UNIVERSIDADE FEDERAL DE SANTA MARIA
CURSO DE ENGENHARIA DE COMPUTAÇÃO
DISCIPLINA DE PROJETO DE SISTEMAS EMBARCADOS
PROFESSOR CARLOS HENRIQUE BARRIQUELLO

RELATÓRIO FINAL DE SISTEMAS EMBARCADOS: GAME GENIUS

DEIVIS STRIEDER
LUANA PALMA

SANTA MARIA, RS
JULHO DE 2016

1 INTRODUÇÃO

O jogo conhecido popularmente como “Genius” consiste em 4 luzes de cores diferentes que acendem em uma certa sequência, sempre uma de cada vez, e ao jogador cabe reproduzir essa sequência pressionando os botões correspondentes às cores até que cometa algum erro.

Para este projeto foram utilizados ferramentas de programação e gravação da fabricante Atmel Corporation, um *shield* de display LCD aliado à botões, *protoboard*, LEDs e *push-buttons* para a montagem e desenvolvimento do jogo e memória EEPROM para armazenamento de dados.

O projeto tem como objetivo implementar o funcionamento do game Genius com um processador Atmel utilizando linguagem C e cumprindo os requisitos dados de entrada, saída e armazenamento de dados.

2 FUNCIONAMENTO DO JOGO

Ao receber alimentação, ou ser resetado, o jogo apresenta o menu inicial no display (figura 1).

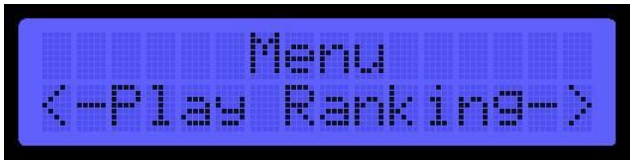

Figura 1 - Menu principal.

Caso o usuário aperte o botão nomeado no *shield* como RIGHT, o sistema mostrará o ranking do jogo (figura 2), que exibe uma posição por vez, sendo estas posições navegáveis utilizando os botões UP e DOWN. Para voltar ao menu inicial, o usuário deve apertar SELECT, que funcionará como o botão “voltar” da interface.

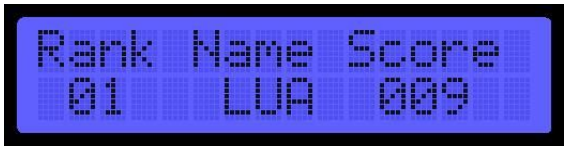

Figura 2 - Ranking.

Partindo do menu inicial, caso o usuário aperte LEFT, a tela para inserção de nome com três caracteres é exibida no display (figura 3). Para navegar pelo alfabeto em cada um dos caracteres do nome, utiliza-se os botões UP e DOWN; para confirmar a letra, deve-se pressionar o botão RIGHT.

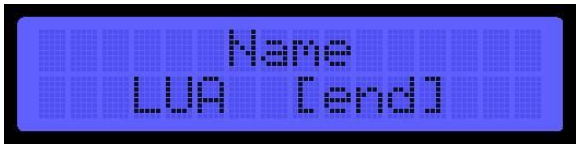

Figura 3 - Interface de inserção de nome.

Ao confirmar a terceira letra, a interface exibe uma confirmação de início de jogo (figura 4) que voltará ao menu principal caso o jogador pressione RIGHT, ou iniciará o jogo caso o botão LEFT seja pressionado.

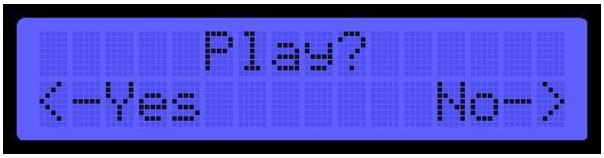

Figura 4 - Confirmação para início de jogo.

Antes que a primeira luz acenda, uma tela com contagem regressiva (figura 5) é exibida no display para preparação do usuário ao início do jogo.

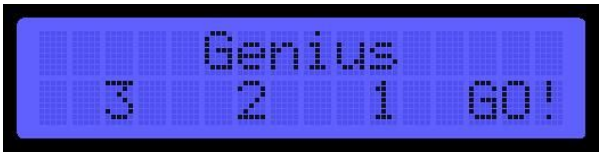

Figura 5 - Jogo em execução.

O jogo acenderá uma primeira luz aleatória e entrará em espera pelo movimento do usuário. Caso o jogador acerte o botão correspondente, uma outra luz é adicionada à sequência e essa é exibida. A cada entrada correta da sequência pelo jogador, um novo elemento aleatório é inserido à esta e o jogo prossegue da mesma maneira, até que o usuário aperte um botão fora da sequência correta. A figura 6 ilustra a interface de jogo.

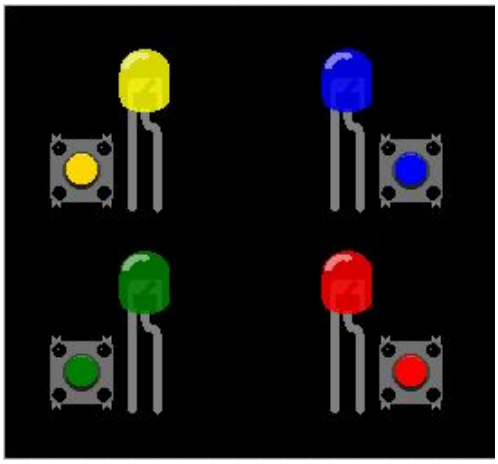

Figura 6 - Interface de jogo.

Ao final do jogo, o display exibe uma tela com o total de pontos, ou seja, a quantidade de elementos inseridos na sequência (figura 7), aguarda alguns instantes e volta ao menu inicial. O jogador entrará para o ranking caso sua pontuação seja maior ou igual ao score de menor valor no ranking.

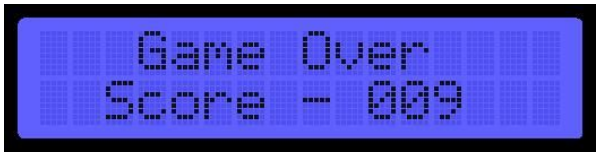

Figura 7 - Tela de final de jogo.

3 MATERIAIS UTILIZADOS

Como base para a implementação do jogo, utilizou-se uma placa Arduino MEGA 2560 que tem como core o microcontrolador ATmega 2560. A figura 8 mostra em detalhe a placa e indica os pinos ocupados pelos periféricos. O shield LCD se encaixa nos pinos apontados em laranja, mas apenas os pinos nomeados na figura são manipulados. Os botões anexos ao shield (figura 9) funcionam em uma única entrada analógica, pois o sinal que indica qual botão foi pressionado é dado por uma faixa de entrada de tensões que difere para cada um. Todos os botões referenciados são entradas do sistema e as outras portas nomeadas são configuradas como saída.

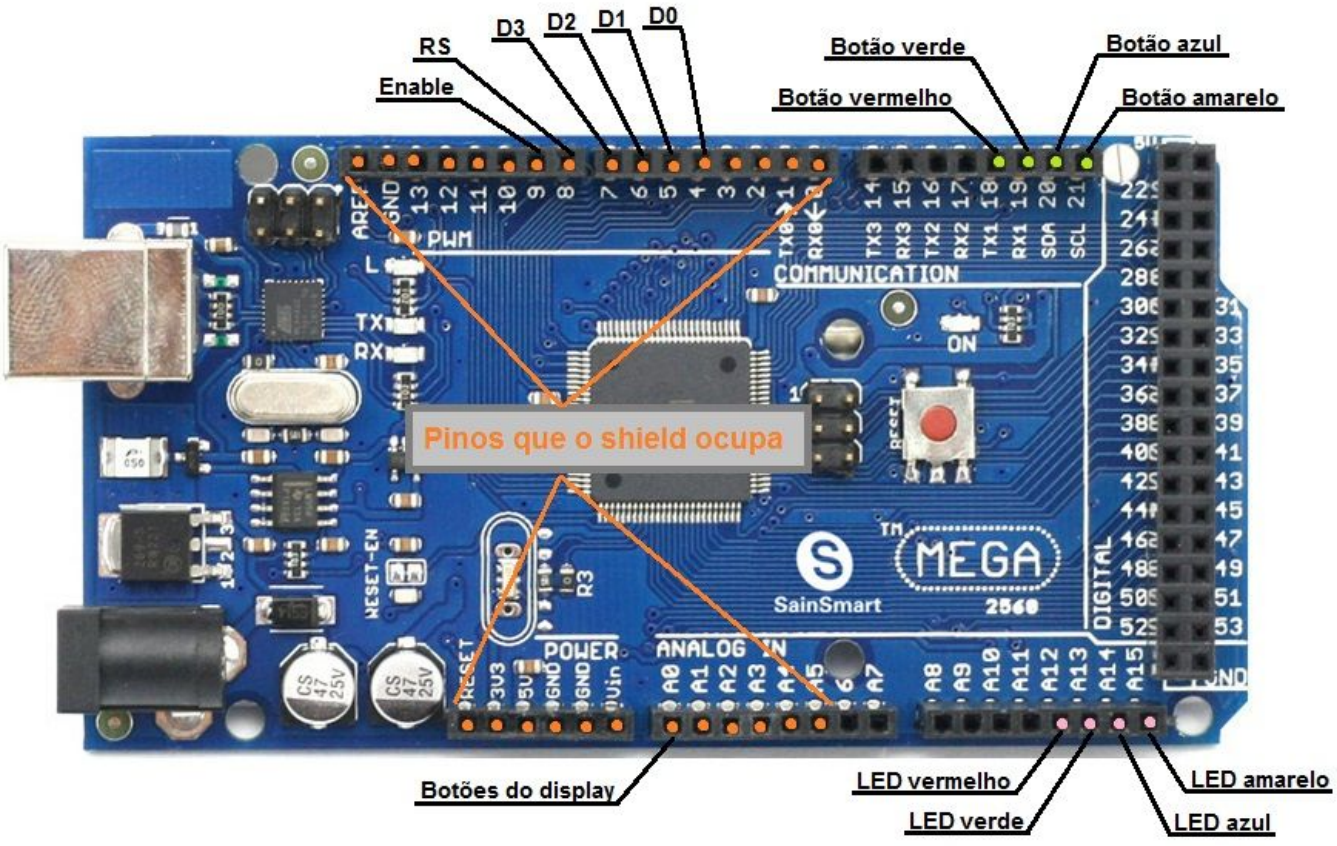

Figura 8 - Arduino MEGA com indicação das portas utilizadas.

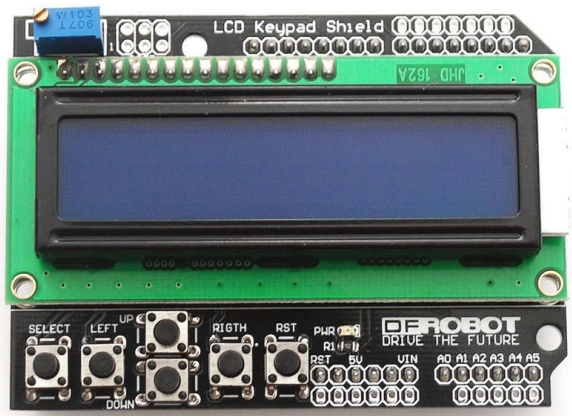

Figura 9 - LCD Keypad Shield.

Para montar a interface de jogo como mostra a figura 6, fez-se necessária a utilização de protoboard para a montagem do circuito (figura 10), onde foram posicionados quatro push-buttons com resistores de pull-up e 4 LEDs com resistores para delimitação de corrente. Os resistores de pull-up inserem obrigatoriamente a regra de que um botão pressionado equivale a 0 V na porta.

A figura 10 também indica o nome das portas onde os botões e LEDs foram conectados a nível de processador. A tabela 1 mostra as portas utilizadas na manipulação do shield.

Tabela 1 - Relação de pinos e portas entre o shield e o microcontrolador

Pino (shield)	**Porta (µc)**
Enable	PH6
RS	PH5
D0	PG5
D1	PE3
D2	PH3
D3	PH4
Botões	ADC0

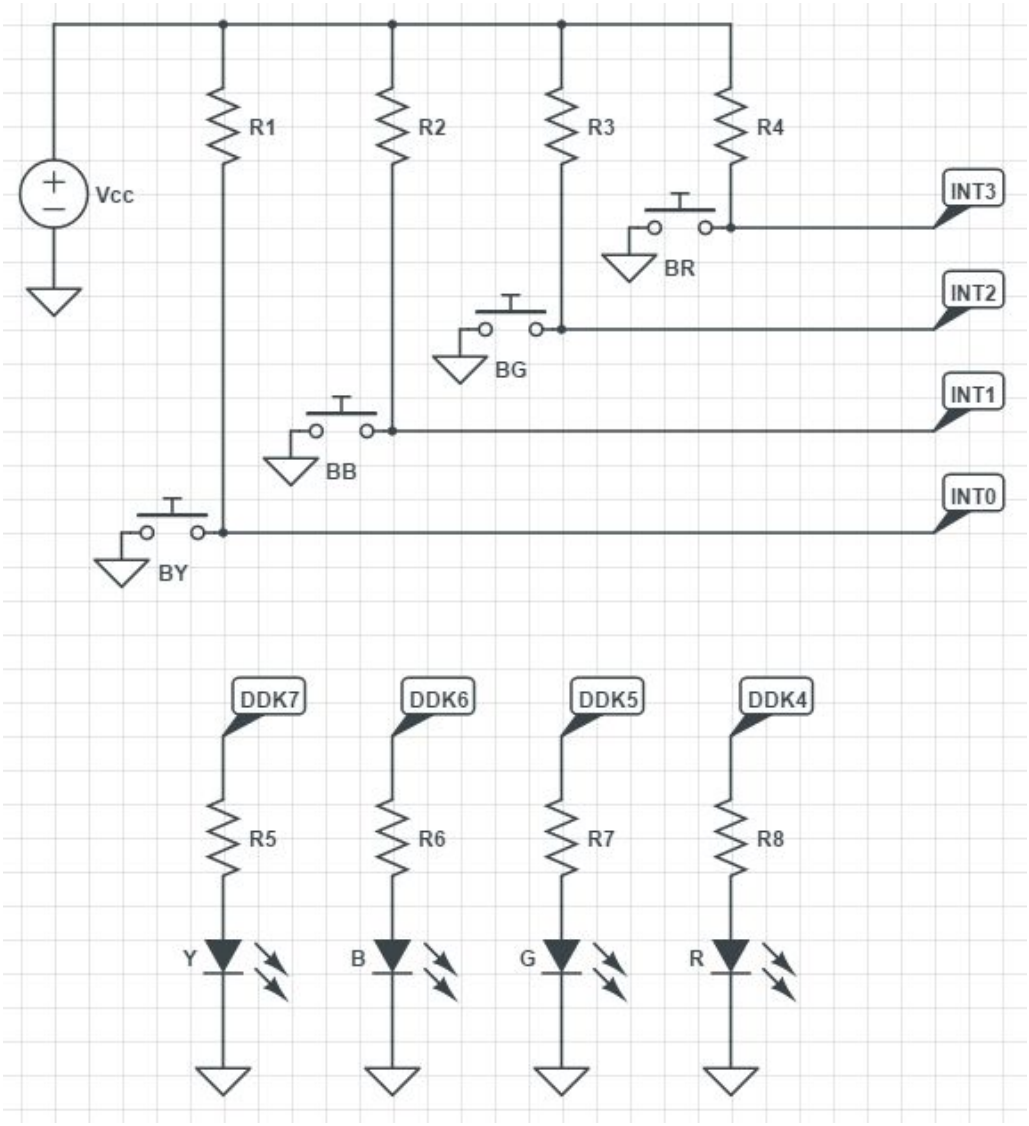

Figura 10 - Circuito da interface do jogo.

4 PROGRAMAÇÃO

A técnica de projeto utilizada para implementação do sistema foi máquina de estados finita com ponteiro de função. A FSM final apresenta seis estados: Menu Inicial; Ranking; Insere Nome; Confirma Jogo; Play Genius e Game Over. Alguns estados contam com algumas funções auxiliares no código principal e seis bibliotecas

diferentes foram construídas para manipulação facilitada de periféricos e organização do código.

4.1 Máquina de estados principal utilizando ponteiro de funções

A figura 11 mostra a máquina de estados principal do jogo, onde a mudança de estados ocorre, em sua maioria, por ativação de algum botão do shield, mas também por estouro de tempo ou sinal de fim de jogo, no caso do estado Game Over e Play Genius, respectivamente.

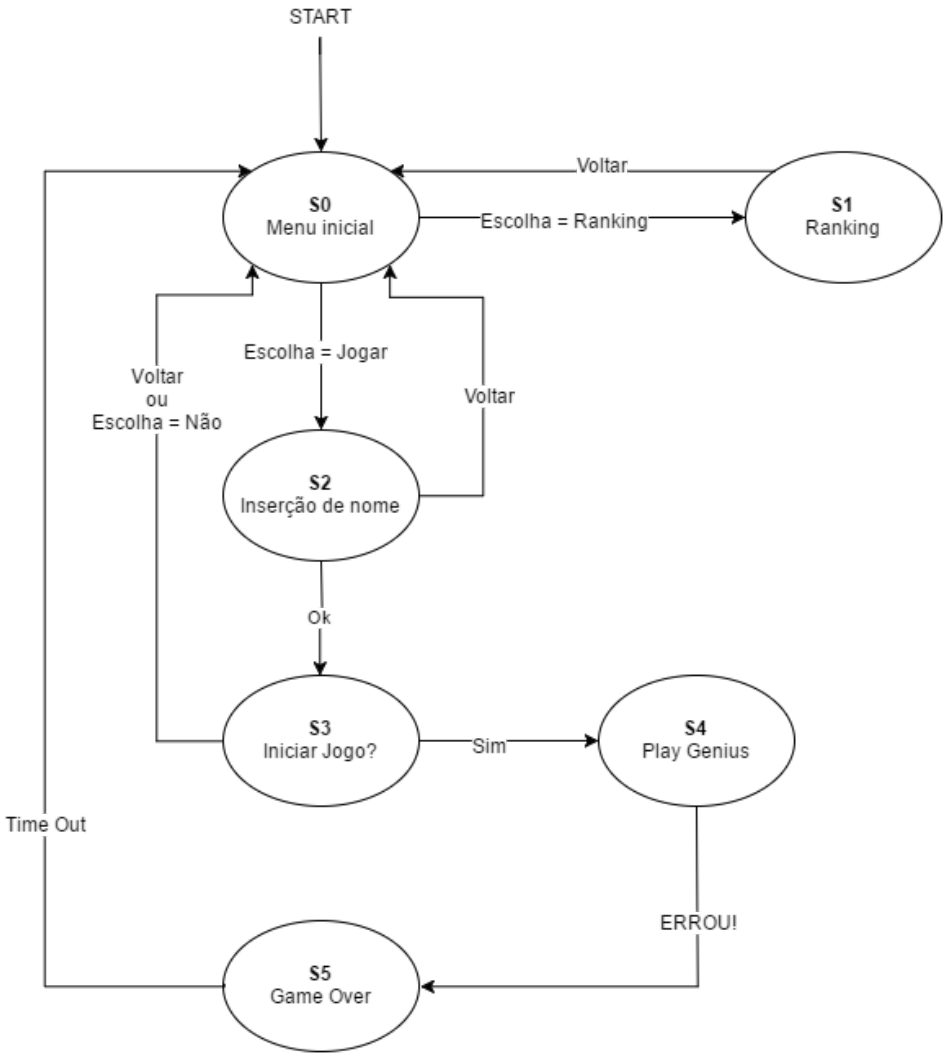

Figura 11 - Máquina de estados do Game Genius.

Para a implementação em código, um tipo “States” foi criado para poder assumir os possíveis estados declarados dentro deste tipo (figura 12). Um tipo denominado “Action” foi definido como um ponteiro para funções sem retorno e sem parâmetros (figura 13). A estrutura global “sm” (figura 14) possui uma variável que guardará o estado atual da máquina e um vetor de ponteiros de função, cada posição deste recebendo o ponteiro de uma função que implementa um estado da máquina. Estas funções foram declaradas tal como apresenta a figura 15.

```
/* Estados possíveis da FSM. */
typedef enum{
    sMenuInicial = 0, sRanking, sInsereNome, sConfirmaJogo, sPlayGenius, sGameOver
}States;
```

Figura 12 - Declaração do tipo States.

```
/* Definição do tipo do ponteiro de função. */
typedef void (*Action)(void);
```

Figura 13 - Declaração do tipo ponteiro de função

```
/* Estrutura da máquina de estados. */
struct StateMachine{
    States state;
    Action action[6];
}sm;
```

Figura 14 - Declaração da estrutura da FSM.

```
/* Protótipos das funções dos estados. */
void MenuInicial();
void Ranking();
void InsereNome();
void ConfirmaJogo();
void PlayGenius();
void GameOver();
```

Figura 15 - Declaração do protótipo das funções dos estados.

As variáveis globais utilizadas para manipular o jogo em si e o funcionamento da interface de menu são apresentadas na figura 16. A variável “name” guarda o nome do

jogador; “score” guarda a pontuação do jogo, sendo sempre zerado ao fim de cada jogo; “sequencia” guarda a sequência de luzes mostrada e incrementada durante o jogo, sendo os números correspondentes às luzes definidos como mostra a figura 17.

```
/* Variáveis globais necessárias ao funcionamento do jogo. */
uint8_t name[3];
uint8_t score = 0;
uint8_t sequencia[MAX_SEQ];
uint8_t indexSequencia = 0;
uint8_t numBotaoPressionado = 0;
bool botaoPressionado = false;
```

Figura 16 - Declaração das variáveis globais.

```
/* Definição dos números correspondentes às cores. */
#define Y_NUM 1
#define B_NUM 2
#define G_NUM 3
#define R_NUM 4
```

Figura 17 - Definição dos números correspondentes às luzes.

O restante das funções declaradas no código principal podem ser vistas na figura 18. InitStateMachine preenche o vetor “action” (figura 14) com os ponteiros para as funções apresentadas na figura 15. IniciaGenius faz a configuração inicial dos periféricos e limpa a memória EEPROM nas posições que serão utilizadas para o ranking. As funções auxiliares são utilizadas dentro das funções de estados, como segue a explicação nos próximos tópicos.

```
/* Protótipos das funções de configuração inicial. */
void InitStateMachine();
void IniciaGenius();

/* Protótipos das funções auxiliares. */
void PrintLinha(pos pPosicao, uint8_t ind);
bool InsereElemento();
void MostraSequencia();
```

Figura 18 - Outras funções do código principal.

4.1.1 Menu principal

A função do menu inicial exibe a tela da figura 1 e entra em polling lendo a entrada analógica referente aos botões do display. Ao receber um valor válido (RIGHT ou LEFT), troca o estado da máquina para a opção correspondente ao botão, seguindo a lógica do funcionamento do jogo explicado em anteriormente.

4.1.2 Ranking

A função Ranking segue o fluxo de funcionamento mostrado na figura 19, sendo o estado de espera correspondente ao polling da entrada analógica da mesma forma que ocorre no menu inicial. Há um total de 10 posições no ranking e a exibição dessas não é circular, ou seja, a primeira e a décima posição nunca poderão ser vistas consecutivamente.

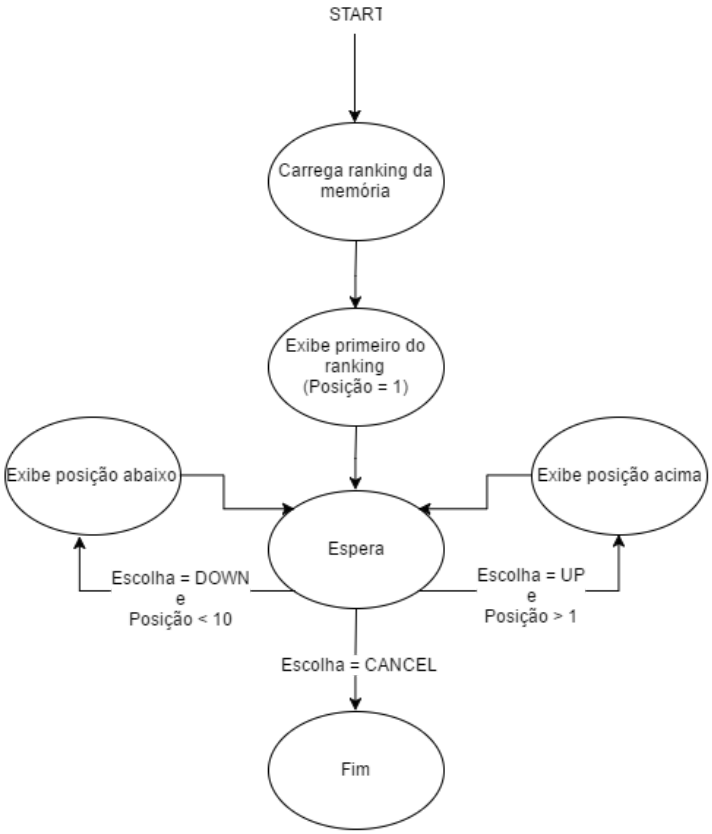

Figura 19 - Fluxograma de funcionamento da função Ranking.

4.1.3 Seleção de nome

A função de seleção de nome é apresentada na figura 20, onde, após exibir a tela da figura 3 sem os caracteres de nome (espaços no lugar de “LUA”, o nome exemplo), entra em polling esperando a estrada do jogador. O polling é quebrado cada vez que o usuário confirma uma letra com o botão RIGHT e, então, ocorre uma verificação para saber se o caractere confirmado é o último (terceiro). Em caso negativo, a função retorna ao polling.

Os três caracteres são guardados variável global “name” e são utilizados posteriormente na função Game Over para armazenar o ranking na memória.

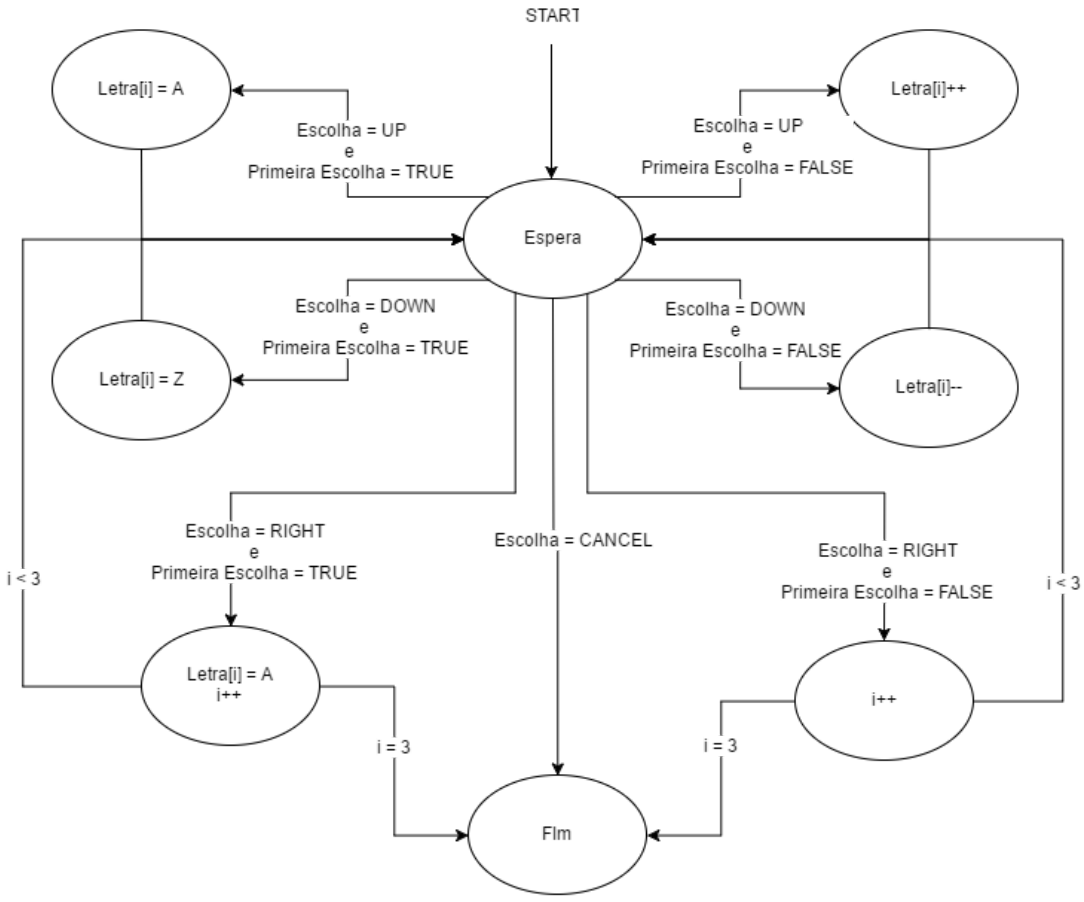

Figura 20 - Fluxograma da função de inserção de nome.

4.1.4 Confirma jogo

A função confirma jogo começa apresentando a tela da figura 4 e entra em polling tal qual a função do menu inicial. À entrada do botão LEFT, a máquina vai para o estado de jogo; para o botão RIGHT ou SELECT, a máquina regressa ao estado inicial.

4.1.5 Play Genius

O processo do jogo segue o fluxo apresentado na figura 21. Ao confirmar início de jogo no estado anterior, o display exibe a tela da figura 5 e a função InsereElemento é chamada. Há timer circular que é ativado na inicialização do sistema e seu valor é lido por esta função para gerar o número correspondente ao novo elemento luminoso da sequencia, garantindo a aleatoriedade necessária ao jogo.

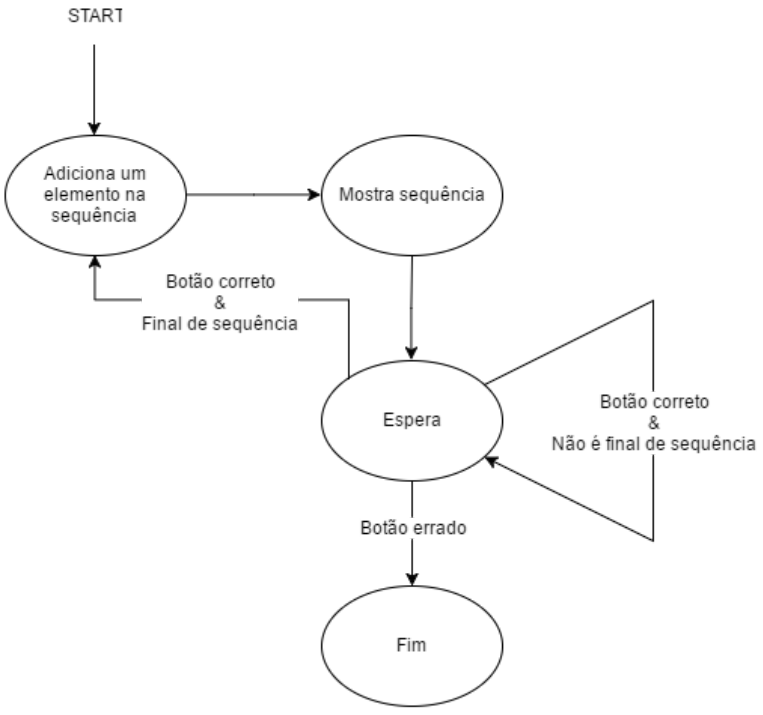

Figura 21 - Fluxograma da função Play Genius.

Após a sequencia ser exibida pela função MostraSequencia, as interrupções são habilitadas e a função entra em polling esperando a flag botaoPressionado se tornar verdadeira (esta é alterada quando um dos botões ativa a sua função de interrupção). As funções dos botões gravam seu valor específico na variável numBotaoPressionado.

A variável indexSequencia, zerada a cada rodada e incrementada a cada botão pressionado, é utilizada para navegar pela array sequencia e conferir se a entrada do jogador (ou seja, o valor de numBotaoPressionado) corresponde ao valor da sequencia naquela posição.

A variável score é incrementada cada vez que o jogador termina uma rodada com sucesso. Quando o jogador errar, este valor será utilizado pela função Game Over, o próximo estado da FSM.

4.1.6 Game Over

O fluxo de funcionamento da função Game Over pode ser analisado na figura 22. Ao entrar neste estado, além de exibir a tela com a pontuação do jogador, também lê da memória as 10 posições do ranking e compara os valores de score das posições com o score recente. Caso encontre em alguma posição uma pontuação que seja menor ou igual ao score atual, sairá do loop de comparações e deslocará uma posição para baixo todas as posições a partir desta encontrada, incluindo ela mesma.

Após o deslocamento, o score atual e o nome (variável “name”) são gravados nesta posição e todo o ranking é gravado novamente na memória.

O retorno ao estado inicial da máquina de estados é dado por estouro de tempo.

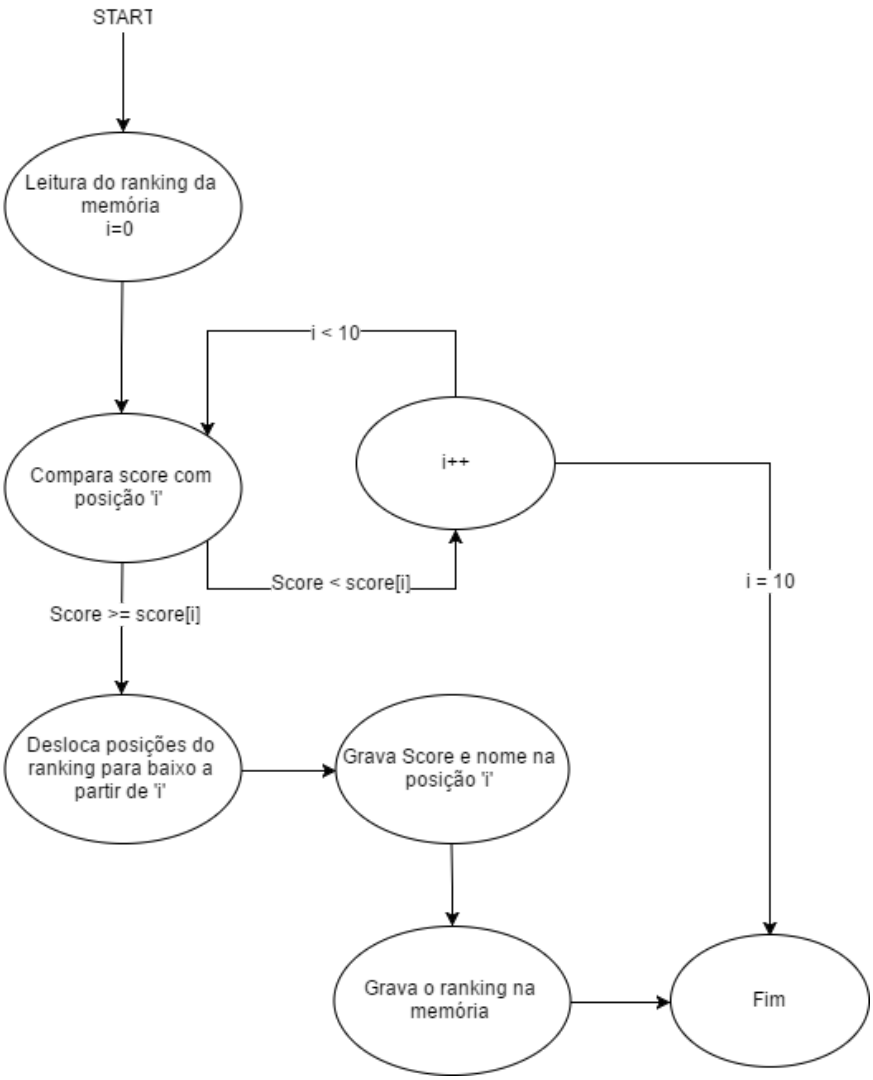

Figura 22 - Fluxograma da função Game Over.

4.2 Bibliotecas

Várias bibliotecas foram construídas para melhor organização do código e facilidade de manipulação dos periféricos. Uma das bibliotecas fundamentais e que é fornecida pela IDE da Atmel foi a “delay.h”, que permite configurar um tempo de espera de forma simples, facilitando a implementação de debounce para os botões, tanto analógicos como digitais, e também simplificando a programação dos LEDs para o jogo.

A biblioteca do display LCD foi adaptada de um exemplo online de Donald Weiman [1] disponível nas referências ao fim do relatório.

4.2.1 Biblioteca para o display LCD

Esta biblioteca possui a definição das portas utilizadas para controle e escrita no LCD; definição das instruções utilizadas neste periférico; função para escrever instruções; função para escrever uma string; função para escrever um caractere; e função de inicialização. Todas estas foram desenvolvidas para o display operando em modo 4 bits.

4.2.2 Biblioteca para entrada analógica e timer

Esta biblioteca possui as definições de valor da entrada analógica para cada botão; função de inicialização do conversor A/D; função que retorna o qual botão foi pressionado; e função de inicialização do timer.

4.2.3 Biblioteca para botões digitais

Esta biblioteca contém definições das portas utilizadas pelos botões, dos nomes das funções correspondentes a estas portas no vetor de interrupções do processador. Como funções contém somente uma para a configuração inicial dos botões e suas interrupções.

4.2.4 Biblioteca para LEDs

Esta biblioteca contém a definição das portas utilizadas pelos LEDs; função para piscar cada um dos LEDs; função para piscar todos os LEDs ao mesmo tempo; e função de inicialização das portas.

4.2.5 Biblioteca para memória EEPROM

Esta biblioteca define o endereço inicial da EEPROM onde o ranking será gravado; possui uma função para escrita em uma posição da EEPROM; uma função para leitura de uma posição; função para leitura de uma linha de ranking; e função para escrita de uma linha de ranking.

O padrão de utilização da memória é ilustrado através da tabela 2. Cada posição da memória EEPROM possui 1 byte, sendo que para armazenar uma linha do ranking é preciso de 4 bytes: três para os caracteres do nome e um para o score (a pontuação máxima é 100). Sendo assim, cada linha ocupa quatro posições da memória, sendo seu endereço inicial sempre um múltiplo de 4, uma vez que o endereço inicial utilizado no sistema é o 0x0000.

Tabela 2 - Organização da memória.

	Nome (3 bytes)			Score (1 byte)
End 0	‘L’	‘U’	‘A’	9
End 4	‘X’	‘X’	‘X’	0
End 8	‘X’	‘X’	‘X’	0
End 12	‘X’	‘X’	‘X’	0
End 16	‘X’	‘X’	‘X’	0
End 20	‘X’	‘X’	‘X’	0
End 24	‘X’	‘X’	‘X’	0
End 28	‘X’	‘X’	‘X’	0
End 32	‘X’	‘X’	‘X’	0
End 36	‘X’	‘X’	‘X’	0

4.2.6 Biblioteca com funções de interface

Esta biblioteca contém seis funções auxiliares aos seis estados da FSM do sistema. Estas funções servem para simplificar a exibição das telas (figuras 1, 2, 3, 4, 5 e 7) para cada um dos estados.

5 CONCLUSÃO

O desenvolvimento deste projeto apresentou seus principais desafios na manipulação dos periféricos, principalmente do display LCD, que tem controle próprio. A biblioteca delay.h possui a necessidade de ser declarada somente após definida a frequência de operação do processador no código, algo que não se mostra óbvio para uma primeira utilização e atrasou vários aspectos do projeto.

O requisito de entrada de dados pedido para o sistema foi contemplado pela utilização de botões de menu e de jogo; a utilização da EEPROM para guardar o ranking do jogo preencheu o requisito de armazenamento de dados; já o display LCD e os LEDs servem ao requisito de saída de dados do sistema.

6 REFERÊNCIAS

[1] Donald Weiman, LCD Programming Example using 'C', disponível em <http://web.alfredstate.edu/weimandn/programming/lcd/ATmega328/LCD_code_gcc_4d.html>.

[2] Atmel Corporation, ATmega 2560 Datasheet, disponível em <http://www.atmel.com/Images/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf>.