

main.c

```
1#include <msp430G2553.h>
2#include <msp430.h>
3
4// p1.3 , p2.0, p2.3, p1.0 -- p1.5
5
6
7// Definindo os LEDs e botões usados para parar o timer e pegar o número aleatório.
8#define RED_LED BIT1 // Red LED (P1.1)
9#define GREEN_LED BIT2 // Green LED (P1.2)
10#define BLUE_LED BIT4 // Blue LED (P1.4)
11#define YELLOW_LED BIT7 // Yellow LED (P1.5)
12
13#define RED_BTN BIT1 // Red button (P2.1)
14#define GREEN_BTN BIT2 // Green button (P2.2)
15#define BLUE_BTN BIT4 // Blue button (P2.4)
16#define YELLOW_BTN BIT5 // Yellow button (P2.5)
17#define RED_SND BIT0 // (P1.0)
18#define GREEN_SND BIT3 // (P1.3)
19#define BLUE_SND BIT0 // (P2.0)
20#define YELLOW_SND BIT3 // (P2.3)
21
22// Tamanhos de pausas diferentes para a função wait()
23#define TEN_MS 1
24#define CENTI_SEC 12
25#define QUART_SEC 30
26#define HALF_SEC 60
27#define ONE_SEC 120
28#define BLINK 80
29#define PAUSE 30
30
31// Timers
32#define ENABLE_PINS 0xFFFFE
33#define ACLK 0x0100
34#define SMCLK 0x0200
35#define CONTINUOUS 0x0020
36#define UP 0x0010
37
38// Numero de rounds para acabar o jogo de acordo com a dificuldade
39#define EASY 8
40#define NORMAL 10
41#define HARD 12
42#define EXTREME 16
43
44//Definition of the notes' frequencies in Hertz.
45#define c 261
46#define d 294
47#define e 329
48#define f 349
49#define g 391
50#define gS 415
51#define a 440
52#define aS 455
53#define b 466
54#define cH 523
55#define cSH 554
56#define dH 587
57#define dSH 622
58#define eH 659
59#define fH 698
60#define fSH 740
61#define gH 784
62#define gSH 830
```

```

63 #define aH 880
64
65 // Funções para o funcionamento do game
66 void Reset(void);
67 int ChooseDifficulty(void);
68 void Wait(int t);
69
70 int GetFirstNumber(void);
71 int GetSecondNumber(void);
72 void MakeSequence(int sequence[16], int first_number, int second_number);
73
74 void BlinkLeds(int sequence[16], int round);
75 int GetAnswer(int sequence[16], int round);
76 void CorrectAnswer(void);
77
78 void Win(void);
79 void Loss(void);
80
81 void Tic();
82 void Toc();
83 void Tuc();
84 void Tac();
85
86 void delay_ms(unsigned int ms )
87 {
88     unsigned int i;
89     for (i = 0; i <= ms; i++)
90         __delay_cycles(500); //Built-in function that suspends the execution for 500 cycles
91 }
92
93 void delay_us(unsigned int us )
94 {
95     unsigned int i;
96     for (i = 0; i <= us/2; i++)
97         __delay_cycles(1);
98 }
99
100 //This function generates the square wave that makes the piezo speaker sound at a
    determinated frequency.
101 void beep(unsigned int note, unsigned int duration)
102 {
103     int i;
104     long delay = (long)(10000/note); //This is the semiperiod of each note.
105     long time = (long)((duration*100)/(delay*2)); //This is how much time we need to spend
    on the note.
106     for (i=0; i<time; i++)
107     {
108         P1OUT |= BIT0; //Set P1.2...
109         delay_us(delay); //...for a semiperiod...
110         P1OUT &= ~BIT0; //...then reset it...
111         delay_us(delay); //...for the other semiperiod.
112     }
113     delay_ms(20); //Add a little delay to separate the single notes
114 }
115
116 void main(void)
117 {
118     // desativa o watch dog timer.
119     WDTCTL = WDTPW | WDTHOLD;
120
121     // inicia o timerA0 com ACLK, contando até 10ms.
122     TA0CTL |= ACLK | UP;

```

main.c

```
123 TA0CCR0 = 400;
124
125 // Inicia o Timer A1 com SMCLK.
126 TA0CTL |= ACLK | CONTINUOUS; // aqui
127 TA1CTL |= SMCLK | CONTINUOUS;
128
129 // habilita os leds como saída e os botões como entrada.
130 P1DIR |= BLUE_LED | YELLOW_LED | RED_LED | GREEN_LED;
131 P1DIR |= RED_SND | GREEN_SND;
132 P1DIR |= BLUE_SND | YELLOW_SND;
133 P1OUT &= (~RED_SND);
134 P1OUT &= (~BLUE_SND);
135 P1OUT &= (~GREEN_SND);
136 P1OUT &= (~YELLOW_SND);
137 P1OUT &= (~RED_LED);
138 P1OUT &= (~BLUE_LED);
139 P1OUT &= (~GREEN_LED);
140 P1OUT &= (~YELLOW_LED);
141 P2OUT |= YELLOW_BTN;
142 P2REN |= YELLOW_BTN;
143 P2OUT |= RED_BTN;
144 P2REN |= RED_BTN;
145 P2OUT |= GREEN_BTN;
146 P2REN |= GREEN_BTN;
147 P2OUT |= BLUE_BTN;
148 P2REN |= BLUE_BTN;
149
150 while(1)
151 {
152     // espera o jogador iniciar o jogo para então gerar o numero randomico.
153     int first_number = 0;
154     Reset();
155     first_number = GetFirstNumber();
156     Wait(QUART_SEC);
157     // espera o jogador selecionar a dificuldade.
158     int difficulty;
159     int second_number = 0;
160     difficulty = ChooseDifficulty();
161     second_number = GetSecondNumber();
162     // Preenche o array com uma combinação de dois números aleatórios.
163     int sequence[16] = {0.0};
164     MakeSequence(sequence, first_number, second_number);
165     int game_state = 1;
166     int round = 0;
167     while(game_state == 1)
168     {
169         Wait(ONE_SEC);
170         BlinkLeds(sequence, round);
171
172         // Espera o jogador prescionar o botão e verifica se é o correto.
173         Wait(TEN_MS);
174         game_state = GetAnswer(sequence, round);
175         Wait(TEN_MS);
176
177         // Se a resposta for correta, pisca o verde e segue para a proxima.
178         if (game_state == 1)
179         {
180             CorrectAnswer();
181             round++;
182         }
183         Wait(TEN_MS);
184     }
```

main.c

```

185         // Ao se terminar o maximo de rounds permitidos pela dificuldade, os leds
           piscaram indicando vitoria.
186         if (round == difficulty)
187         {
188             game_state = 2;
189         }
190         Wait(TEN_MS);
191     }
192     if (game_state == 2)
193     {
194         Win();
195     }
196     // If not, then the loop quit because game_state == 0; show a loss
197     else
198     {
199         Loss();
200     }
201 }
202 }
203
204 void Reset(void)
205 {
206     P1OUT |= RED_LED;
207     int x = 0;
208     while (x < 1)
209     {
210         if ((P2IN & RED_BTN) == 0)
211         {
212             P1OUT &= (~RED_LED);
213             x = 1;
214         }
215     }
216 }
217
218 /*****
219 /   Mostra para o jogador 3 cores de LEDs (Verdem, azul e amarelo); Eles podem prescionar
220 /   o botão correspondente para selecionar a dificuldade, que decide o número de
221 /   rounds que devem ser cumpridos para ganhar o jogo, assim como a velocidade de
           amostragem.
222 *****/
223 int ChooseDifficulty(void)
224 {
225     P1OUT |= (BLUE_LED | YELLOW_LED | GREEN_LED);
226     int x = 0;
227     int i;
228     while (x < 1)
229     {
230         if ((P2IN & GREEN_BTN) == 0) //
           -----
231         {
232             P1OUT &= (~GREEN_LED);
233             P1OUT &= (~(BLUE_LED | YELLOW_LED));
234             x = EASY;
235             for (i = 0; i < 8; i++)
236             {
237                 P1OUT = (P1OUT ^ GREEN_LED);
238                 Wait(CENTI_SEC);
239             }
240         }
241         if ((P2IN & BLUE_BTN) == 0) //
           -----
242         {

```

main.c

```

243         P1OUT &= (~GREEN_LED);
244         P1OUT &= (~(BLUE_LED | YELLOW_LED));
245         x = NORMAL;
246         for (i = 0; i < 8; i++)
247         {
248             P1OUT = (P1OUT ^ BLUE_LED);
249             Wait(CENTI_SEC);
250         }
251     }
252     if ((P2IN & YELLOW_BTN) == 0) //
253     {
254         P1OUT &= (~GREEN_LED);
255         P1OUT &= (~(BLUE_LED | YELLOW_LED));
256         x = HARD;
257         for (i = 0; i < 8; i++)
258         {
259             P1OUT = (P1OUT ^ YELLOW_LED);
260             Wait(CENTI_SEC);
261         }
262     }
263     if ((P2IN & RED_BTN) == 0) //
264     {
265         P1OUT &= (~GREEN_LED);
266         P1OUT &= (~(BLUE_LED | YELLOW_LED));
267         x = EXTREME;
268         for (i = 0; i < 8; i++)
269         {
270             P1OUT = (P1OUT ^ RED_LED);
271             Wait(CENTI_SEC);
272         }
273     }
274 }
275 return x;
276 }
277
278 void Wait(int t)
279 {
280     int i = 0;
281     // enquanto a contagem não atingir o limite:
282     while (i <= t)
283     { // quando outros 10 ms tiverem passado.
284         if (TA0CTL & TAIFG)
285         {
286             // aumenta o contador e conta novamente 10 ms.
287             i++;
288             TA0CTL &= (~TAIFG);
289         }
290     }
291 }
292
293 void Tic ()
294 {
295     beep(a, 500);
296 }
297 void Toc ()
298 {
299     beep(cH, 150);
300 }
301 void Tac ()
302 {

```

```

303     beep(b, 125);
304 }
305 void Tuc ()
306 {
307     beep(aH, 880);
308 }
309
310 int GetFirstNumber(void)
311 {
312     int first_num = 0;
313     first_num = TA0R;
314     return first_num;
315 }
316
317 int GetSecondNumber(void)
318 {
319     int second_num = 0;
320     second_num = TA1R;
321     return second_num;
322 }
323
324 // cira uma sequencia semi aleatória usando os dois arrays criados anteriormente.
325 void MakeSequence(int sequence[8], int first_number, int second_number)
326 {
327     int i;
328     int first_array[16] = {0.0};
329     int second_array[16] = {0.0};
330     for (i = 0; i < 16; i++)
331     {
332         first_array[(15 - i)] = ((first_number >> i) & 0x01);
333         second_array[(15 - i)] = ((second_number >> i) & 0x01);
334     }
335     for (i = 0; i < 2; i++)
336     {
337         sequence[i] = (first_array[i]) + (second_array[i] * 1);
338     }
339     for (i = 2; i < 3; i++)
340     {
341         sequence[i] = (first_array[i]) + (second_array[i] * 3);
342     }
343     for (i = 3; i < 4; i++)
344     {
345         sequence[i] = (first_array[i]) + (second_array[i] * 4);
346     }
347     for (i = 4; i < 5; i++)
348     {
349         sequence[i] = (first_array[i]) + (second_array[i] * 1);
350     }
351     for (i = 5; i < 6; i++)
352     {
353         sequence[i] = (first_array[i]) + (second_array[i] * 3);
354     }
355     for (i = 6; i < 7; i++)
356     {
357         sequence[i] = (first_array[i]) + (second_array[i] * 4);
358     }
359     for (i = 7; i < 8; i++)
360     {
361         sequence[i] = (first_array[i]) + (second_array[i] * 4);
362     }
363     for (i = 8; i < 9; i++)
364     {

```

main.c

```

365     sequence[i] = (first_array[i]) + (second_array[i] * 1);
366 }
367 for (i = 9; i < 10; i++)
368 {
369     sequence[i] = (first_array[i]) + (second_array[i] * 3);
370 }
371 for (i = 10; i < 11; i++)
372 {
373     sequence[i] = (first_array[i]) + (second_array[i] * 4);
374 }
375 for (i = 11; i < 12; i++)
376 {
377     sequence[i] = (first_array[i]) + (second_array[i] * 3);
378 }
379 for (i = 12; i < 13; i++)
380 {
381     sequence[i] = (first_array[i]) + (second_array[i] * 1);
382 }
383 for (i = 13; i < 14; i++)
384 {
385     sequence[i] = (first_array[i]) + (second_array[i] * 3);
386 }
387 for (i = 14; i < 15; i++)
388 {
389     sequence[i] = (first_array[i]) + (second_array[i] * 1);
390 }
391 for (i = 15; i < 16; i++)
392 {
393     sequence[i] = (first_array[i]) + (second_array[i] * 4);
394 }
395 }
396
397 void BlinkLeds(int sequence[16], int round)
398 {
399     int i = 0;
400     do
401     {
402         switch (sequence[i])
403         {
404             case (0):    P1OUT |= RED_LED;
405                         Wait(BLINK);
406                         P1OUT &= (~RED_LED);
407                         Tic();
408                         Wait(PAUSE);
409                         break;
410
411             case (1):    P1OUT |= GREEN_LED;
412                         Wait(BLINK);
413                         P1OUT &= (~GREEN_LED);
414                         Tac();
415                         Wait(PAUSE);
416                         break;
417
418             case (3):    P1OUT |= BLUE_LED;
419                         Wait(BLINK);
420                         P1OUT &= (~BLUE_LED);
421                         Tuc();
422                         Wait(PAUSE);
423                         break;
424
425             case (4):    P1OUT |= YELLOW_LED;
426                         Wait(BLINK);

```

main.c

```

427         P1OUT &= (~YELLOW_LED);
428         Toc();
429         Wait(PAUSE);
430         break;
431     }
432
433     i = i + 1;
434
435 }
436 while (i <= round);
437 }
438
439 /*****
440 / Essa função espera o jogador aplicar uma resposta para depois julga-la.
441 /
442 / Isto é feito esperando o jogador aplicar uma resposta e depois verificando se
443 / se é a entrada certa para o elemento i da sequencia do array, em que i é o
444 / n-nesimo LED mostrado.
445 *****/
446 int GetAnswer(int sequence[16], int round)
447 {
448     int i = 0;
449     int game_over = 0;
450     // Permanece no loop até que uma resposta errada seja emitida.
451     while (i <= round && game_over == 0)
452     {
453         // Wait(HALF_SEC);
454         if ((P2IN & RED_BTN) == 0)
455         {
456             P1OUT |= RED_LED;
457             if (sequence[i] == 0)
458                 i++;
459             else
460                 game_over = 1;
461             Wait(QUART_SEC);
462             P1OUT &= (~RED_LED);
463         }
464         if ((P2IN & GREEN_BTN) == 0)
465         {
466             P1OUT |= GREEN_LED;
467             if (sequence[i] == 1)
468                 i++;
469             else
470                 game_over = 1;
471             Wait(QUART_SEC);
472             P1OUT &= (~GREEN_LED);
473         }
474         if ((P2IN & BLUE_BTN) == 0)
475         {
476             P1OUT |= BLUE_LED;
477             if (sequence[i] == 3)
478                 i++;
479             else
480                 game_over = 1;
481             Wait(QUART_SEC);
482             P1OUT &= (~BLUE_LED);
483         }
484         if ((P2IN & YELLOW_BTN) == 0)
485         {
486             P1OUT |= YELLOW_LED;
487             if (sequence[i] == 4)
488                 i++;

```



```

489         else
490             game_over = 1;
491             Wait(QUART_SEC);
492             P1OUT &= (~YELLOW_LED);
493     }
494 }
495 // Se a resposta for errada, envia um sinal para a função main() para mostrar
496 // Loss().
497 if (game_over == 0)
498     return 1;
499 // Caso contrário, envia um sinal para a função main() para indicar que a
500 // resposta foi correta e continuar.
501 else
502     return 0;
503 }
504
505
506 /*****
507 /   Essa função pisca o LED verde 8 vezes bem rapido indicando que a sequencia
508 /   inserida é correta.
509 *****/
510 void CorrectAnswer(void)
511 {
512     int i;
513     for (i = 0; i < 8; i++)
514     {
515         P2OUT = (P2OUT ^ GREEN_LED);
516         Wait(CENTI_SEC);
517     }
518 }
519
520 /*****
521 /   Essa função faz com que os LEDs pisquem rapido indicando a vitória
522 *****/
523 void Win(void)
524 {
525     int i;
526     for (i = 0; i < 3; i++)
527     {
528         P1OUT |= RED_LED;
529         Wait(CENTI_SEC);
530         P1OUT &= (~RED_LED);
531         Wait(CENTI_SEC);
532         P1OUT |= GREEN_LED;
533         Wait(CENTI_SEC);
534         P1OUT &= (~GREEN_LED);
535         Wait(CENTI_SEC);
536         P1OUT |= BLUE_LED;
537         Wait(CENTI_SEC);
538         P1OUT &= (~BLUE_LED);
539         Wait(CENTI_SEC);
540         P1OUT |= YELLOW_LED;
541         Wait(CENTI_SEC);
542         P1OUT &= (~YELLOW_LED);
543         Wait(CENTI_SEC);
544     }
545 }
546
547 /*****
548 /   Essa função faz com que o LED vermelho pisque 3 vezes devagar indicando a derrota
549 *****/
550 void Loss(void)

```

main.c

```
551{
552    int i;
553    for (i = 0; i < 3; i++)
554    {
555        P1OUT |= RED_LED;
556        Wait(QUART_SEC);
557        P1OUT &= (~RED_LED);
558        Wait(QUART_SEC);
559    }
560    Wait(ONE_SEC);
561}
562
```