

LEDs Memory Game

Felipe Chermont Pereira
Engenharia Eletrônica
Universidade de Brasília, FGA
Brasília, Brasil
chernox27@gmail.com

Guilherme Simões Dias
Engenharia Eletrônica
Universidade de Brasília, FGA
Brasília, Brasil
g.simoesdias@gmail.com

Abstract—Development of a memory game using a MSP430G from Texas Instruments, to help kids with TDAH to improve their abilities in memorize sequences and to focus in a single task. The code is written in C language and the project uses four LEDs, four push buttons e a buzzer to be able to read the commands and send visual and auditive stimulus to the player.

Keywords—MSP340G, TDAH, C language, memory, focus.

I. INTRODUÇÃO

[1] O Transtorno de Déficit de Atenção com Hiperatividade (TDAH) é um transtorno neurobiológico que aparece geralmente na infância e se não acompanhado e trabalhado pode atrapalhar o âmbito pessoal e profissional durante a vida adulta. [2] Segundo os dados da Organização Mundial de Saúde, cerca de 4% da população adulta tem o TDAH, equivalente a aproximadamente 2 milhões de brasileiros. Já na população jovem, o TDAH afeta 6% das crianças e destas apenas 69% concluem os estudos.

Estes números estão diretamente associados ao mau desenvolvimento das capacidades cognitivas como memória e atenção, que são de extrema importância na vida pessoal de uma pessoa, a falta dessas habilidades causa um impacto enorme na vida pessoal de um adulto. [4] Como destacado pela Organização Brasileira de Déficit de Atenção, os sintomas na vida adulta podem ocasionar prejuízos no trabalho, nas relações amorosas, problemas na condução de veículos dentre outros. Este problema pode ser atenuado se algumas regiões do cérebro forem estimuladas desde a infância, para realizar este estímulo um método que vem se provando eficaz é a aplicação de jogos da memória. Segundo o site NeuroSaber [3] o jogo da memória está entre as 10 melhores brincadeiras simples para acalmar crianças hiperativas. Sendo bastante usada pelos professores e pais de crianças que sofrem com este transtorno, por estimular habilidades como o pensamento, a memorização a identificação de cores e sons estabelecendo conceitos de igualdade e diferença, entre outros.

No entanto, para se manter a atenção de uma criança com TDAH é necessário fazer uso recursos chamativos como luzes e sons. Logo, foi decidido fazer uso no jogo da memória de LEDs e buzzer. Agora, para realizar a execução jogo foi necessário escolher um microcontrolador, dentre as opções estavam a MSP430 e a Armel AVR (Presente no Arduino). Por conta do preço, acessibilidade e familiaridade, foi feita a escolha pelo MSP, que oferece vários modos de LPM (Low Power Mode), contém uma arquitetura de 16 bits e a MSP 430 permite a mudança do clock, algo que o Arduino não permite.

II. DESENVOLVIMENTO

A. Planejamento

Para desenvolver o jogo, idealizamos como que seria o produto para depois analisar quais processos internos teriam que ser desenvolvidos para o sistema funcionar.

Partindo com a referência do jogo já existente “Genius”, fora decidido que usaríamos quatro cores diferentes, representadas por LEDs, para indicar a sequência a ser seguida sendo que cada cor também estaria associada a um respectivo som, emitido por um *buzzer*. Para inserção da resposta optamos por usar quatro *push buttons*. Adentrando na funcionalidade, o produto tem como base um gerador de números pseudoaleatórios que dará origem a sequência que deverá ser repetida fazendo uso de entradas associadas aos botões.

Posteriormente, foram levantados os requisitos que o projeto deveria atender para alcançar nossos objetivos:

- Acender os LEDs em uma sequência pseudoaleatória.
- Utilizar sons diferentes associados a cada um dos LEDs para orientar o jogador.
- Realizar uma leitura precisa dos botões para verificar a sequência inserida.
- Disponibilizar 3 dificuldades diferentes para se adequar as capacidades do jogador.
- Criar uma interface intuitiva que auxilie a criança a saber o que está ocorrendo.

B. Descrição de Hardware

Para a construção do projeto foram usados os seguintes materiais:

Material	Quantidade	Preço	Total
MSP430G2	1	R\$40,00	R\$40,00
Protoboard	1	R\$20,00	R\$20,00
LED 10mm	4	R\$0,80	R\$3,20
Botões	4	R\$0,20	R\$0,80
Buzzer 5V	1	R\$1,50	R\$1,50
Jumper macho/fêmea	10	R\$0,25	R\$2,50
Jumper macho/macho	4	R\$0,25	R\$2,50
Preço total:			70,50

Fig. 1. Bill of materials.

A montagem do circuito foi realizada de acordo com a imagem abaixo:

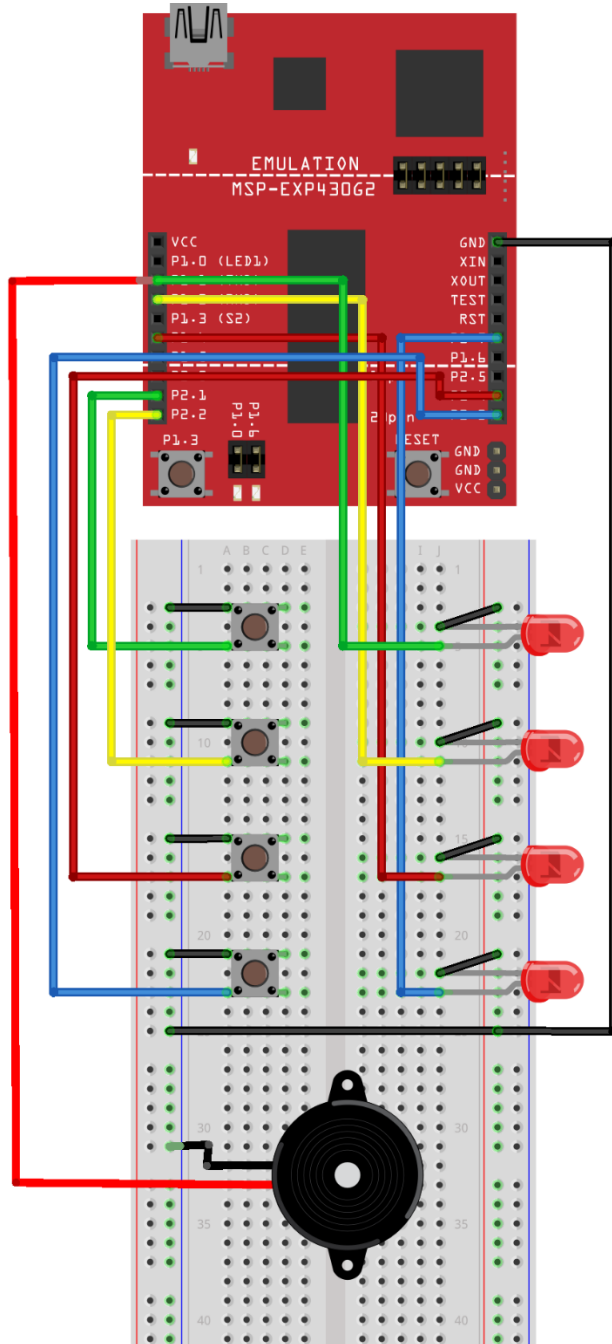


Fig. 2. Esquemático do circuito.

No total foram usadas nove portas do microcontrolador, sendo P1.1, P1.2, P1.4 e P1.5 para os LEDs, P2.1, P2.2, P2.4 e P2.5 para os botões e P1.0 para o *buzzer*. Contudo, por ser um protótipo, o projeto foi montado em uma *proto board* e foram usados jumpers para fazer as ligações. Lembrando que a saída dos pinos usados é de 3Voltz, tensão abaixo do suportado pelo buzzer mas, para os LEDs é recomendado acoplar um divisor de tensão pois estes funcionam na faixa de 1,5 a 2,5V dependendo da cor.

C. Descrição do Software

O desenvolvimento do software do jogo foi feito por partes separadas em funções para depois ser realizada a integração. Para isto, foi necessário organizar as etapas que seriam

necessárias para o funcionar do jogo como desejado. A solução adotada foi a criação de um diagrama de blocos que descrevesse todo o funcionamento do código.

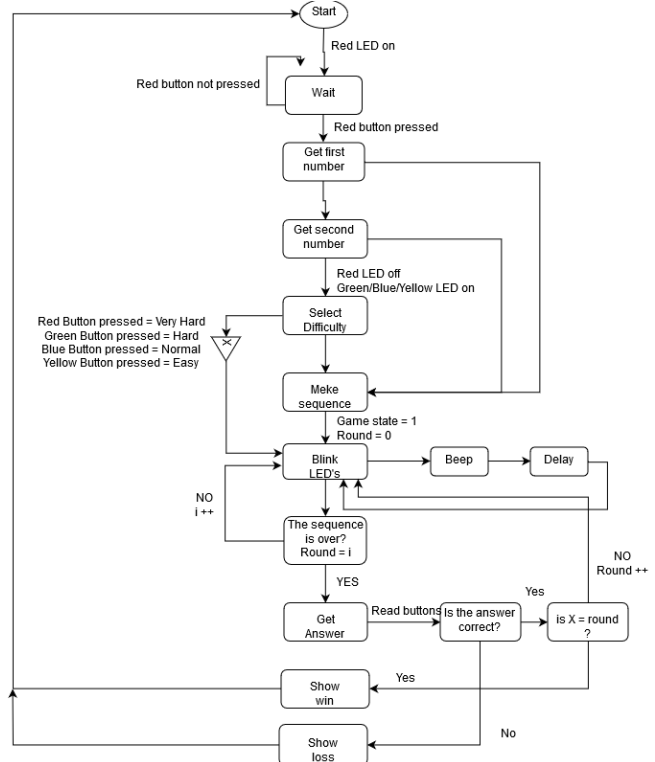


Fig. 3. Diagrama de blocos do software.

Para organizar a sequência em que cada função seria chamada e para declarar os *timers* e as saídas e entradas usadas foi criada a função principal nomeada de *main*. Partindo dela a primeira função a ser chamada foi a *Reset* na qual acende o LED vermelho e entra em um loop criado por um *while* enquanto espera que o botão associado ao LED seja pressionado, quando o evento ocorre o LED apaga e volta para a *main*.

Retornando a *main*, o próximo passo é adquirir dois números aleatórios e para isso são chamadas as funções *GetFirstNumber* e *GetSecondNumber* em que nela, é armazenado em uma variável o valor presente no *TimerA1* e no *TimerA0* no dado instante.

A próxima etapa é a escolha da dificuldade, ação realizada na função *ChooseDifficulty*. Nesta função, os LEDs verde, amarelo e azul são acesos e é criado um loop de espera pela resposta do jogador. Para sair do loop, existem 4 condições que são o ato de pressionar qualquer um dos botões, sendo que cada um deles está associado a uma dificuldade dentro do jogo. Quando um botão é pressionado, todos os LEDs se apagam e é enviado um valor para uma variável 'x' contendo o tamanho da sequência que deverá ser repetida, sendo os valores 8, 10, 12, 16 para o verde, azul, amarelo e vermelho respectivamente, para o fim do jogo e em seguida ocorre um laço que faz com que o LED associado ao botão pressionado pisque 8 vezes para evidenciar a escolha.

Dando continuidade, a próxima função *MakeSequence* é chamada e são enviados a ela os valores obtidos e armazenados do *TimerA*. Primeiramente os vetores do *GetFirstNumber* e *GetSecondNumber* são transferidos para outros dois vetores por meio de um *shift register* para depois

serem manipulados. A criação da sequência é feita por partes, preenchendo de dois em dois o vetor da sequência pseudoaleatória. O número aleatório é criado por meio da soma ponderada dos dois vetores recebidos, em que o primeiro vetor tem um multiplicador tem peso 1 e o peso do segundo multiplicador varia entre 1, 3 e 4. Dependendo do peso, os LEDs que podem ser selecionados se alteram, então quanto maior o número de divisões mais aleatório será a sequência gerada.

Com as preparações prontas, a variável `game_state` é setada em 1 e o round é zerado dando assim início ao jogo.

Nesta etapa, é chamada a função `BlinkLeds` que recebe o valor do round e da sequência pseudoaleatória para começar. Ela funciona usando um switch case em que pega o primeiro termo de array e verifica se corresponde a 0, 1, 3 ou 4 e pisca o respectivo LED (vermelho, verde, azul, amarelo) e chama a função `beep` que gera o som determinado para aquela cor.

Todo o código faz uso de intervalos de tempo entre uma ação e outra, esse tempo é controlado chamando a função `wait`. Esta função recebe um valor dentre os determinados no início do código que correspondem até que número o contador terá que contar. A contagem é feita comparando o `TA0CTL` com o `TAIFG`, quando estes se encontram significa que se passaram 10 mili segundos e a contagem é acrescida em 1. Quando a contagem se iguala ao número desejado a função retorna, gerando um atraso entre um centésimo, um quarto, meio ou um segundo.

Já a função `beep` que é chamada quando se quer ouvir um som, esta precisa receber qual nota que deve ser tocada e a duração dela. O valor de cada nota é definido no início do código. A realização de diferentes notas em um buzzer é feita a partir da oscilação da tensão aplicada nele, quanto mais rápidas as oscilações, mais agudo será o som e quanto mais lentas mais grave. Este controle é feito pelo cálculo do atraso e do tempo. Em que:

$$\text{Delay} = 1000/\text{nota}$$

$$\text{Tempo} = (\text{duração} * 100)/(\text{delay}/2)$$

Tendo estes valores calculados, a tensão no buzzer é alternada a cada delay durante o tempo definido.

Continuando o jogo, a próxima etapa é a verificação da resposta do jogador, que é feita na função `GetAnswer`. Seu funcionamento se baseia em verificar as entradas de acordo com o número do round, logo para isso é feito um laço `while` que enquanto o número de botões apertados for menor que o round e o jogador não tiver errado ele permanecerá verificando as entradas. Ao se pressionar um botão, o respectivo LED acende e é chamada a função `beep` para emitir o som, logo após é realizada a checagem de que o botão apertado é o correto. Isto é feito comparando o valor associado a cada botão com o valor no vetor da sequência pseudoaleatória em uma certa posição. A posição do valor do vetor que será usada da comparação inicialmente é zero e vai sendo incrementada conforme o jogador for inserindo respostas corretas. Ao acertar a sequência, o round é incrementado em 1 e a sequência expandida é mostrada novamente nos LEDs, entretanto, se o valor do round for igual ao valor máximo da dificuldade escolhida, a variável `game_state` muda para 2. Na outra ponta, se o jogador insere a resposta errada, a flag `game_over` é setada em 1, a leitura sai do laço e retorna para `main` o `game_state` igual a 0.

Quando `game_state` igual a 2, é chamada a função `win`, que acendera os LEDs um por um em sequência 3 vezes indicando a vitória e retornando para o início do código para um novo jogo. Já quando `game_state` igual a 0, é chamada a função `loss` que irá piscar o LED vermelho 10 vezes bem rápido indicando a derrota e retornando também para o início do código. Com isso, o ciclo do jogo é finalizado.

III. RESULTADOS

Durante o desenvolvimento do sistema, separamos os objetivos em pequenas partes. Para o primeiro ponto de controle, foi criada a sequência pseudoaleatória, verificamos com o acionamento dos quatro LEDs como a sequência criada se comportava e seu funcionamento.

Logo após conseguir gerar a sequência, foi integrado os botões aos LEDs, onde conseguiríamos verificar se a sequência apertada pelos *push buttons* era a mesma sequência gerada pelos LEDs.

Com o funcionamento correto dos LEDs e dos botões, criamos os níveis de dificuldade, onde o jogador antes de iniciar o jogo escolheria a dificuldade em que iria jogar, pelos LEDs.

A parte final do projeto consistiu na integração de um *buzzer*, que resultaria em todo o estímulo auditivo do jogo.

Todo o sistema funcionou conforme os requisitos estabelecidos. Desta forma, foi possível atender as necessidades do projeto, gerando os estímulos visuais e auditivos com perfeição.

IV. CONCLUSÃO

O Transtorno de Déficit de Atenção hoje, afeta a vida de milhares de pessoas, e apelos para ajudar essa causa são sempre bem-vindos aos olhos de quem precisa. Jogos como o jogo da memória, pode ajudar no desenvolvimento mais saudável do cérebro de uma criança com esse transtorno.

O jogo da memória descrito neste relatório, pode ajudar nessa causa, criando estímulos visuais e auditivos capazes de desenvolver habilidades como a concentração, a memória e o foco que podem ser exercidos em outras áreas da vida do indivíduo com tal transtorno.

Os objetivos definidos no trabalho foram alcançados e os resultados foram totalmente dentro do esperado.

Para trabalhos futuros, é possível aumentar o som do *buzzer* com um amplificador, para aumentar o impacto auditivo do projeto, criar uma estrutura física melhor onde podemos implementar o projeto em uma placa de circuito impressa, dentre outras melhorias possíveis.

REFERENCES

- [1] VINOCUR – Evelyn – TDAH: o que é, sintomas e tratamento – Disponível em: < <https://www.minhavidade.com.br/saude/temas/tdah>>. Acesso em: 13 set. 2019.
- [2] JOSÉ - Fernando - Aumenta o número de pessoas com TDAH e o diagnóstico adequado é o maior desafio - 16 abril 2019 - Disponível em: . Acesso em: 13 set. 2019.
- [3] 10 brincadeiras simples para desacelerar as crianças com TDAH - 05 ago. 2018 - Disponível em: . Acesso em: 13 set. 2019.
- [4] GHIGIARELLI - Denise - O TDAH NO ADULTO E O PROCESSAMENTO DAS EMOÇÕES - 27 abr. 2016. Disponível em: . Acesso em: 13 set. 2019.

[5] Alfy7 – MSP430 – Launchpad-Examples, Disponível em:
<https://github.com/alfy7/MSP430-Launchpad-Examples>

[6] AdityaWadhwa - CatchTheLED, Disponível em: <
<https://github.com/AdityaWadhwa/CatchTheLED>>

Apêndice:

main.c

```
1#include <msp430G2553.h>
2#include <msp430.h>
3
4// Definindo os LEDs e botões usados para parar o timer e pegar o número aleatório.
5#define RED_LED          BIT1          // Red LED          (P1.1)
6#define GREEN_LED        BIT2          // Green LED        (P1.2)
7#define BLUE_LED         BIT4          // Blue LED         (P1.4)
8#define YELLOW_LED       BIT7          // Yellow LED       (P1.5)
9
10#define RED_BTN          BIT1          // Red button       (P2.1)
11#define GREEN_BTN        BIT2          // Green button     (P2.2)
12#define BLUE_BTN         BIT4          // Blue button      (P2.4)
13#define YELLOW_BTN       BIT5          // Yellow button    (P2.5)
14
15#define START            BIT3          // (P1.3)
16#define SOUND            BIT0          // (P1.0)
17
18// Tamanhos de pausas diferentes para a função wait()
19#define TEN_MS           1
20#define CENTI_SEC        12
21#define QUART_SEC        30
22#define HALF_SEC         60
23#define ONE_SEC          120
24#define BLINK             80
25#define PAUSE             30
26
27// Timers
28#define ENABLE_PINS      0xFFFFE
29#define ACLK              0x0100
30#define SMCLK             0x0200
31#define CONTINUOUS        0x0020
32#define UP                0x0010
33
34// Numero de rounds para acabar o jogo de acordo com a dificuldade
35#define EASY               8
36#define NORMAL            10
37#define HARD              12
38#define EXTREME           16
39
40//Definition of the notes' frequcies in Hertz.
41#define c 261
42#define d 294
43#define e 329
44#define f 349
45#define g 391
46#define gS 415
47#define a 440
48#define aS 455
49#define b 466
50#define cH 523
51#define cSH 554
52#define dH 587
53#define dSH 622
54#define eH 659
55#define fH 698
56#define fSH 740
57#define gH 784
58#define gSH 830
59#define aH 880
60
61// Funções para o funcionamento do game
62void Reset(void);
```

```

63 int ChooseDifficulty(void);
64 void Wait(int t);
65
66 int GetFirstNumber(void);
67 int GetSecondNumber(void);
68 void MakeSequence(int sequence[16], int first_number, int second_number);
69
70 void BlinkLeds(int sequence[16], int round);
71 int GetAnswer(int sequence[16], int round);
72 void CorrectAnswer(void);
73
74 void Win(void);
75 void Loss(void);
76
77 void delay_ms(unsigned int ms )
78 {
79     unsigned int i;
80     for (i = 0; i<= ms; i++)
81         __delay_cycles(500); //Built-in function that suspends the execution for 500 cycles
82 }
83
84 void delay_us(unsigned int us )
85 {
86     unsigned int i;
87     for (i = 0; i<= us/2; i++)
88         __delay_cycles(1);
89 }
90
91 //This function generates the square wave that makes the piezo speaker sound at a
   determinated frequency.
92 void beep(unsigned int note, unsigned int duration)
93 {
94     int i;
95     long delay = (long)(10000/note); //This is the semiperiod of each note.
96     long time = (long)((duration*100)/(delay*2)); //This is how much time we need to spend
   on the note.
97     for (i=0;i<time;i++)
98     {
99         P1OUT |= BIT0;    //Set P1.2...
100         delay_us(delay);  //...for a semiperiod...
101         P1OUT &= ~BIT0;   //...then reset it...
102         delay_us(delay);  //...for the other semiperiod.
103     }
104     delay_ms(20); //Add a little delay to separate the single notes
105 }
106
107 void main(void)
108 {
109     // desativa o watch dog timer.
110     WDTCTL = WDTPW | WDTHOLD;
111
112     // inicia o timerA0 com ACLK, contando até 10ms.
113     TA0CTL |= ACLK | UP;
114     TA0CCR0 = 400;
115
116     // Inicia o Timer A1 com SMCLK.
117     TA0CTL |= ACLK | CONTINUOUS; // aqui
118     TA1CTL |= SMCLK | CONTINUOUS;
119
120     // habilita os leds como saída e os botões como entrada.
121     P1DIR |= BLUE_LED | YELLOW_LED | RED_LED | GREEN_LED;
122     P1DIR |= SOUND;

```

main.c

```
123 P1OUT &= (~SOUND);
124 P1OUT &= (~RED_LED);
125 P1OUT &= (~BLUE_LED);
126 P1OUT &= (~GREEN_LED);
127 P1OUT &= (~YELLOW_LED);
128 P2OUT |= YELLOW_BTN;
129 P2REN |= YELLOW_BTN;
130 P2OUT |= RED_BTN;
131 P2REN |= RED_BTN;
132 P2OUT |= GREEN_BTN;
133 P2REN |= GREEN_BTN;
134 P2OUT |= BLUE_BTN;
135 P2REN |= BLUE_BTN;
136 P1IES |= START; // set for 1->0 transition
137 P1IFG = 0x00;
138 P1IE |= START; // enable interrupt
139
140 while(1)
141 {
142     // espera o jogador iniciar o jogo para então gerar o numero randomico.
143     int first_number = 0;
144     Reset();
145     first_number = GetFirstNumber();
146     Wait(QUART_SEC);
147     // espera o jogador selecionar a dificuldade.
148     int difficulty;
149     int second_number = 0;
150     difficulty = ChooseDifficulty();
151     second_number = GetSecondNumber();
152     // Preenche o array com uma combinação de dois números aleatórios.
153     int sequence[16] = {0.0};
154     MakeSequence(sequence, first_number, second_number);
155     int game_state = 1;
156     int round = 0;
157     while(game_state == 1)
158     {
159         Wait(ONE_SEC);
160         BlinkLeds(sequence, round);
161
162         // Espera o jogador prescionar o botão e verifica se é o correto.
163         Wait(TEN_MS);
164         game_state = GetAnswer(sequence, round);
165         Wait(TEN_MS);
166
167         // Se a resposta for correta, pisca o verde e segue para a proxima.
168         if (game_state == 1)
169         {
170             CorrectAnswer();
171             round++;
172         }
173         Wait(TEN_MS);
174
175         // Ao se terminar o maximo de rounds permitidos pela dificuldade, os leds
176         piscaram indicando vitoria.
177         if (round == difficulty)
178         {
179             game_state = 2;
180         }
181         Wait(TEN_MS);
182     }
183     if (game_state == 2)
184     {
```

main.c

```

184         Win();
185     }
186     // If not, then the loop quit because game_state == 0; show a loss
187     else
188     {
189         Loss();
190     }
191 }
192 }
193
194 void Reset(void)
195 {
196     P1OUT |= RED_LED;
197     int x = 0;
198     while (x < 1)
199     {
200         if ((P2IN & RED_BTN) == 0)
201         {
202             P1OUT &= (~RED_LED);
203             x = 1;
204             Wait(HALF_SEC);
205             _enable_interrupts();
206         }
207     }
208 }
209
210 /*****
211 /   Mostra para o jogador 3 cores de LEDs (Verdem, azul e amarelo); Eles podem prescionar
212 /   o botão correspondente para selecionar a dificuldade, que decide o número de
213 /   rounds que devem ser cumpridos para ganhar o jogo, assim como a velocidade de
214 /   amostragem.
215 *****/
216 int ChooseDifficulty(void)
217 {
218     P1OUT |= (BLUE_LED | YELLOW_LED | GREEN_LED);
219     int x = 0;
220     int i;
221     while (x < 1)
222     {
223         if ((P2IN & GREEN_BTN) == 0) //
224         {
225             P1OUT &= (~GREEN_LED);
226             P1OUT &= (~(BLUE_LED | YELLOW_LED));
227             x = EASY;
228             for (i = 0; i < 8; i++)
229             {
230                 P1OUT = (P1OUT ^ GREEN_LED);
231                 Wait(CENTI_SEC);
232             }
233             if ((P2IN & BLUE_BTN) == 0) //
234             {
235                 P1OUT &= (~GREEN_LED);
236                 P1OUT &= (~(BLUE_LED | YELLOW_LED));
237                 x = NORMAL;
238                 for (i = 0; i < 8; i++)
239                 {
240                     P1OUT = (P1OUT ^ BLUE_LED);
241                     Wait(CENTI_SEC);
242                 }

```



```

243     }
244     if ((P2IN & YELLOW_BTN) == 0) //
-----
245     {
246         P1OUT &= (~GREEN_LED);
247         P1OUT &= (~(BLUE_LED | YELLOW_LED));
248         x = HARD;
249         for (i = 0; i < 8; i++)
250         {
251             P1OUT = (P1OUT ^ YELLOW_LED);
252             Wait(CENTI_SEC);
253         }
254     }
255     if ((P2IN & RED_BTN) == 0) //
-----
256     {
257         P1OUT &= (~GREEN_LED);
258         P1OUT &= (~(BLUE_LED | YELLOW_LED));
259         x = EXTREME;
260         for (i = 0; i < 8; i++)
261         {
262             P1OUT = (P1OUT ^ RED_LED);
263             Wait(CENTI_SEC);
264         }
265     }
266 }
267 return x;
268 }
269
270 void Wait(int t)
271 {
272     int i = 0;
273     // enquanto a contagem não atingir o limite:
274     while (i <= t)
275     { // quando outros 10 ms tiverem passado.
276         if (TA0CTL & TAIFG)
277         {
278             // aumenta o contador e conta novamente 10 ms.
279             i++;
280             TA0CTL &= (~TAIFG);
281         }
282     }
283 }
284
285 int GetFirstNumber(void)
286 {
287     int first_num = 0;
288     first_num = TA0R;
289     return first_num;
290 }
291
292 int GetSecondNumber(void)
293 {
294     int second_num = 0;
295     second_num = TA1R;
296     return second_num;
297 }
298
299 // cria uma sequencia semi aleatória usando os dois arrays criados anteriormente.
300 void MakeSequence(int sequence[8], int first_number, int second_number)
301 {
302     int i;

```

main.c

```
303 int first_array[16] = {0.0};
304 int second_array[16] = {0.0};
305 for (i = 0; i < 16; i++)
306 {
307     first_array[(15 - i)] = ((first_number >> i) & 0x01);
308     second_array[(15 - i)] = ((second_number >> i) & 0x01);
309 }
310 for (i = 0; i < 2; i++)
311 {
312     sequence[i] = (first_array[i]) + (second_array[i] * 1);
313 }
314 for (i = 2; i < 3; i++)
315 {
316     sequence[i] = (first_array[i]) + (second_array[i] * 3);
317 }
318 for (i = 3; i < 4; i++)
319 {
320     sequence[i] = (first_array[i]) + (second_array[i] * 4);
321 }
322 for (i = 4; i < 5; i++)
323 {
324     sequence[i] = (first_array[i]) + (second_array[i] * 1);
325 }
326 for (i = 5; i < 6; i++)
327 {
328     sequence[i] = (first_array[i]) + (second_array[i] * 3);
329 }
330 for (i = 6; i < 7; i++)
331 {
332     sequence[i] = (first_array[i]) + (second_array[i] * 4);
333 }
334 for (i = 7; i < 8; i++)
335 {
336     sequence[i] = (first_array[i]) + (second_array[i] * 4);
337 }
338 for (i = 8; i < 9; i++)
339 {
340     sequence[i] = (first_array[i]) + (second_array[i] * 1);
341 }
342 for (i = 9; i < 10; i++)
343 {
344     sequence[i] = (first_array[i]) + (second_array[i] * 3);
345 }
346 for (i = 10; i < 11; i++)
347 {
348     sequence[i] = (first_array[i]) + (second_array[i] * 4);
349 }
350 for (i = 11; i < 12; i++)
351 {
352     sequence[i] = (first_array[i]) + (second_array[i] * 3);
353 }
354 for (i = 12; i < 13; i++)
355 {
356     sequence[i] = (first_array[i]) + (second_array[i] * 1);
357 }
358 for (i = 13; i < 14; i++)
359 {
360     sequence[i] = (first_array[i]) + (second_array[i] * 3);
361 }
362 for (i = 14; i < 15; i++)
363 {
364     sequence[i] = (first_array[i]) + (second_array[i] * 1);
```

```

365     }
366     for (i = 15; i < 16; i++)
367     {
368         sequence[i] = (first_array[i]) + (second_array[i] * 4);
369     }
370 }
371
372 void BlinkLeds(int sequence[16], int round)
373 {
374     int i = 0;
375     do
376     {
377         switch (sequence[i])
378         {
379             case (0):    P1OUT |= RED_LED;
380                         Wait(BLINK);
381                         P1OUT &= (~RED_LED);
382                         beep(a, 500);
383                         Wait(PAUSE);
384                         break;
385
386             case (1):    P1OUT |= GREEN_LED;
387                         Wait(BLINK);
388                         P1OUT &= (~GREEN_LED);
389                         beep(b, 125);
390                         Wait(PAUSE);
391                         break;
392
393             case (3):    P1OUT |= BLUE_LED;
394                         Wait(BLINK);
395                         P1OUT &= (~BLUE_LED);
396                         beep(aH, 880);
397                         Wait(PAUSE);
398                         break;
399
400             case (4):    P1OUT |= YELLOW_LED;
401                         Wait(BLINK);
402                         P1OUT &= (~YELLOW_LED);
403                         beep(cH, 150);
404                         Wait(PAUSE);
405                         break;
406         }
407
408         i = i + 1;
409     }
410     while (i <= round);
411 }
412
413
414 /*****
415 / Essa função espera o jogador aplicar uma resposta para depois julga-la.
416 /
417 / Isto é feito esperando o jogador aplicar uma resposta e depois verificando se
418 / se é a entrada certa para o elemento i da sequencia do array, em que i é o
419 / n-nesimo LED mostrado.
420 *****/
421 int GetAnswer(int sequence[16], int round)
422 {
423     int i = 0;
424     int game_over = 0;
425     // Permanece no loop até que uma resposta errada seja emitida.
426     while (i <= round && game_over == 0)

```

```

427 {
428 //    Wait(HALF_SEC);
429     if ((P2IN & RED_BTN) == 0)
430     {
431         P1OUT |= RED_LED;
432         beep(aH, 880);
433         if (sequence[i] == 0)
434             i++;
435         else
436             game_over = 1;
437         Wait(QUART_SEC);
438         P1OUT &= (~RED_LED);
439     }
440     if ((P2IN & GREEN_BTN) == 0)
441     {
442         P1OUT |= GREEN_LED;
443         beep(aH, 880);
444         if (sequence[i] == 1)
445             i++;
446         else
447             game_over = 1;
448         Wait(QUART_SEC);
449         P1OUT &= (~GREEN_LED);
450     }
451     if ((P2IN & BLUE_BTN) == 0)
452     {
453         P1OUT |= BLUE_LED;
454         beep(aH, 880);
455         if (sequence[i] == 3)
456             i++;
457         else
458             game_over = 1;
459         Wait(QUART_SEC);
460         P1OUT &= (~BLUE_LED);
461     }
462     if ((P2IN & YELLOW_BTN) == 0)
463     {
464         P1OUT |= YELLOW_LED;
465         beep(aH, 880);
466         if (sequence[i] == 4)
467             i++;
468         else
469             game_over = 1;
470         Wait(QUART_SEC);
471         P1OUT &= (~YELLOW_LED);
472     }
473 }
474 // Se a resposta for errada, envia um sinal para a função main() para mostrar
475 // Loss().
476 if (game_over == 0)
477     return 1;
478 // Caso contrário, envia um sinal para a função main() para indicar que a
479 // resposta foi correta e continuar.
480 else
481     return 0;
482 }
483
484
485 /*****
486 /    Essa função pisca o LED verde 8 vezes bem rapido indicando que a sequencia
487 /    inserida é correta.
488 *****/

```

```

489 void CorrectAnswer(void)
490 {
491     int i;
492     for (i = 0; i < 8; i++)
493     {
494         P2OUT = (P2OUT ^ GREEN_LED);
495         Wait(CENTI_SEC);
496     }
497 }
498
499 /*****
500 / Essa função faz com que os LEDs pisquem rapido indicando a vitória
501 *****/
502 void Win(void)
503 {
504     int i;
505     for (i = 0; i < 3; i++)
506     {
507         P1OUT |= RED_LED;
508         Wait(CENTI_SEC);
509         P1OUT &= (~RED_LED);
510         Wait(CENTI_SEC);
511         P1OUT |= GREEN_LED;
512         Wait(CENTI_SEC);
513         P1OUT &= (~GREEN_LED);
514         Wait(CENTI_SEC);
515         P1OUT |= BLUE_LED;
516         Wait(CENTI_SEC);
517         P1OUT &= (~BLUE_LED);
518         Wait(CENTI_SEC);
519         P1OUT |= YELLOW_LED;
520         Wait(CENTI_SEC);
521         P1OUT &= (~YELLOW_LED);
522         Wait(CENTI_SEC);
523     }
524 }
525
526 /*****
527 / Essa função faz com que o LED vermelho pisque 3 vezes devagar indicando a derrota
528 *****/
529 void Loss(void)
530 {
531     int i;
532     for (i = 0; i < 5; i++)
533     {
534         P1OUT |= RED_LED;
535         Wait(CENTI_SEC);
536         P1OUT &= (~RED_LED);
537         Wait(CENTI_SEC);
538         P1OUT &= (~RED_LED);
539         Wait(CENTI_SEC);
540         P1OUT &= (~RED_LED);
541         Wait(CENTI_SEC);
542     }
543     Wait(ONE_SEC);
544 }
545
546 interrupt void button_handler(void) {
547
548     // check which interrupt occurred
549     if (P1IFG & START) {
550         P1IFG = 0x00;; // reset the interrupt flag

```

main.c

```
551     }  
552 }  
553  
554 // declare interrupt handlers  
555 //-----  
556 ISR_VECTOR(button_handler, ".int02")           // declare P1 interrupt handler  
557
```