```c
#include <msp430G2553.h>
#include <msp430.h>

// Definindo os LEDs e botões usados para parar o timer e pegar o número aleatório.
#define RED_LED         BIT1                        // Red LED          (P1.1)
#define GREEN_LED       BIT2                        // Green LED        (P1.2)
#define BLUE_LED        BIT4                        // Blue LED         (P1.4)
#define YELLOW_LED      BIT7                        // Yellow LED       (P1.5)

#define RED_BTN         BIT1                        // Red button       (P2.1)
#define GREEN_BTN       BIT2                        // Green button     (P2.2)
#define BLUE_BTN        BIT4                        // Blue button      (P2.4)
#define YELLOW_BTN      BIT5                        // Yellow button    (P2.5)

#define START           BIT3                        // (P1.3)
#define SOUND           BIT0                        // (P1.0)

// Tamanhos de pausas diferentes para a função wait()
#define TEN_MS      1
#define CENTI_SEC   12
#define QUART_SEC   30
#define HALF_SEC    60
#define ONE_SEC     120
#define BLINK       80
#define PAUSE       30

// Timers
#define ENABLE_PINS 0xFFFE
#define ACLK        0x0100
#define SMCLK       0x0200
#define CONTINUOUS  0x0020
#define UP          0x0010

// Numero de rounds para acabar o jogo de acordo com a dificuldade
#define EASY            8
#define NORMAL          10
#define HARD            12
#define EXTREME         16

//Definition of the notes' frequecies in Hertz.
#define c 261
#define d 294
#define e 329
#define f 349
#define g 391
#define gS 415
#define a 440
#define aS 455
#define b 466
#define cH 523
#define cSH 554
#define dH 587
#define dSH 622
#define eH 659
#define fH 698
#define fSH 740
#define gH 784
#define gSH 830
#define aH 880

// Funções para o funcionamento do game
void 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 cicles
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;
```

```
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
    piscaram indicando vitoria.
176             if (round == difficulty)
177             {
178                 game_state = 2;
179             }
180             Wait(TEN_MS);
181         }
182         if (game_state == 2)
183         {
```

```
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
    amostragem.
214 *****************************************************************************/
215 int ChooseDifficulty(void)
216 {
217     P1OUT |= (BLUE_LED | YELLOW_LED | GREEN_LED);
218     int x = 0;
219     int i;
220     while (x < 1)
221     {
222         if ((P2IN & GREEN_BTN) == 0) //
    ------------------------------------------------------------------------------
223         {
224             P1OUT &= (~GREEN_LED);
225             P1OUT &= (~(BLUE_LED | YELLOW_LED));
226             x = EASY;
227             for (i = 0; i < 8; i++)
228             {
229                 P1OUT = (P1OUT ^ GREEN_LED);
230                 Wait(CENTI_SEC);
231             }
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 // cira 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;
```

```
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     }
411     while (i <= round);
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 occured
549     if (P1IFG & START) {
550         P1IFG = 0x00;; // reset the interrupt flag
```

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