o que gera sons. o componente utilizado é um alimento ativo, possuindo oscilador interno, como visto em (2).

Na figura 4, notamos que devemos conectar uma tensão (usamos $5V$, que é a da placa arduino) no V da figura. O outro conector do *buzzer* é colocado para chavear o transistor. Daí, quando conduz o *buzzer* emite som, e quando não conduz o *buzzer* não emite som.

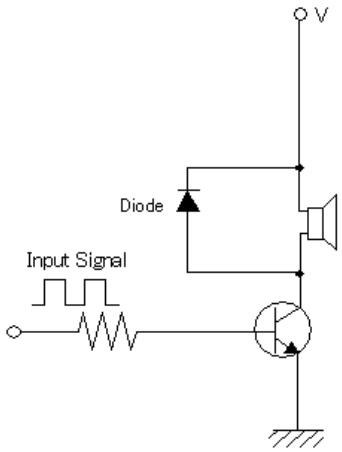

Figura 4 – Sistema elétrico de um *buzzer* Fonte: https://www.murata.com/en-us/support/faqs/products/sound/sounder/char/sch0007

2.1.1.5 *Led* de alto brilho

Um *led*, ou *Light Emissor Diode*, é um diodo emissor de luz, isto é, um componente eletrônico feito de germânio ou silício, que conduz apenas corrente no sentido da polarização, onde o catodo deve ser positivo e o anodo negativo. O Led, em particular é um diodo que, quando passa corrente, emite luz, como explicado em (3).

O *led* é um componente bipolar, e a forma como está polarizado determina o sentido que a corrente pode passar, como está indicado na figura 5.

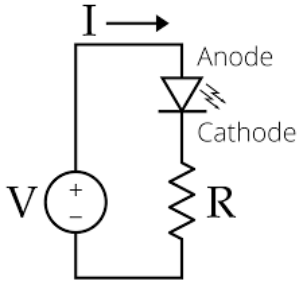

Figura 5 – Esquemático elétrico de um *led*

Existem vários tipos de *led*. Dentre eles, destacamos:

- ***led* difuso comum:** Em *leds* desse tipo, a luz é espalhada por toda a cápsula de plástico, buscando distribuição uniforme desta.
- ***led* de alto brilho:** A potência luminosa de *leds* de alto brilho é bastante superior a dos *leds* difusos, aumentando a luminosidade deste.

Utilizamos *leds* de alto brilho nesse projeto, nas cores verde e branco. O l*led* utilizado foi o de $5mm$, de $3,3V$ e de $30mA$. Como a tensão utilizada do arduino foi a $5V$,

seria interessante que utilizássemos um resistor de 62Ω. Porém, utilizamos um resistor de 330Ω, que era o menor que tínhamos à disposição, que diminui um pouco o brilho, mas isso para o projeto não surtiu efeito significativo.

2.1.2 Integração do *hardware*

2.1.2.1 Conexão *display* LCD - Arduino

Para a conexão do *display* com módulo I2C, conectamos VCC e GND nos respectivos da placa por meio do *protoboard*, e conectamos o SCL no pino 21 do Arduino (pino de comunicação SCL) e conectamos o SDA no pino 20 (pino de comunicação SCA).

2.1.2.2 Conexão *led* - Arduino

Para a conexão dos *leds* de alto brilho o colocamos com a polaridade negativa no GND e a positiva em uma entrada digital do Arduino, intermediada pelo resistor de 330Ω para controle da corrente. Para os leds, utilizamos as portas $\{22, 24, 26, 28, 36, 38, 40, 42\}$.

2.1.2.3 Conexão *buzzer* - Arduino

Para a conexão do *buzzer*, colocamos a polaridade negativa no GND e a positiva em uma entrada analógica, no caso a porta 8.

2.1.2.4 Conexão teclado matricial - Arduino

Para conexão do teclado matricial, utilizamos as entradas de 44 até 53, como sinais digitais para a leitura do teclado como já explicado anteriormente.

2.1.3 Esquemático do Circuito

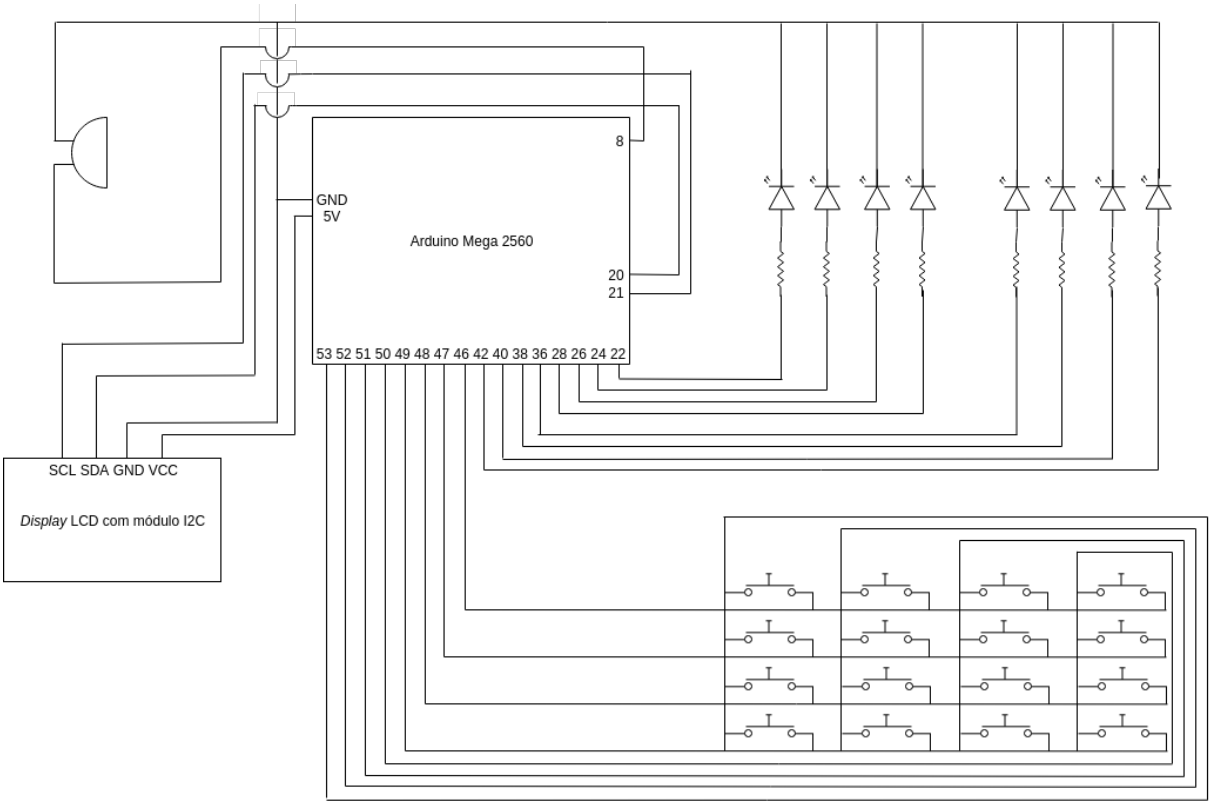

Figura 6 – Esquemático elétrico do projeto

2.2 *Software*

Por critério de organização do presente relatório, iremos apresentar parte a parte do código, dividido em 4 grandes partes: as bibliotecas e sua comunicação com o *hardware*, a inicialização do jogo, a manutenção do jogo e o término do jogo.

2.2.1 Bibliotecas utilizadas

As bibliotecas feitas para o Arduino visam facilitar a utilização de periféricos e dispositivos resolvendo como obter/enviar sinais. Desta maneira, vamos discutir um pouco sobre como as bibliotecas utilizadas funcionam e as funções que utilizamos.

2.2.1.1 Keypad.h

A biblioteca **Keypad.h** é para a utilização do teclado matricial. Na descrição sobre o teclado matricial discutimos que a leitura das teclas pressionadas é feita por varredura. Essa biblioteca esconde esse método na sua função **getKey()**. Ela testa se houve mudança de estado e se houve alguma atividade no teclado pela função **getKeys()**, que é a função que verifica se mais de um botão foi apertado, e com isso varre o teclado para a obtenção do que foi teclado, como precisávamos. Nesse projeto, só utilizamos a função getKey() e o

construtor do teclado, no qual passamos o tamanho do teclado e quais os dígitos desse teclado.

2.2.1.2 Wire.h

A biblioteca **Wire.h** serve para comunicação com I2C/TWI.Ela lida com os sinais de SCL e SDA, com funções de recebimento e envio e dados.

2.2.1.3 LiquidCrystal_I2C.h

A biblioteca **LiquidCrystal_I2C.h** lida com *display* LCD quando acoplado a um módulo I2C. Logo, objetiva-se passar pelo SDA os dados necessários para ter o mesmo efeito que teríamos se utilizássemos um *display* normal.

Voltado a esse fato, precisamos de um construtor que recebe todos os dados necessários de do *display* para construí-lo. Utilizamos como parâmetros do construtor o endereço de lcd, o número de linhas e o número de colunas.

Além disso, precisamos inicializar o *display*. Para isso utilizamos a função *init()* para inicializar o *display*. Em relação a biblioteca para lcd direto, utilizaríamos o begin nesse lugar.

Outra função que utilizamos é a de limpar a tela, que é a **lcd.clear()** para apagar o conteúdo da tela. Utilizamos também função de **setCursor** para dizer onde desejamos começar a escrever no *display*.

2.2.2 Cabeçalho do Programa

2.2.2.1 Bibliotecas Utilizadas

```
1 /* Bibliotecas Utilizadas */
2 # include <Wire.h> // Comunicacao I2C
3 #include <LiquidCrystal_I2C.h> // Comunicacao com o display lcd
4 #include <Keypad.h> // teclado matricial
```

Como já discutimos antes, importamos as bibliotecas Wire, LiquidCrystal_I2C e Keypad para esse projeto.

2.2.2.2 Definição da Pinagem

```
1 /* Definicao das constantes para as portas dos leds */
2 const int ledGreen1 = 22;
3 const int ledGreen2 = 24;
4 const int ledGreen3 = 26;
5 const int ledGreen4 = 28;
6
```

```
const int ledWhite1 = 36;
const int ledWhite2 = 38;
const int ledWhite3 = 40;
const int ledWhite4 = 42;

/* Definicao das constantes para a porta do buzzer*/
const int buzzer = 8;
const int melodyPin = 8;


// Inicializacoes do teclado
const byte linhas = 4; //4 linhas
const byte colunas = 4; //4 colunas

// Definicao dos valores das portas onde esta conectado o teclado
    matricial
byte pinoslinhas[linhas] = {46,47,48,49}; //pinos utilizados nas
   linhas
byte pinoscolunas[colunas] = {50,51,52,53}; //pinos utilizados
   nas colunas

//teclado matricial
char matrizteclado[linhas][colunas] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
```

Essa parte do código serve para apenas definir quais portas do Arduino serão utilizadas para cada uma das entradas/saídas necessárias. Destacamos nessa parte a declaração do teclado matricial como uma matriz de char, o que facilita o retorno à leitura dos dígitos.

2.2.2.3 Variáveis do Jogo

```
//inicializando o teclado
Keypad teclado = Keypad( makeKeymap(matrizteclado), pinoslinhas,
   pinoscolunas, linhas, colunas );

/* Instanciacao do display lcd */
LiquidCrystal_I2C lcd(0x27,16,2);

```

```
/* Variaveis de controle do jogo */
int senhaParam[4] = {1,3,2,1};
int senhaDigitada[4];
int contadorReset = 3;
int verdes = 0;
int brancos = 0;
int contadorJogadas = 9;
int contadorJogadasReset = 9;
int tom;
int sorteadorSenha = 1000;
```

Nesse trecho estamos apenas construindo variáveis. Vale destacar que utilizamos os construtores das bibliotecas Keypad e do *display* de LCD para utilizá-los. Essas duas variáveis são do *hardware*.

Olhando para as variáveis de controle do jogo, a ***senhaParam[]*** possui valor inicial para a construção das senhas aleatórias da versão *single player* (veremos essa função posteriormente). a variável ***sorteadorSenha*** é um controle sobre qual iteração da escolha de senhas o programa está, e optamos por começar da milésima iteração do algoritmo.

As variáveis ***verdes*** e ***brancos*** servem para controlar o número de *leds* que vão acender a cada resposta de jogada.

2.2.3 Definição da música de vitória

```
/* Constantes para tocar a musica de vitoria */
#define NOTE_G6  1568
#define NOTE_C7  2093
#define NOTE_E7  2637
#define NOTE_G7  3136

/* Definicao do vetor de notas da musica de vitoria */
int melody[] = {
  NOTE_E7, NOTE_E7, 0, NOTE_E7,
  0, NOTE_C7, NOTE_E7, 0,
  NOTE_G7, 0, 0,  0,
  NOTE_G6, 0, 0, 0,
};

/* Definicao do vetor de tempos das notas da musica de vitoria */
int tempo[] = {
  12, 12, 12, 12,
  12, 12, 12, 12,
  12, 12, 12, 12,
```

```
  12, 12, 12, 12,
};
```

As definições feitas são para mandar a nota correta para o*buzzer*. Dado isso, definimos dois vetores: um que possui as notas que devem ser tocadas e outro que oferece os tempos para cada nota.

2.2.3.1 *Setup*

```
void setup() {
  //Leds de resposta
  pinMode(ledGreen1,OUTPUT);
  pinMode(ledGreen2,OUTPUT);
  pinMode(ledGreen3,OUTPUT);
  pinMode(ledGreen4,OUTPUT);

  //Leds de resposta
  pinMode(ledWhite1,OUTPUT);
  pinMode(ledWhite2,OUTPUT);
  pinMode(ledWhite3,OUTPUT);
  pinMode(ledWhite4,OUTPUT);

  // Buzzer
  pinMode(buzzer,OUTPUT);

  //Define o numero de colunas e linhas do LCD
  lcd.init();
  lcd.setBacklight(HIGH);

  iniciaJogo();
}
```

O **setup** do programa é bem simples também. A ele cabe apenas indicar que todos os *leds* são canais de saída como o *buzzer* e ligar o *display*. Feito isso e todas as configurações básicas do projeto estão prontas, chamando assim a função **icicia**Jogo(). Para o lcd precisamos utilizar as funções **lcd.init()** para incializar o *display* e colocamos a luz do *display* ligada com a função **lcd.setBacklight()**.

2.2.4 Lógica do jogo - Ciclo principal

```
void loop(){
  // Jogadas ainda sao aceitas
  boolean ganhou = false;
```

```
4   if(contadorJogadas >= 0 || ganhou){ // se ainda tem tentativas
        ou ganhou
5     recebeChute(); // recebe a tentativa de senha do usuario
6     ganhou = confereSenha(); // confere se a senha esta correta
7     if(!ganhou){ // se nao ganhou
8       respondeJogada(); // imprime nos leds a resposta colorida
9       contadorJogadas--; // informa que foi mais uma rodada.
10    } else {
11      ganhaJogo(); // chama a funcao de ganhar o jogo
12      iniciaJogo(); // inicia o jogo outra vez
13    }
14  } else {
15    perdeJogo(); // perde o jogo
16    iniciaJogo(); // inicia o jogo outra vez
17  }
18 }
```

O *loop* principal é onde concentramos a lógica geral do jogo. A ideia é que temos uma variável booleana que controla se o jogador já ganhou o jogo e se ainda tem jogadas. Começa recebendo o chute do jogador, e depois testando se este é a senha. Se não for, ele responde as jogadas nos *leds* e decrementa a quantidade de jogadas ainda permitidas. Caso o jogador tenha acertado a senha, chama-se a função de ganhar o jogo e o reinicia.

Se passarem as 10 tentativas, então a função de perder o jogo é chamada e o jogo reiniciado da mesma forma.

Em forma de diagrama, podemos representar:

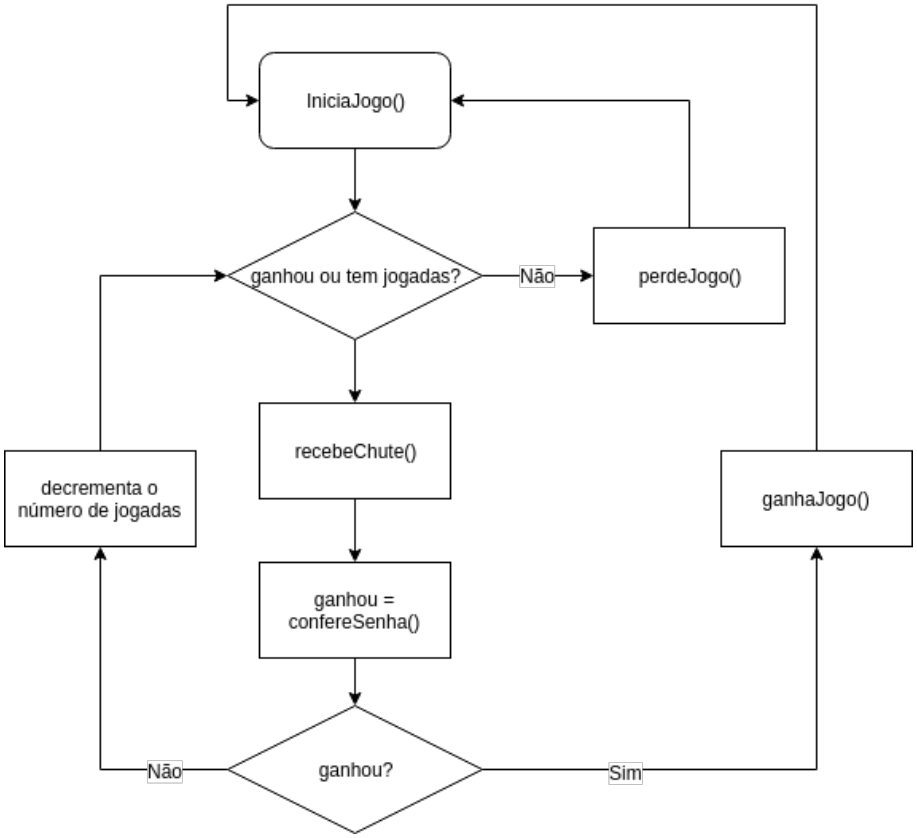

Figura 7 – Fluxograma da lógica principal do jogo

2.2.5 Inicialização do Jogo

```
void iniciaJogo(){
  int modo = selecionaModo();
  if(modo == 2){
    leSenha();
  } else {
    sorteiaSenha(); // sorteia a senha
  }
  contadorJogadas = contadorJogadasReset; // inicializa o
     contador de jogadas no maximo
  //Limpa a tela
  lcd.clear();
  //Posiciona o cursor na coluna 3, linha 0;
  lcd.setCursor(3, 0);
  //Envia o texto entre aspas para o LCD
  lcd.print("MASTERMIND");
  lcd.setCursor(1, 1);
  lcd.print("Seja bem-vindo!");
  delay(5000);
```

```
18    lcd.clear();
19    lcd.setCursor(0,0);
20    lcd.print("Sua missao, se");
21    lcd.setCursor(0,1);
22    lcd.print("aceitar, eh des-");
23    delay(2500);
24    lcd.clear();
25    lcd.setCursor(0,0);
26    lcd.print("aceitar, eh des-");
27    lcd.setCursor(0,1);
28    lcd.print("cobrir a senha");
29    delay(2500);
30    lcd.clear();
31    lcd.setCursor(2,0);
32    lcd.print("Voce tem 10");
33    lcd.setCursor(4,1);
34    lcd.print("chances");
35    delay(2500);
36    lcd.clear();
37    lcd.setCursor(0,0);
38    lcd.print("As senhas sao");
39    lcd.setCursor(3,1);
40    lcd.print(" de 4 digitos");
41    delay(2500);
42    lcd.clear();
43    lcd.setCursor(1,0);
44    lcd.print("Os digitos vao");
45    lcd.setCursor(4,1);
46    lcd.print("de 1 a 6");
47    delay(2500);
48    lcd.clear();
49    lcd.setCursor(3,0);
50    lcd.print("BOA SORTE!");
51    lcd.setCursor(3,1);
52    lcd.print("MASTERMIND");
53    delay(2500);
54 }
```

A inicialização do jogo começa recebendo o modo de jogo, para saber se é *single player* ou *multi player*. Com esse resultado, se for *single player*, o programa sorteia uma senha. Se for *multi player*, o sistema lê uma senha do teclado matricial. O resto da função é apenas para escrever o prólogo do jogo no *display* LCD. Utiliza-se a função

lcd.setCursor(x,y) para dizer onde se deve escrever a mensagem de **lcd.print("s")**, e a função **lcd.clear()** para limpar o *display*.

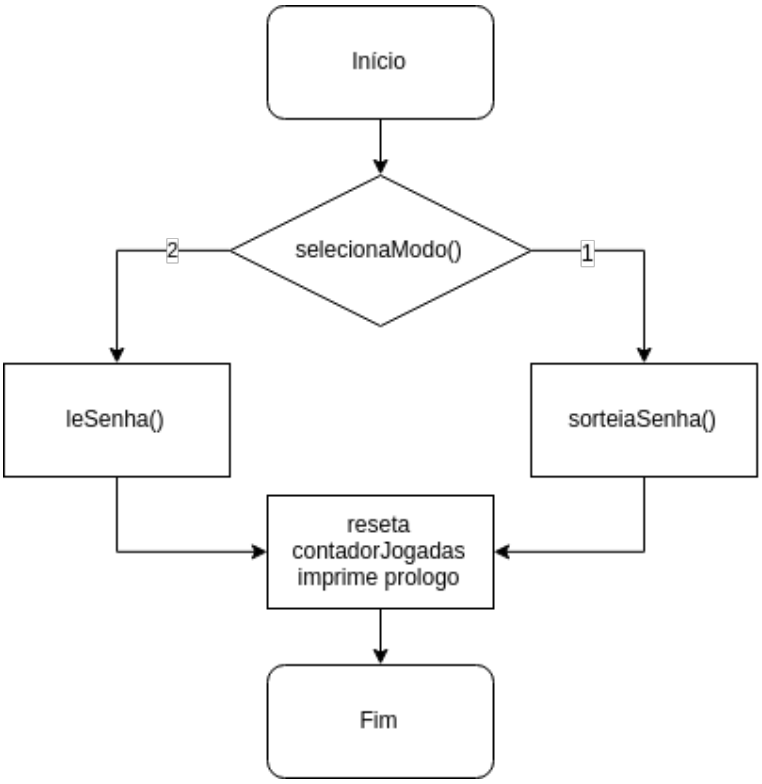

Figura 8 – Fluxograma do iniciaJogo()

2.2.5.1 Seleção do Modo de Jogo

```
int selecionaModo(){
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("SinglePlayer?(1)");
  lcd.setCursor(0,1);
  lcd.print("MultiPlayer? (2)");
  int digito = leDigito();
  while(digito != 1 && digito != 2){
    digito = leDigito();
  }
  return digito;
}
```

A função de seleção de modo permanece em *loop* nela mesma até que um dígito 1 ou 2 seja digitado. Ao ler o dígito, retorna para a função de início do jogo.

2.2.5.2 Sorteio de Senha para o caso *Single player*

```
void sorteiaSenha(){
```

```
2   int senha = 0;
3   for(int i = 0;i < 4;i++){
4     senha = senha * 10 + senhaParam[i];
5   }
6   for(int i = 0;i < sorteadorSenha;i++){
7     senha = nextSenha(senha);
8   }
9   senhaParam[0] = senha/1000;
10  senhaParam[1] = (senha/100)%10;
11  senhaParam[2] = (senha/10)%10%10;
12  senhaParam[3] = senha%1000%100%10;
13  for(int i = 0;i < 4;i++){
14    if(senhaParam[i] > 6){
15      senhaParam[i] = senhaParam[i]%7;
16    } else if(senhaParam[i] == 0){
17      senhaParam[i] = i;
18    }
19  }
20  sorteadorSenha++;
21 }
```

Para fazer a senha aleatória, decidimos criar a senha seguinte a partir da última senha digitada (justificando termos iniciado a senha como 1321). A lógica é que a senha será atualizada a quantidade de vezes que o sorteadorSenha valer. Destacamos as linhas de 17 a 21 que servem para garantir que os dígitos da senha sorteada fiquem entre 1 e 6.

```
1  int nextSenha(int senhaAtual){
2    senhaAtual = 3 * senhaAtual - 1;
3    while(senhaAtual > 9999){
4      senhaAtual = senhaAtual / 2;
5    }
6    while(senhaAtual < 1000){
7      senhaAtual = senhaAtual * 2 ;
8    }
9    return senhaAtual;
10 }
```

A função criada multiplica a senha por três e subtrai 1. Se a senha tiver mais de 4 dígitos, divide por 2; se tiver menos de 4 dígitos, multiplica por 2.

2.2.5.3 Leitura da Senha do Teclado

```
1  void leSenha(){
2    int contador = contadorReset;
```

```
3   int digito;
4   lcd.clear();
5   lcd.setCursor(0,0);
6   lcd.print("Proponha a senha");
7   lcd.setCursor(0,1);
8   lcd.print("Digitos de 1 a 6");
9   delay(3000);
10  lcd.clear();
11  lcd.setCursor(0,0);
12  lcd.print("Proponha a senha");
13  lcd.setCursor(6,1);
14  while(contador >= 0){
15    digito = leDigito();
16    if(aceitaDigito(digito)){
17      senhaParam[contadorReset - contador] = digito;
18      contador--;
19      lcd.print("*");
20    }
21  }
22  delay(2000);
23 }
```

Novamente utilizamos a função leDigito() para ler do teclado, mas só aceitamos quando a aceitaDigito() permite. A aceitaDigito é um condicional if, não tendo importância para ser exposto aqui. A leDigito(), por outro lado, é bastante interessante de ser discutida, por ser a que lida com o teclado matricial.

2.2.5.4 Leitura de Dígito

```
1  int leDigito(){
2    char digito = teclado.getKey();
3      while(!digito){
4        digito = teclado.getKey();
5      }
6    int d = (int)digito;
7    tom = 420 + 100 * (d - 48);
8    digitalWrite(buzzer,HIGH);
9    delayMicroseconds(tom);
10   delay(100);
11   digitalWrite(buzzer,LOW);
12   delayMicroseconds(tom);
13   delay(100);
14   return d - 48;
```

```
15 }
```

Para a leitura do dígito do teclado matricial, por intermédio das funções da biblioteca Keypad.h, utilizamos a função getKey(), sendo teclado o nome da variável que criamos no cabeçalho para o tipo Keypad. Para que o *buzzer* apite toda vez que um botão for apertado, quando o digito recebe algo e sai do laço while, mandamos um sinal alto para o *buzzer*, com a função de fazê-lo apitar.

2.2.6 Manutenção do Jogo

Como mostramos no fluxograma da figura 7, a manutenção do jogo é o processo de testar se o jogo acabou, seja pelo número de tentativas ou porque o ganhou, então avisar da próxima jogada, receber o próximo chute caso o jogador tenha errado, conferir a senha e novamente testar se ganhou. A função de recebeChute() é igual a função de leSenha(), com a diferença de que recebeChute() salva a senha recebida no vetor senhaDigitada para poder compará-la com a senhaParam. Assim, não a exporemos aqui.

2.2.6.1 Conferimento da Senha

```
1  boolean confereSenha(){
2    int controleDigitada[4];
3    int controleParametro[4];
4    brancos = 0;
5    verdes = 0;
6    for(int i = 0;i < 4;i++){
7      controleDigitada[i] = 0;
8      controleParametro[i] = 0;
9    }
10   // testa as posicoes corretas
11   for(int i = 0;i < 4;i++){
12     if(senhaDigitada[i] == senhaParam[i]){
13       verdes++;
14       controleDigitada[i] = 1;
15       controleParametro[i] = 1;
16     }
17   }
18   if(verdes == 4){
19     return true;
20   }else{
21    for(int i = 0;i < 4;i++){
22     if(controleDigitada[i] != 1){
23       for(int j = 0; j < 4;j++){
24         if(controleParametro[j] != 1){
```

```
25            if(senhaParam[i] == senhaDigitada[j]){
26              brancos++;
27              controleParametro[j] = 1;
28              break;
29            }
30          }
31        }
32      }
33    }
34  }
35  return false;
36 }
```

Em termos de iteração com o *hardware*, essa função é irrelevante. Porém, para o jogo ela é a parte mais crucial, pois é ela que define a resposta de quantos e como foram os seus acertos e se o jogo foi ganho.

O objetivo dessa senha é modificar o número de pinos verdes para assumir o valor da quantidade de dígitos nas posições corretas que foram chutados, e modificar o número de pinos brancos para assumir o valor da quantidade de dígitos que de fato estão na senha, mas na posição incorreta. Destacamos que não é trivial verificar isso. Precisamos testar as posições corretas e depois, colocando um FLAG de "já visitada"para as que estavam em posição correta, podendo comparar para os pinos brancos apenas as posições livres.

2.2.6.2 Resposta luminosa à jogada

```
1  void respondeJogada(){
2    if(verdes == 0){
3      digitalWrite(ledGreen1,LOW);
4      digitalWrite(ledGreen2,LOW);
5      digitalWrite(ledGreen3,LOW);
6      digitalWrite(ledGreen4,LOW);
7    } else if(verdes == 1){
8      digitalWrite(ledGreen1,HIGH);
9      digitalWrite(ledGreen2,LOW);
10     digitalWrite(ledGreen3,LOW);
11     digitalWrite(ledGreen4,LOW);
12   } else if(verdes == 2){
13     digitalWrite(ledGreen1,HIGH);
14     digitalWrite(ledGreen2,HIGH);
15     digitalWrite(ledGreen3,LOW);
16     digitalWrite(ledGreen4,LOW);
17   } else if(verdes == 3){
```

```
18      digitalWrite(ledGreen1,HIGH);
19      digitalWrite(ledGreen2,HIGH);
20      digitalWrite(ledGreen3,HIGH);
21      digitalWrite(ledGreen4,LOW);
22    }
23
24    if(brancos == 0){
25      digitalWrite(ledWhite1,LOW);
26      digitalWrite(ledWhite2,LOW);
27      digitalWrite(ledWhite3,LOW);
28      digitalWrite(ledWhite4,LOW);
29    } else if(brancos == 1){
30      digitalWrite(ledWhite1,HIGH);
31      digitalWrite(ledWhite2,LOW);
32      digitalWrite(ledWhite3,LOW);
33      digitalWrite(ledWhite4,LOW);
34    } else if(brancos == 2){
35      digitalWrite(ledWhite1,HIGH);
36      digitalWrite(ledWhite2,HIGH);
37      digitalWrite(ledWhite3,LOW);
38      digitalWrite(ledWhite4,LOW);
39    } else if(brancos == 3){
40      digitalWrite(ledWhite1,HIGH);
41      digitalWrite(ledWhite2,HIGH);
42      digitalWrite(ledWhite3,HIGH);
43      digitalWrite(ledWhite4,LOW);
44    } else if(brancos == 4) {
45      digitalWrite(ledWhite1,HIGH);
46      digitalWrite(ledWhite2,HIGH);
47      digitalWrite(ledWhite3,HIGH);
48      digitalWrite(ledWhite4,HIGH);
49    }
50  }
```

A função de resposta da jogada tem apenas a função te integrar o resultado da lógica do jogo com a interface física do jogo. Ela acende a quantidade de *leds* de cada tipo - verde e branco - utilizando a função digitalWrite() para isso. Essa função recebe a porta e qual o sinal digital que quer enviar - HIGH ou LOW.

2.2.7 Finalização do Jogo

2.2.7.1 Ganhou o Jogo

```
void ganhaJogo(){
  lcd.clear();
  lcd.setCursor(3,0);
  lcd.print("ACERTOU!!!");
  lcd.setCursor(2,1);
  lcd.print("Pontuacao:");
  lcd.print(contadorJogadas + 1);
  sing();
}
```

A função de ganhaJogo() imprime que o usuário ganhou e chama a função sing(), que é onde será tocada a música de vitória.

```
void sing() {
  int size = sizeof(melody) / sizeof(int);
  for (int thisNote = 0; thisNote < size; thisNote++) {
    int noteDuration = 1000 / tempo[thisNote];
    buzz(melodyPin, melody[thisNote], noteDuration);
    int pauseBetweenNotes = noteDuration * 1.30;
    delay(pauseBetweenNotes);
    buzz(melodyPin, 0, noteDuration);
  }
}
```

A função sing() é bastante interessante. Ela começa contando o número de notas e chama a função buzz() para tocá-las. Além disso, para cada nota ela dá um pausa com a função delay().

```
void buzz(int targetPin, long frequency, long length) {
  long delayValue = 1000000 / frequency / 2; // calcula o delay
     entre as transicoes
  long numCycles = frequency * length / 1000; // calcula o numero
      de ciclos
  for (long i = 0; i < numCycles; i++) { // calcula o tempo

    digitalWrite(ledGreen1,HIGH);
    digitalWrite(ledGreen2,HIGH);
    digitalWrite(ledGreen3,HIGH);
    digitalWrite(ledGreen4,HIGH);

    digitalWrite(ledWhite1,HIGH);
    digitalWrite(ledWhite2,HIGH);
    digitalWrite(ledWhite3,HIGH);
    digitalWrite(ledWhite4,HIGH);
```

```
15
16      digitalWrite(targetPin,HIGH);
17      delayMicroseconds(delayValue); // espera pelo tempo do delay
            calculado
18
19      digitalWrite(ledGreen1,LOW);
20      digitalWrite(ledGreen2,LOW);
21      digitalWrite(ledGreen3,LOW);
22      digitalWrite(ledGreen4,LOW);
23      digitalWrite(targetPin,LOW);
24
25      digitalWrite(ledWhite1,LOW);
26      digitalWrite(ledWhite2,LOW);
27      digitalWrite(ledWhite3,LOW);
28      digitalWrite(ledWhite4,LOW);
29
30
31      delayMicroseconds(delayValue); // espera novamente pelo tempo
            de delay calculado
32    }
33    digitalWrite(13, LOW);
34  }
```

Finalmente, a função de buzz() faz o buzzer apitar com digitalWrite() para garantir que ele apite no comprimento de onda correto. Além disso, a função faz com que os *leds* pisquem com a mesma frequência das notas.

2.2.7.2 Perdeu o Jogo

```
1  void perdeJogo(){
2    lcd.clear();
3    lcd.setCursor(4,0);
4    lcd.print("PERDEU");
5    lcd.setCursor(3,1);
6    lcd.print("GAME OVER");
7    #define tempo 10
8    int frequencia = 0;
9    for (frequencia = 150; frequencia < 1800; frequencia += 1){
10     tone(buzzer, frequencia, tempo);
11     digitalWrite(ledGreen1,HIGH);
12     digitalWrite(ledGreen2,HIGH);
13     digitalWrite(ledGreen3,HIGH);
14     digitalWrite(ledGreen4,HIGH);
```

```
15
16      digitalWrite(ledWhite1,HIGH);
17      digitalWrite(ledWhite2,HIGH);
18      digitalWrite(ledWhite3,HIGH);
19      digitalWrite(ledWhite4,HIGH);
20      delay(1);
21    }
22    for (frequencia = 1800; frequencia > 150; frequencia -= 1){
23      tone(buzzer, frequencia, tempo);
24      digitalWrite(ledGreen1,HIGH);
25      digitalWrite(ledGreen2,HIGH);
26      digitalWrite(ledGreen3,HIGH);
27      digitalWrite(ledGreen4,HIGH);
28
29      digitalWrite(ledWhite1,HIGH);
30      digitalWrite(ledWhite2,HIGH);
31      digitalWrite(ledWhite3,HIGH);
32      digitalWrite(ledWhite4,HIGH);
33      delay(1);
34    }
35      digitalWrite(ledGreen1,LOW);
36      digitalWrite(ledGreen2,LOW);
37      digitalWrite(ledGreen3,LOW);
38      digitalWrite(ledGreen4,LOW);
39
40      digitalWrite(ledWhite1,LOW);
41      digitalWrite(ledWhite2,LOW);
42      digitalWrite(ledWhite3,LOW);
43      digitalWrite(ledWhite4,LOW);
44
45    delay(1000);
46  }
```

A função de perdeJogo() apita uma sirene com o *buzzer*. Porém, apesar da música ser distinta da música da vitória, o resto do código é bastante semelhante.

2.2.8 Fluxograma Geral do Jogo

Vamos mostrar fluxograma agora completo. Dividimos a imagem em caixas relativas a parte do jogo a que pertencem A caixa roxa é para inicialização, a caixa vermelha é para manutenção do jogo e a verde é para a finalização do jogo.

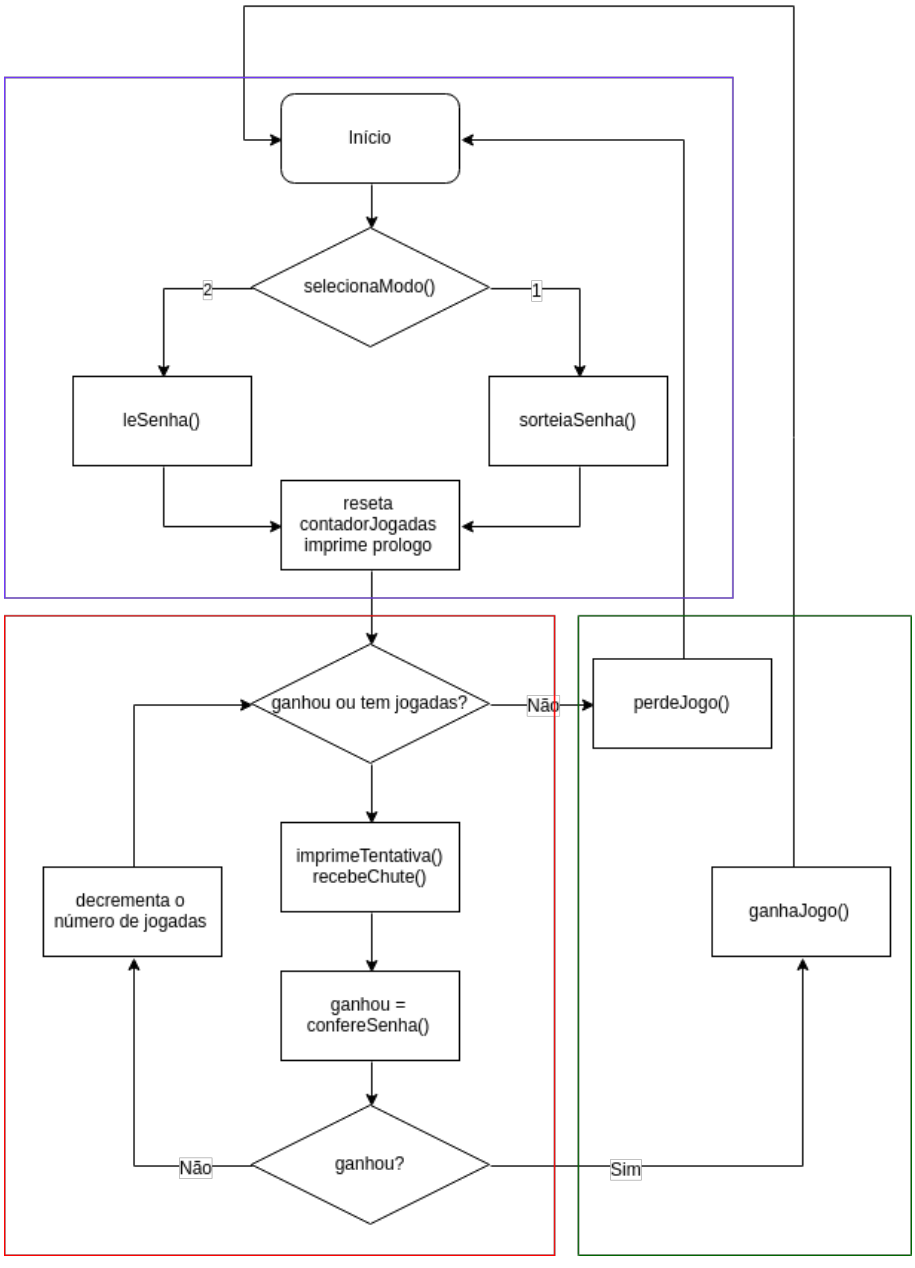

Figura 9 – Fluxograma do *Mastermind*

3 Componentes do Projeto

3.1 Lista de componentes utilizados

- 1 placa Arduino Mega 2560;
- 1 teclado matricial 4×4 de membrana;
- 1 *display* LCD de 16×2;
- 1 módulo I2C, acoplado ao *display* LCD;
- 4 *leds* de alto brilho na cor branca;
- 4 *leds* de alto brilho na cor verde;
- 1 *buzzer* de $5V$ passivo;
- 8 resistores de *330* Ω;
- 4 *jumpers* macho-fêmea de $200mm$;
- 18 *jumpers* macho-macho de $200mm$;
- 19 *jumpers* macho-mach de $110mm$;
- 1 *protoboard* de 3220 pontos.

Referências

1 PROTOCOLO I2C. 2019. Disponível em: <http://www.univasf.edu.br/~romulo.camara/novo/wp-content/uploads/2013/11/Barramento-e-Protocolo-I2C.pdf>. Citado 2 vezes nas páginas 4 e 5.

2 USANDO o buzzer com Arduino – Transdutor piezo elétrico. 2019. Disponível em: <https://portal.vidadesilicio.com.br/usando-o-buzzer-com-arduino-transdutor-piezo-eletrico/>. Citado na página 5.

3 O que é um LED? 2019. Disponível em: <https://www.mundodaeletrica.com.br/o-que-e-um-led/>. Citado na página 6.