MAIN CODE

Group 16

Members:

-Edouard David

-LUCIA BARRANCO MORENO

-MARIA FERNANDA MONTIEL ZAVALA

START

```
1.    /* USER CODE BEGIN Header */
2.  /**
3.    ******************************************************************************
4.    * @file           : main.c
5.    * @brief          : Main program body
6.    ******************************************************************************
7.    * @attention
8.    *
9.    * Copyright (c) 2022 STMicroelectronics.
10.   * All rights reserved.
11.   *
12.   * This software is licensed under terms that can be found in the LICENSE file
13.   * in the root directory of this software component.
14.   * If no LICENSE file comes with this software, it is provided AS-IS.
15.   *
16.   ******************************************************************************
17.   */
18. /* USER CODE END Header */
19. /* Includes ------------------------------------------------------------------*/
20. #include "main.h"
21.
22. /* Private includes ----------------------------------------------------------*/
23. /* USER CODE BEGIN Includes */
24. #include "UTILS.h"
25. #include<stdlib.h>
26.
27. /* USER CODE END Includes */
28.
29. /* Private typedef -----------------------------------------------------------*/
30. /* USER CODE BEGIN PTD */
31.
32. /* USER CODE END PTD */
33.
34. /* Private define ------------------------------------------------------------*/
35. /* USER CODE BEGIN PD */
36.
37.
38.
39. #define game1 49
40. #define game2 50
41. #define game3 51
42.
43. //defines for the melody generation
44. #define len 3 //the number of notes
45. #define pulses_half_second 2000 //0.5s, duration of each note
46.
47. //game 3
48. #define DIMX 20
49. #define DIMY 20
```

```
50.
51. /* USER CODE END PD */
52.
53. /* Private macro -------------------------------------------------------------*/
54. /* USER CODE BEGIN PM */
55.
56. /* USER CODE END PM */
57.
58. /* Private variables ---------------------------------------------------------*/
59. ADC_HandleTypeDef hadc;
60.
61. DAC_HandleTypeDef hdac;
62.
63. I2C_HandleTypeDef hi2c1;
64.
65. TIM_HandleTypeDef htim2;
66. TIM_HandleTypeDef htim3;
67. TIM_HandleTypeDef htim4;
68. TIM_HandleTypeDef htim9;
69. TIM_HandleTypeDef htim11;
70.
71. UART_HandleTypeDef huart2;
72.
73. /* USER CODE BEGIN PV */
74. uint32_t tempCode;
75. uint8_t bitIndex;
76. uint8_t cmd;
77. uint8_t cmdli;
78. uint32_t code;
79. unsigned char controller=5;
80. /* USER CODE END PV */
81.
82. /* Private function prototypes -----------------------------------------------*/
83. void SystemClock_Config(void);
84. static void MX_GPIO_Init(void);
85. static void MX_USART2_UART_Init(void);
86. static void MX_DAC_Init(void);
87. static void MX_TIM4_Init(void);
88. static void MX_TIM3_Init(void);
89. static void MX_ADC_Init(void);
90. static void MX_TIM2_Init(void);
91. static void MX_TIM9_Init(void);
92. static void MX_TIM11_Init(void);
93. static void MX_I2C1_Init(void);
94. /* USER CODE BEGIN PFP */
95.
96. void ON_LED1();
97. void OFF_LED2();
98.
99. void ON_LED2();
100.  void OFF_LED2();
101.
102.  void OFF_LEDS(); //turn off all LEDs
103.  //void EXTI9_5_IRQHandler(void);
104.  void EXTI15_10_IRQHandler(void);
105.  uint8_t function_random(int upper_boundary, int seed);
106.  void TIM4_IRQHandler(void);
107.  void write_message(unsigned char* message, unsigned char message_length);
108.
109.  void delay(unsigned int time);
110.
111.  //Game 3
112.  void init_cart( int Nb_tab,int** tableau);
113.  void deplace_perso( int Nb_perso,int Nb_tab,int** tableau, int* tab_c);
114.  void affiche_carte (int Nb_perso,int Nb_tab,int** tableau, int* tab_c);
115.
116.  /* USER CODE END PFP */
117.
118.  /* Private user code ---------------------------------------------------------*/
119.  /* USER CODE BEGIN 0 */
```

```
120.
121.  //Common variables for both games
122.  unsigned char game_choice;
123.  unsigned char recieved[7]="      ";
124.  unsigned char text[6];
125.
126.  //Common variables for both games : time measurement variables
127.  unsigned short start_time_g1, present_time_g2; //initial time for the game1 (moment
    when LED turned on), present time for game2 (when the counter reaches zero)
128.  int time_player1;   //duration of pressing for player1
129.  int time_player2;   //duration of pressing for player2
130.  uint8_t random_time;  //random time to be used in game 1 or 2
131.
132.  //Game 2 variables
133.  unsigned char i ,End_Of_Melody;
134.  unsigned int potent_value;
135.  unsigned int difficulty;  //level of difficulty to be changed by the user
136.  char sign;
137.  //GAME 2 : melody variables
138.  uint8_t count, state, change;
139.  uint16_t *music;              //pointer to change which melody is being played
140.  uint16_t melody1[len]={4000,2000,1000};  // OCTAVE 3 (DO 125Hz, SI 250Hz) OCTAVE 4(SI
    500Hz)
141.  uint16_t melody2[len]={1000,1000,2000};  // OCTAVE 4  (SI 500Hz,SI 500Hz) OCTAVE 3 (DO
    125Hz)
142.
143.  //Ultra sound variables
144.  unsigned char new_measurement =0;      // variable for the question "has a new
    measuremnt been captured"
145.  int techo1,techo;            //for the echo pulse duration measurement
146.                                                          //techo1 (rising edge, aka
    initial time),
147.                                                          // techo duration of the
    pulse
148.  unsigned char has_10us_passed=0;  //variable for the time measurement using TIM2
149.
    //has_10us_passed=0, 10 us has passed
150.
    //has_10us_passed=1, 100ms has passed
151.
152.  int counter=0;
153.  const float speedOfSound = 0.0343/2;
154.  void ON_LED1() {
155.  //the LED is in  PA12
156.            GPIOA->BSRR |= (1 << 12);
157.  }
158.
159.  void OFF_LED1() {
160.            GPIOA->BSRR |= (1 << 28);
161.  }
162.
163.  void ON_LED2() {
164.  //the LED is in PC10
165.            GPIOC->BSRR |= (1 << 10);
166.  }
167.  void OFF_LED2() {
168.            GPIOC->BSRR |= (1 << 10) << 16;
169.  }
170.
171.  void OFF_LEDS(){
172.            OFF_LED1() ;
173.            OFF_LED2() ;
174.  }
175.
176.  // generate a random number between 1 and  upper_boundary
177.  uint8_t function_random(int upper_boundary, int seed) {
178.
179.            uint8_t randNumber;
180.
181.            //initialise the seed
```

```
182.            srand(seed);
183.
184.            randNumber=(rand() % (upper_boundary-1+ 1))+1;
185.
186.            return randNumber;
187.    }
188.    /*
189.    void EXTI15_10_IRQHandler(void) {
190.            uint32_t readings = EXTI->PR; //reading the state register of exxti
191.            uint32_t button_user_pressed = readings & (1 << 13); //verify bit 13
192.
193.            if (button_user_pressed == (1 << 13)) {
194.                    if (game_choice == game1)
195.                            game_choice = game2;
196.                    else
197.                            game_choice = game1;
198.
199.                    EXTI->PR |= (1 << 0); // clear flag
200.            }
201.    }
202.    */
203.    ////////////////// Handler for milestone 2
204.    void TIM4_IRQHandler(void) {
205.
206.            /// !!! DISCLAIMER !!! : In this part we do not deal with the antirebounce
    effect since
207.            ///                                                     we consider that
    the time to have access to the handler is enough for that
208.
209.            uint32_t readings = TIM4->SR; //retreiving the value of the interruption
210.            // The following lines are a step to be sure that the flage that rose is
    associated with the right channel
211.            uint32_t button_1_edge = readings & (1 << 1); //button_1_edge has the
    condition if button1 has changed value
212.            uint32_t button_2_edge = readings & (1 << 2); //button_2_edge has the
    condition if button2 has changed value
213.
214.
215.            ///////handling block for the player 1
216.            if (button_1_edge == (1 << 1)) // If the event is CH1, button PB6 pressed
    (rising edge)
217.            {
218.                    time_player1 = TIM4->CCR1;    // save the time when the button was
    pressed
219.                    TIM4->SR &= ~(0x0002);
220.            }
221.
222.            ///////handling block for the player 2
223.            if (button_2_edge == (1 << 2)) // If the event is CH2,  button PB7 pressed
    (rising edge)
224.            {
225.                    time_player2 = TIM4->CCR2;    // save the time when the button was
    pressed
226.                    TIM4->SR &= ~(0x0004);
227.            }
228.
229.            TIM4->SR = 0x0000; // Clear all flags (although only CH1 is expected and
    already cleared)
230.    }
231.
232.    ////////////////// Handlers for milestone 3
233.
234.    void TIM2_IRQHandler(void)
235.    {
236.            uint32_t readings= TIM2->SR;
237.            uint32_t channel1_irq = readings & (1 << 1);
238.            uint32_t channel2_irq = readings & (1 << 2);
239.            uint32_t channel3_irq = readings & (1 << 3);
240.
241.            if (channel1_irq ==(1<<1)){   //Melody mode ?
```

```
242.                    TIM2->SR=0;
243.                    TIM2->CNT=0;
244.          }
245.
246.          //IMPROVEMENT N° 1
247.
248.          if (channel2_irq==(1<<2)){             //Ultrasound mode (counting base) ?
249.                    if (has_10us_passed==1){     //10us has passed
250.                              TIM2->SR&=~(1<<(2));
251.                              GPIOA->BSRR |= ((1 << 1)<<16); //clear pin PA1
252.                              TIM2->CCR2+=50000;           //Update CCR2 to count until
    100 ms
253.                              has_10us_passed=0;
254.                    }
255.                    else{                                          //
256.                              TIM2->SR&=~(1<<(2));
257.                              GPIOA->BSRR |= (1 << 1); //set pin PA1
258.                              TIM2->CCR2+=5;               //Update CCR2 to count until
    10 us
259.                              has_10us_passed=1;
260.                    }
261.          }
262.          if(channel3_irq==(1<<3)){   //Ultrasound mode (measurement) ?
263.
264.                    if ((GPIOB->IDR&(1<<1*10))!=0){        // If rising edge
265.                              techo1=TIM2->CCR3;                   // time when the
    pulse starts
266.                    }
267.                    else{
      //falling edge
268.                              techo=TIM2->CCR3-techo1;     //techo=pulse duration
269.                              if (techo<0) techo+= 0x0FFFF;
270.                              new_measurement=1;
271.                              counter++;
272.                    }
273.                    TIM2->SR&=~(1<<(3)); //PB10
274.          }
275.
276. }
277.
278.
279. //handler for the duration of the tone
280. void TIM3_IRQHandler(void)
281. {
282.          uint32_t readings= TIM3->SR;
283.          //if the duration of the tone has passed in channel 2
284.          uint32_t channel2_irq = readings & (1 << 2);
285.          if (channel2_irq==(1 << 2)&&((sign=='+')||(sign=='-')))// check if CH2
    triggered and whether we are in
286.
                 // the melody block code
287.
                 // count==0 only in that block and the one where
288.
                 // the melody has finished
289.          {
290.                    TIM3->SR=0;
291.                    count++;
      //increment the frequency to be generated
292.                    if(count>=len)
293.                    {
294.                              count=0;                                        //if
    we've reached the end of the melody reset the counter
295.                              End_Of_Melody=1;
296.                    }
297.                    TIM2->CCR1=music[count];              //Initialise the TIM2 to
    play the following melody
298.                    TIM2->CNT=0;                                            //reset
    counters of TIM2 AND TIM3
299.                    TIM3->CNT=0;
```

```
300.            }
301.            //TIM3->SR=0;
302.            TIM3->SR=0;
303.    }
304.
305.    void write_message(unsigned char* message, unsigned char message_length){
306.            unsigned char j=0;
307.            for (j=0;j< message_length;j++){
308.                    HAL_UART_Transmit(&huart2, message+j, 1, 10000);
309.            }
310.    }
311.    void delay(unsigned int time){ //delay function using timer 9 (time in ms)
312.            TIM9->CCR1 = time;
313.            TIM9->CNT=0;
314.            TIM9->CR1 |= 0x0001;
315.            while ((TIM9->SR&0x0002)==0);
316.            TIM9->SR = 0;
317.            TIM9->CR1 &= ~(0x0001);
318.
319.    }
320.
321.    //Improvement N°2 : a 2D game controlled using an IR remote controller, we used printf
    in the functions
322.                                    // of this game since it's an improvement
323.    void init_cart( int Nb_tab,int** tableau)
324.    {
325.        int i,j;
326.        for (i = 0 ; i <20; i++)            /*giving values*/
327.        {
328.            for (j = 0 ; j < 20; j++)
329.            {
330.                tableau [i][j] = 0 ;
331.            }
332.        }
333.        tableau [2][3]=5;
334.
335.        tableau [6][5]=3;
336.        tableau [4][5]=2;
337.        tableau [7][17]=2;
338.        tableau [12][7]=2;                  /*0=grass, 1=flower, 2=tree,3=rock,4= key,5=gold
    coin,6=padlock, 7=trap,8=monster*/
339.        tableau [9][8]=6;
340.        tableau [15][4]=8;
341.        tableau [2][18]=8;
342.        tableau [19][13]=8;
343.        tableau [15][6]=2;
344.        tableau [6][11]=3;
345.        tableau [9][9]=1;
346.        tableau [19][4]=5;
347.        tableau [2][3]=5;
348.        tableau [17][3]=5;
349.        tableau [2][9]=5;
350.
351.    }
352.
353.    void deplace_perso( int Nb_perso,int Nb_tab, int** tableau, int* tab_c)
354.    {
355.
356.        int Choice;
357.        int HP;
358.        int Gold_coin;
359.        int x,y;
360.        int key;
361.
362.        key = 0;
363.        HP = 10;
364.        Gold_coin=0;
365.
366.        x=0;
367.        y=0;
```

```
368.      tab_c[0]=x;
369.      tab_c[1]=y;
370.      printf("Welcome to the greed island to move to the right direction tap 6, left 4,
    up 8 , down 2 , to exit 0\n\r ");
371.
372.
373.
374.
375.      while ((HP>0)&&(Gold_coin<10))
376.      {
377.
378.
379.           while (controller==5);
380.           Choice=controller;
381.          switch ( Choice)
382.          {
383.
384.          case 4 :
385.              printf(" You are going to the left\n\r");
386.              y--;
387.              if (y < 0)
388.              {
389.                  printf("You shall not pass\n\r ");
390.                  y++;
391.              }
392.              else
393.              {
394.                  if  ((tableau [x][y]==8)||(tableau[x][y]==7))
395.                  {
396.                      HP--;
397.                      printf("You were beyond a threat thus Yous lost 1 HP, Your current
    HP is %d\n\r",HP);
398.                      tableau [x][y]=0;
399.                  }
400.
401.                  if ((tableau [x][y]==2)|| (tableau [x][y]==3))
402.                  {
403.                      printf ("You are beyond an obstacle take another path\n\r");
404.                      y++;
405.                  }
406.                  if  (tableau [x][y]==5)
407.                  {
408.                      Gold_coin++;
409.                      printf("You found a gold coin keep it up You have currently %d gold
    coin (s)\n\r",Gold_coin);
410.                      tableau [x][y]=0;
411.                  }
412.
413.                  if (tableau [x][y]==6)
414.                  {
415.                      if (key>=1)
416.                      {
417.                          printf("You have some keys so You will open this padlock
    \n\r");
418.                          key--;
419.                          tableau [x][y]=0;
420.                      }
421.                      else
422.                      {
423.                          printf("You cannot pass beyond You don't have any keys \n\r");
424.                          y++;
425.                      }
426.
427.                  }
428.
429.              tab_c[0]=x;
430.              tab_c[1]=y;
431.              affiche_carte (   Nb_perso, Nb_tab,tableau,  tab_c );
432.              }
433.
```

```
434.             break;
435.         case 2 :
436.             printf(" You are going to the up\n\r");
437.
438.             x--;
439.             if ( x<0)
440.             {
441.                 printf("You shall not pass \n\r");
442.                 x++;
443.             }
444.             else
445.             {
446.
447.                 if  ((tableau [x][y]==8)||(tableau [x][y]==7))
448.                 {
449.                     HP--;
450.                     printf("You were beyond a threat thus You lost 1 HP, Your current
    HP is %d \n\r",HP);
451.                     tableau [x][y]=0;
452.                 }
453.
454.                 if ((tableau [x][y]==2)|| (tableau [x][y]==3))
455.                 {
456.                     printf ("You are beyond an obstacle take another path\n\r");
457.                     x++;
458.                 }
459.                 if  (tableau [x][y]==5)
460.                 {
461.                     Gold_coin++;
462.                     printf("You found a gold coin keep it up You have currently %d gold
    coin (s)\n\r",Gold_coin);
463.                     tableau [x][y]=0;
464.                 }
465.
466.                 if (tableau [x][y]==6)
467.                 {
468.                     if (key>=1)
469.                     {
470.                         printf("You have some keys so You will open this padlock \n\r
    ");
471.                         key--;
472.                         tableau [x][y]=0;
473.
474.                     }
475.                     else
476.                     {
477.                         printf("You cannot pass beyond You don't have any keys \n\r");
478.                         x++;
479.
480.                     }
481.
482.                 }
483.
484.                 tab_c[0]=x;
485.                 tab_c[1]=y;
486.
487.                 affiche_carte(   Nb_perso, Nb_tab,tableau,  tab_c );
488.             }
489.             break;
490.         case 6:
491.             printf(" You are going to the right \n\r ");
492.
493.             y++;
494.             if ( y >19)
495.             {
496.                  y--;
497.                 printf("You shall not pass \n\r");
498.             }
499.             else
500.             {
```

```
501.                 if  ((tableau [x][y]==8)||(tableau [x][y]==7))
502.                 {
503.                     HP--;
504.                     printf("You were beyond a threat thus You lost 1 HP, Your current
    HP is %d \n\r",HP);
505.                     tableau [x][y]=0;
506.
507.                 }
508.
509.                 if ((tableau [x][y]==2)|| (tableau [x][y]==3))
510.                 {
511.                     printf ("You are beyond an obstacle take another path \n\r");
512.                     y--;
513.                     printf("\n\r Your coordinate(%d;%d)\n\r",x,y);
514.
515.                 }
516.                 if  (tableau [x][y]==5)
517.                 {
518.                     Gold_coin++;
519.                     printf("You found a gold coin keep it up You have currently %d gold
    coin (s) \n\r",Gold_coin);
520.                     tableau [x][y]=0;
521.
522.                 }
523.
524.                 if (tableau [x][y]==6)
525.                 {
526.                     if (key>=1)
527.                     {
528.                         printf("You have some keys so You will open this padlock \n\r
    ");
529.                         key--;
530.                         tableau [x][y]=0;
531.
532.                     }
533.                     else
534.                     {
535.                         printf("You cannot pass beyond You don't have any keys \n\r ");
536.                         y--;
537.                     }
538.
539.                 }
540.
541.             tab_c[0]=x;
542.             tab_c[1]=y;
543.
544.             affiche_carte (   Nb_perso, Nb_tab,tableau,  tab_c );
545.             }
546.
547.             break;
548.
549.         case 8:
550.             printf(" You are going down \n\r ");
551.
552.             x++;
553.             if ( x >19)
554.             {
555.                 printf("You shall not pass \n\r");
556.                 x--;
557.             }
558.             else
559.             {
560.
561.                 if  ((tableau [x][y]==8)||(tableau [x][y]==7))
562.                 {
563.                     HP--;
564.                     printf("You were beyond a threat thus You lost 1 HP, Your current
    HP is %d \n\r",HP);
565.                     tableau [x][y]=0;
566.                 }
```

```
                if ((tableau [x][y]==2)|| (tableau [x][y]==3))
                {
                    printf ("You are beyond an obstacle take another path \n\r");
                    x--;
                }
                if  (tableau [x][y]==5)
                {
                    Gold_coin++;
                    printf("You found a gold coin keep it up You have currently %d gold coin (s) \n\r",Gold_coin);
                    tableau [x][y]=0;

                }

                if (tableau [x][y]==6)
                {
                    if (key>=1)
                    {
                        printf("You have some keys so You will open this padlock \n\r ");
                        key--;
                        tableau [x][y]=0;

                    }
                    else
                    {
                        printf("You cannot pass beyond You don't have any keys \n\r ");
                    }

                }

            tab_c[0]=x;
            tab_c[1]=y;

            affiche_carte ( Nb_perso, Nb_tab,tableau, tab_c );
            }


            break;
        case 0:
            printf ("You will exit the game ");
        default :
            printf("Wrong number \n\r");
            break;
        }
        controller=5;
    }
    if (HP==0){
        printf("Your greed was not enough \n\r ");
    }
    else if (Gold_coin==10){
        printf("You are one of the seven deadly sins \n\r ");
    }


}
 void affiche_carte (int Nb_perso,int Nb_tab,int** tableau, int* tab_c)
{
    int x=tab_c[0],y=tab_c[1],i,j;

    printf("\n\r Your coordinate(%d;%d)\n\r",x,y);


    for (i = 0 ; i < 20 ; i ++)
    {
        for (j = 0 ; j < 20 ; j ++)
        {
            if (x==i&&y==j)
            {
                printf("X ");
```

```
635.              }
636.              else
637.              {
638.                  printf ("%d ", tableau [i][j]) ;            /* show the map */
639.              }
640.          }
641.           printf ("\n\r\r") ;
642.
643.      }
644.
645.      printf ("\n\r\r") ;
646.
647.  }
648.
649.
650.
651.
652.  /* USER CODE END 0 */
653.
654.  /**
655.    * @brief  The application entry point.
656.    * @retval int
657.    */
658.  int main(void)
659.  {
660.    /* USER CODE BEGIN 1 */
661.
662.    /* USER CODE END 1 */
663.
664.    /* MCU Configuration--------------------------------------------------------*/
665.
666.    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
667.    HAL_Init();
668.
669.    /* USER CODE BEGIN Init */
670.
671.    /* USER CODE END Init */
672.
673.    /* Configure the system clock */
674.    SystemClock_Config();
675.
676.    /* USER CODE BEGIN SysInit */
677.
678.    /* USER CODE END SysInit */
679.
680.    /* Initialize all configured peripherals */
681.    MX_GPIO_Init();
682.    MX_USART2_UART_Init();
683.    MX_DAC_Init();
684.    MX_TIM4_Init();
685.    MX_TIM3_Init();
686.    MX_ADC_Init();
687.    MX_TIM2_Init();
688.    MX_TIM9_Init();
689.    MX_TIM11_Init();
690.    MX_I2C1_Init();
691.    /* USER CODE BEGIN 2 */
692.    HAL_TIM_Base_Start(&htim11);
693.    __HAL_TIM_SET_COUNTER(&htim11, 0);
694.    recieved[0]=0;
695.    HAL_UART_Receive_IT(&huart2, recieved,1);
696.
697.            //Conifgure the user button PC13 digital input 00
698.            GPIOC->MODER &= ~(1 << (13 * 2 + 1)); //coloco 0
699.            GPIOC->MODER &= ~(1 << (13 * 2)); //coloco 0
700.
701.            ////////////////////////////////////////// Configuration of leds
702.            //PA12 as digital output . LED1
703.            GPIOA->MODER &= ~(1 << (12 * 2 + 1)); //coloco 0
704.            GPIOA->MODER |= (1 << (12 * 2)); //coloco 1
```

```
705.
706.            //PC10 as digital output . LED2 !!!HAD TO CHANGE IT, PD2 DOES NOT FUNCTION
707.            GPIOC->MODER &= ~(1 << (10 * 2 + 1)); //coloco 0
708.            GPIOC->MODER |= (1 << (10 * 2)); //coloco 1
709.            /* !!!!! SECOND MILESTONE CONFIGURATION */
710.
711.            ////////////configuration of the pins
712.            //Configure PB6 in  AF configuration 10
713.            GPIOB->MODER |= 1 << (2 * 6 + 1);
714.            GPIOB->MODER &= ~(1 << (2 * 6));
715.            GPIOB->AFR[0] |= (0x02 << (6 * 4)); //AFR[0] to associate PB6 with AF2
    (TIM4,CH1)
716.
717.            //Configure PB7 in  AF configuration 10
718.            GPIOB->MODER |= 1 << (2 * 7 + 1);
719.            GPIOB->MODER &= ~(1 << (2 * 7));
720.            GPIOB->AFR[0] |= (0x02 << (7 * 4)); //AFR[0] to associate PB7 with AF2
    (TIM4,CH2)
721.
722.            //Configure PB6 in PULLDOWN (10)
723.            GPIOB->PUPDR &= ~(1 << (6 * 2));
724.            GPIOB->PUPDR |= (1 << (6 * 2 + 1));
725.            //Configure PB7 in PULLDOWN (10)
726.            GPIOB->PUPDR &= ~(1 << (7 * 2));
727.            GPIOB->PUPDR |= (1 << (7 * 2 + 1));
728.
729.            ////////////////////////Configuration of the timer 4 (used for capturing the
    times of the players) //
730.
731.            TIM4->CR1 = 0;                                              //CEN =0 counter
    disabled,
732.
733.            TIM4->CR2 = 0;                                              //always zero in
    this class
734.            TIM4->SMCR = 0;                                             //always zero in
    this class
735.            TIM4->DIER |= 0x000E; // interrupt for channel 1,2,3 enabled
    (CC3IE=1,CC2IE=1,CC1IE=1) (ch1,2<=>player1,2)(ch3<=>echo signal measurement)
736.            TIM4->CCMR1 |= 0x0101;                              //Setting channel 1 & 2 as
    TIC, CCyS = 01
737.
738.
739.            TIM4->CCER |= 0x0033;                   // rising edge interrupt enabled (ch1
    &ch2), capture enabled
740.                                                                                    //
    Capture enabled CCyE=1
741.
742.            TIM4->CNT = 0;                                              //Initial value of
    the counter
743.            TIM4->PSC = 31999;                                  //Tu= 1ms
744.            TIM4->ARR = 0xFFFF; // Recommendation, in this case we needn't a value of
    reset, we measure time
745.            //Counter enabling
746.
747.
748.            TIM4->EGR |= 0x0001;                                        // UG =1 Clear and
    refresh the counter
749.            TIM4->SR = 0;
      //clearing the counter flags
750.
751.            // Interrupts enabling TIM4_IRQHandler at NVIC (position 30)
752.            NVIC->ISER[0] = (1 << 30);
753.
754.
755.
756.            //configuring the interruption of the pin PC13 with rising edge
757.            SYSCFG->EXTICR[3] &= ~(0x0F << (1 * 4));
758.            SYSCFG->EXTICR[3] |= (2 << (1 * 4));
759.
760.            EXTI->RTSR |= (1 << 13); // activate rising edge PC13
```

```
761.            EXTI->FTSR &= ~(1 << 13); // deactivate falling edge PC13
762.            EXTI->IMR |= (1 << 13); // unmasking the interrupt request
763.            NVIC->ISER[1] |= (1 << (40 - 32)); //NVIC enabling the exti 15_10
764.            ////////////////////////////////////Configuration of the timer used for
    generating melody PA5 (AF01) TIM2 CH1
765.
766.            //Channel 2  is going to be used to measure time TOC without output (5us and
    100ms)
767.            //CHANNEL 3  is going to be used for measuring the duration of the signal
    echo (PB10,AF01)
768.            // PIN PA1 is going to be used to generate the output trigger
769.            //Configure PA1 as digital output 01
770.            GPIOA->MODER&=~(1<<(1*2+1));
771.            GPIOA->MODER|=(1<<(1*2));
772.            //Configure PA5 as an alternate function
773.            GPIOA->MODER&=~(1<<(5*2));
774.            GPIOA->MODER|=(1<<(5*2+1));
775.            // Configure PA5 to AF01 (0001)
776.            GPIOA->AFR[0]|=(1<<(5*4));
777.            //Configure PB10 as an alternate function to read techo
778.            GPIOB->MODER&=~(1<<(10*2));
779.            GPIOB->MODER|=(1<<(10*2+1));
780.            //Configure PB10 to AF01 (0001)
781.            GPIOB->AFR[1]|=(1<<((2*4)));
782.            //Configure the timer
783.            TIM2->CR1 = 0;                                          // ARPE = 0 -> Not
    PWM, it is TOC
784.
      // CEN = 0; Counter off
785.            TIM2->CR2 = 0;                                          // Always 0x0000
    in this subject
786.            TIM2->SMCR = 0;                                         // Always 0x0000
    in this subject
787.            // Setting up the counter functionality : PSC,CNT,ARR
788.            TIM2->PSC=61;                                           //tu = 2 us
789.            TIM2->CNT = 0;                                  //Initial value for CNT
790.            TIM2->ARR = 0xFFFF;                             // Recommended value =FFFF
791.            //IRQ SELECTON
792.            TIM2->DIER = 0x000E;                            //IRQ enabled only for CH1,
    CC1IE=1; CH2 CC2IE=1 and CH3 CC3IE=1
793.            TIM2->CCMR1 = 0x0030;                           //OC1M=011 toggle
794.
      //CC1S=00 TOC,OC1E=0,OC1PE=0 (not PWM,TOC)
795.
      //CC2S=00 TOC, OC2M=000 timing base
796.            TIM2->CCMR2 |= 0x0001;                          //Setting channel 3 as TIC,
    CC3S = 01
797.            TIM2->CCR2=5;                                           //10us
798.            TIM2->CCER = 0x0B00;                            //CC1NP=CC1P=0(TOC),CC1E=1,
    output enabled
799.
      //CC2NP=CC2P=0(TOC),CC2E=0, output disabled
800.
      //CC3NP=CC3P=1(TIC)rising and falling edge interrupt enabled (CH3), capture enabled
    CC3E =1
801.            TIM2->EGR |= 0x0001;                            //UG=1 Update event
802.            TIM2->SR = 0;                                           // Clear all flags
803.            NVIC->ISER[0] |= (1 << 28);             //NVIC enabling the TIM2 GLOBAL IRQ
804.
805.            ////////////////////////timer for the delays
806.
807.            // Internal clock selection: CR1, CR2, SMRC
808.            TIM9->CR1 = 0;                          // ARPE = 0 -> Not PWM, it is TOC
809.                                                                    // CEN = 0;
    Counter off
810.            TIM9->CR2 = 0;                          // Always 0x0000 in this subject
811.            TIM9->SMCR = 0;                         // Always 0x0000 in this subject
812.            // Setting up the counter functionality : PSC,CNT,ARR,and CCR1
813.            TIM9->PSC = 31999;              //Time unit = 1 ms
814.            TIM9->CNT = 0;                          //Initial value for CNT
```

```
815.            TIM9->ARR = 0xFFFF;           // Recommended value =FFFF
816.            TIM9->CCR1 = 3000;            // 3 seconds (preload value since we are in a
    toc configuration
817.            //IRQ SELECTON
818.            TIM9->DIER = 0x0004;          //IRQ enabled only for AND CH2, CC1IE=1,CC2IE=1
819.            // Counter output mode
820.            TIM9->CCMR1 = 0x0000;         //CC1S=CC2S=0 (TOC)
821.                                                                  //OC1M=OC2M = 000
    (no external output)
822.                                                                  //OC1PE=OC2PE =0
    (no preload)
823.
824.            TIM9->CCER = 0;                         //CC1P=CC2P= 0 always for TOC
825.                                                                  //CC1E=CC2E =0 (no
    output)
826.
827.            // Enabling the counter
828.
829.            TIM9->CR1 |= 1;                         //CEN=1 starting the counter
830.            TIM9->EGR |= 1;                         //UG=1 Update event
831.            TIM9->SR = 0;                           // Clear all flags
832.
833.
834.
835.            ////////////////////////////////////Configuration of the timer used as a
    counter TIM3 CH1*
836.
837.            // Internal clock selection: CR1, CR2, SMRC
838.            TIM3->CR1 = 0;                          // ARPE = 0 -> Not PWM, it is TOC
839.                                                                  // CEN = 0;
    Counter off
840.            TIM3->CR2 = 0;                          // Always 0x0000 in this subject
841.            TIM3->SMCR = 0;                         // Always 0x0000 in this subject
842.            // Setting up the counter functionality : PSC,CNT,ARR,and CCR1
843.            TIM3->PSC = 31999;            //Time unit = 1 ms
844.            TIM3->CNT = 0;                          //Initial value for CNT
845.            TIM3->ARR = 0xFFFF;           // Recommended value =FFFF
846.            TIM3->CCR1 = 30000;           // 3 seconds (preload value since we are in a
    toc configuration
847.            //IRQ SELECTON
848.            TIM3->DIER = 0x0004;          //IRQ enabled only for CH2,CC2IE=1
849.            // Counter output mode
850.            TIM3->CCMR1 = 0x0000;         //CC1S=CC2S=0 (TOC)
851.                                                                  //OC1M=OC2M = 000
    (no external output)
852.                                                                  //OC1PE=OC2PE =0
    (no preload)
853.
854.            TIM3->CCER = 0;                         //CC1P=CC2P= 0 always for TOC
855.                                                                  //CC1E=CC2E =0 (no
    output)
856.            NVIC->ISER[0] |= (1 << 29);             //NVIC enabling the TIM3 GLOBAL IRQ
857.
858.
859.            // Enabling the counter
860.
861.            TIM3->CR1 |= 1;                         //CEN=1 starting the counter
862.            TIM3->EGR |= 1;                         //UG=1 Update event
863.            TIM3->SR = 0;                           // Clear all flags
864.
865.            ///////////////Configuration of the ADC
866.
867.            //PA4 as analog
868.            GPIOA->MODER |= (1 << (4 * 2 + 1));
869.            GPIOA->MODER |= (1 << (4 * 2));
870.
871.            ADC1->CR2 &= ~(1 << (1 * 1)); //ADON = 0 (ADC powerd off)
872.            ADC1->CR1 = 0x00000000;       // OVRIE =0 (overrun IRQ disabled)
873.                                                                  //00 resolution =
    12 bits
```

```
                                                                    // 0-Scan disabled
                                                                    // 0- EOCIE : end of conversion interrupt

            ADC1->CR2 = (1 << (10 * 1));// EOCS =1 (EOC to be activated after each conversion)
                                                                                    //DELS = 000 (no delay)
                                                                                    //CONT =0 (single conversion)
            ADC1->SQR1 = 0x00000000;      // 1 channel in the sequence
            ADC1->SQR5 = 0x00000004;      // Channel is AIN4
            ADC1->CR2 |= 0x00000001;      //ADON = 1 (ADC powered on)


            ////////////////////////////////////////// Initialisation of variable


            game_choice = game1;
            difficulty=1000;
            End_Of_Melody=0;
            recieved[0]=0;
            has_10us_passed=1;
            /* USER CODE END SysInit */

    /* USER CODE END 2 */

    /* Infinite loop */
    /* USER CODE BEGIN WHILE */
            while (1)
            {


                    delay(3000);
                    //if needed turn off all LEDS
                    OFF_LEDS();
                    TIM2->CR1 |= 1;                         //CEN=1 starting the counter
                    GPIOA->BSRR |= (1 << 1);

                    write_message((unsigned char*)"\r\n\r Welcome, pass Your hand alongside the Ultrasonic module\r\n\r", sizeof("\r\n\r Welcome, pass Your hand alongside the Ultrasonic module\r\n\r"));

                    delay(2000);
                    counter+=0;
                    while (new_measurement==0);
                    new_measurement=0;
                    has_10us_passed=0;
                    float distance = (techo*2* 100) / 5882;//from clock cycles to microseconds

                    //We used a printf in this case since it's related to the improvement
                    printf ("\r\n\r Distance measured %f cm \r\n\r",distance);

                    delay(1000);
                    if (1<=distance&&distance<=7){ // start the game


                                                            //WAIT 3 SECONDS
                                                            delay(3000);

                                                            unsigned char message[]="\n\r Welcome to the game of reflexes, select: game 1 REACTION TIME (press 1), game 2 COUNTDOWN (press 2), game 3 Greed-Island (press 3) \r\n\r";

                                                            write_message(message, sizeof(message));
                                                            while(recieved[0]==0);
```

```
933.
934.
935.                                                         // PRESS the user button
    here
936.                                                         game_choice=recieved[0];
937.
938.                                                         recieved[0]=0;
939.
940.                                                         switch (game_choice) {
941.                                                                 case game1: //GAME
    1
942.
943.
944.
945.
      write_message((unsigned char*)"\r\n\r game 1 \r\n\r", sizeof("\r\n\r game 1 \r\n\r"));
946.
947.
948.
                TIM4->CR1 |= 0x0001;                                //CEN=1 counter ENabled
949.
                recieved[0]=0;
950.
951.
                //initialisation phase
952.
                time_player2 =0;
953.
                time_player1=0;
954.
955.
                random_time = function_random(5,(int)TIM4->CNT);  // random time between 1
    and 5 ms
956.
                //Waiting time
957.
                delay(random_time*1000);
958.
959.
                start_time_g1 = TIM4->CNT;              //get the present time
960.
961.
                ON_LED1();
962.
963.
                //Wait 1s
964.
                delay(1000);
965.
966.
                OFF_LED1();
967.
968.
                //Deciding which player has won
969.
                if (time_player1 || time_player2) {     //First detect if at least one of them
    has pressed the button
970.
                        if (time_player1) {
971.
                        // To resolve the sign issue
972.
                                if (time_player1 > start_time_g1) {
973.
                                        time_player1 -= start_time_g1;//If the player 1
    has pressed, determine after how long has he/she pressed
974.
                                } else {
975.
                                        time_player1 = start_time_g1- time_player1;
```

```
976.
                                        }
977.
                                }
978.
                                if (time_player2) {
979.
                                // To resolve the sign issue
980.
                                        if (time_player2 > start_time_g1) {
981.
                                                time_player2 -= start_time_g1;//If the player 2 has pressed, determine after how long has he/she pressed
982.
                                        } else {
983.
                                                time_player2 = start_time_g1- time_player2;
984.
                                        }
985.
                                }
986.
                                if (((time_player1 > time_player2)&&(TIM4->CCR2))||((time_player2 > time_player1)&&(!(TIM4->CCR1)))) {//If the time after which player 1 has pressed the button is longer
987.
                                                                        //the condition TIM4->CCR2 is added as a guarantee that the player 2
988.
                                                                        //has pressed his/her button (avoid the situation where time_player2=0)
989.
                                        ON_LED2();
990.
                                        Bin2Ascii(time_player2, &text[0]);
991.
                                        write_message((unsigned char*)"2 ",sizeof("2 "));
992.
                                        write_message(text,sizeof(text));
993.
                                        write_message((unsigned char*)"\n\r\r",sizeof("\n\r\r"));
994.
995.
996.
                                        }
997.
                                if (((time_player2 > time_player1)&&(TIM4->CCR1))||((time_player1 > time_player2)&&(!(TIM4->CCR2))){//If the time after which player 2 has pressed the button is longer
998.
                                                                        //the condition TIM4->CCR2 is added as a guarantee that the player 2
999.
                                                                        //has pressed his/her button (avoid the situation where time_player1=0)
1000.
                                        ON_LED1();
1001.
1002.
1003.
                                        Bin2Ascii(time_player1, &text[0]);
1004.
                                        write_message((unsigned char*)"1 ",sizeof("1 "));
1005.
                                        write_message(text,sizeof(text));
```

```
1006.
                                    write_message((unsigned char*)"\n\r\r",sizeof("\n\r\r")
    );
1007.
1008.
1009.
1010.
                            }
1011.
                }
1012.
                else {
1013.
                            write_message((unsigned char*)"\r\n\rEND of the game no winner no
    loser\r\n\r",sizeof("\r\n\rEND of the game no winner no loser\r\n\r") );
1014.           //
1015.
1016.
                }
1017.
1018.
1019.
1020.
      break;
1021.                                                                    case game2: //GAME
    2
1022.
1023.
      write_message((unsigned char*)"\r\n\r game 2 \r\n\r",sizeof("\r\n\r game 2 \r\n\r") );
1024.
1025.
1026.
      //initialisation
1027.
      time_player2 =0;
1028.
      time_player1=0;
1029.
1030.
      write_message((unsigned char*)"\r\n\rPlease choose the difficulty level
    \r\n\r",sizeof("\r\n\rPlease choose the difficulty level \r\n\r") );
1031.
      write_message((unsigned char*)"You have 2 seconds to do so \r\n\r",sizeof("You have 2
    seconds to do so \r\n\r") );
1032.
1033.
1034.
      //wait 2 seconds for user interface easiness
1035.
      delay(2000);
1036.
1037.
      write_message((unsigned char*)"\t Easy (countdown 2s) \r\n\r",sizeof("\t Easy
    (countdown 2s) \r\n\r") );
1038.
      write_message((unsigned char*)"Potentiometer value
    [0;1330]\r\n\r",sizeof("Potentiometer value [0;1330]\r\n\r") );
1039.
1040.
      write_message((unsigned char*)"\t Normal (countdown 1s) \r\n\r",sizeof("\t Normal
    (countdown 1s) \r\n\r") );
1041.
      write_message((unsigned char*)"Potentiometer value
    [1330;2600]\r\n\r",sizeof("Potentiometer value [1330;2600]\r\n\r") );
1042.
1043.
1044.
      write_message((unsigned char*)"\t Hard (countdown 0.5s) \r\n\r",sizeof("\t Hard
    (countdown 0.5s) \r\n\r") );
```

```
1045.
      write_message((unsigned char*)"Potentiometer value
    [2600;4000]\r\n\r",sizeof("Potentiometer value [2600;4000]\r\n\r") );
1046.
1047.
1048.
      //wait 2 seconds
1049.
      delay(2000);
1050.
1051.
1052.
       // start conversion
1053.
       while ((ADC1->SR&(1<<(6*1)))==0); //While ADONS = 0, i.e, ADC is not ready
1054.
       // to convert, I wait
1055.
       ADC1->CR2|=(1<<30*1);                    //When ADONS = 1, I start conversion
1056.
       //(SWSTART=1)
1057.
       // Wait till conversion is finished
1058.
       while ((ADC1->SR&0x0002)==0);   // If EOC = 0, i.e., the conversion is not
1059.
       // finished, I wait
1060.
       potent_value=ADC1->DR;                              // When EOC=1, I take the result and
    store it in
1061.
       // variable called value
1062.
       //Convert result to string
1063.
       Bin2Ascii(potent_value,text);
1064.
1065.
       write_message((unsigned char*)"potentiometer value :",sizeof("potentiometer value :")
    );
1066.
       write_message((unsigned char*)text,sizeof(text) );
1067.
       write_message((unsigned char*)"\n\r\r",sizeof("\n\r\r") );
1068.
1069.
1070.
       if (potent_value <1330){
1071.
                  write_message((unsigned char*)"You have chosen the easy level
    \r\n\r",sizeof("You have chosen the easy level \r\n\r") );
1072.
1073.
                  //configure the timer,
1074.
                  difficulty=2000;
1075.
1076.
       }
1077.
1078.
       if(1330<=potent_value && potent_value<2600  ){
1079.
                  write_message((unsigned char*)"You have chosen the normal level
    \r\n\r",sizeof("You have chosen the normal level \r\n\r") );
1080.
1081.
                  //configure the timer,
1082.
                  difficulty=1000;
```

```
1083.
1084.
      }
1085.
1086.
       if(2600<=potent_value && potent_value<=4095  ){
1087.
                write_message((unsigned char*)"You have chosen the Hard level
    \r\n\r",sizeof("You have chosen the Hard level \r\n\r") );
1088.
1089.
               //configure the timer,
1090.
                difficulty=500;
1091.
1092.
      }
1093.
1094.
1095.
      random_time = function_random(3,(int)TIM4->CNT); // random time between 1 and 3, in
    this casE it'll be the moment
1096.
                                    // starting from which the countdown message will
    disappear
1097.
      //configure the difficulty
1098.
1099.
      delay(difficulty);// input difficulty
1100.
1101.
      TIM4->CR1 |= 0x0001;                              //CEN=1 counter ENabled
1102.
1103.
      unsigned char k=57;           //57 is the ascii code of the char 9, no code ascii for
    10
1104.
      for (i =10 ; i> 0; i--) {
1105.
1106.
              if (i==10){
1107.
                        write_message((unsigned char*)("10\n\r\r"),sizeof("10\n\r\r") );
1108.
                        delay(difficulty);
1109.
1110.
              }
1111.
              if (i <= random_time) {
1112.
                        write_message((unsigned char*)" \n\r\r",sizeof(" \n\r\r") );
1113.
                        delay(difficulty);
1114.
1115.
              } else if (i!=10) {
1116.
                        write_message((unsigned char*)(&k),sizeof(k) ); // the respective
    values of k and i are the same from when i=9
1117.
                        write_message((unsigned char*)"\n\r\r",sizeof("\n\r\r") );
1118.
                        k--;                //decreasing its value until it reaches 49
    (ascii code of 1)
1119.
1120.
                        delay(difficulty);
```

```
1121.
1122.
                }
1123.
                if (i==1){
1124.
                          //Store the value of the present time
1125.
                          present_time_g2 = TIM4->CNT;
1126. //
                          delay(difficulty);
1127.
                }
1128.
                //Wait 1 se before next iteration
1129.
1130.
      }
1131.
1132.
      // Waiting 2 seconds after the end of the countdown
1133.
      delay(2000);
1134.
1135.
      //Deciding which player has won
1136.
      if (time_player1 || time_player2) {    //First detect if at least one of them has
    pressed the button
1137.
                if (time_player1) {
1138.
                          // To resolve the sign issue
1139.
                          if (time_player1 > present_time_g2) {//pressed after the counter
    reaches zero
1140.
                                    time_player1 -= present_time_g2;//If the player 1 has
    pressed, determine after how long has he/she pressed
1141.
                                    sign='+';
1142.
                          } else {  //pressed before the counter reaches zero
1143.
                                    time_player1 = present_time_g2 - time_player1;
1144.
                                    sign='-';
1145.
                          }
1146.
                }
1147.
                if (time_player2) {
1148.
                          // To resolve the sign issue
1149.
                          if (time_player2 > present_time_g2) { //pressed after the counter
    reaches zero
1150.
                                    time_player2 -= present_time_g2;//If the player 2 has
    pressed, determine after how long has he/she pressed
1151.
                                    sign='+';
1152.
                          } else {   //pressed before the counter reaches zero
1153.
                                    time_player2 = present_time_g2- time_player2;
1154.
                                    sign='-';
```

```
1155.
                        }
1156.
                }
1157.
                if (((time_player1 > time_player2)&&(time_player2))||((time_player2 >
    time_player1)&&(!(time_player1)))) {//If the time after which player 1 has pressed the
    button is longer
1158.



                  //the condition time_player2 is added as a guarantee that the player 2
1159.



                  //has pressed his/her button (avoid the situation where time_player2=0)
1160.
                        ON_LED2();
1161.
                        Bin2Ascii(time_player2, &text[0]);
1162.
                        write_message((unsigned char*)"2 ",sizeof("2 ")); //print that the
    player 2 has won
1163.
                        write_message((unsigned char*)(&sign),sizeof(sign));
1164.
                        write_message(text,sizeof(text));
1165.
                        write_message((unsigned char*)"\n\r\r",sizeof("\n\r\r") );
1166.
1167.
1168.
                }
1169.
                if (((time_player2 > time_player1)&&(time_player1))||((time_player1 >
    time_player2)&&(!(time_player2)))){//If the time after which player 2 has pressed the
    button is longer
1170.



                  //the condition time_player1 is added as a guarantee that the player 1
1171.



                  //has pressed his/her button (avoid the situation where time_player1=0)
1172.
                        ON_LED1();
1173.
                        Bin2Ascii(time_player1, &text[0]);
1174.
                        write_message((unsigned char*)"1 ",sizeof("1 ")); //print that the
    player 1 has won
1175.
                        write_message((unsigned char*)(&sign),sizeof(sign) );
1176.
                        write_message(text,sizeof(text));
1177.
                        write_message((unsigned char*)"\n\r\r",sizeof("\n\r\r") );
1178.
1179.
1180.
                }
1181.
1182.
```

```
1183.
                //generating melody depending on whether the button was pressed before or
    after the end of the countdown
1184.
                //PA5 timer2 channel 1
1185.
                TIM3->CNT=0;
1186.
                TIM3->CCR1 = 0;
1187.
                TIM2->CCER|=(1<<(0));//enabling hardware output
1188.
                delay(500);
1189.
                if (sign =='-' ){ //melody 1 is the melody when we press the button before
    the end of the countdown
1190.
1191.
                                        music=melody1;
1192.
                                        count=0;
1193.
                                        TIM2->CCR1=music[count];
1194.
                                        TIM2->CR1=1;
1195.
                                        TIM3->CCR2=pulses_half_second;
1196.
                                        TIM3->CR1=1;
1197.
1198.
1199.
                }
1200.
                else if (sign =='+'){
1201.
1202.
                                        music=melody2;
1203.
                                        count=0;
1204.
                                        TIM2->CCR1=music[count];
1205.
                                        TIM2->CR1=1;
1206.
                                        TIM3->CCR2=pulses_half_second;
1207.
                                        TIM3->CR1=1;
1208.
1209.
                }
1210.
                while(End_Of_Melody!=1);
1211.
                delay(500);
1212.
                TIM2->CCER&=~(1<<(0));//disabling hardware output
1213.
                TIM2->CR1=0;
1214.
                TIM3->CR1=0;
1215.
1216.
      }
1217.
      else {
1218.
                write_message((unsigned char*)"END of the game no winner no
    loser\r\n\r",sizeof("END of the game no winner no loser\r\n\r") );
1219.
```

```
1220.
      }
1221.
1222.
1223.
1224.
                break;
1225.
1226.                                                                                    //Improvement N°2
    : a 2D game controlled using an IR remote controller, we used printf in the functions
1227.
          // of this game since it's an improvement
1228.                                                                                  case game3:
1229.
      write_message((unsigned char*)"\r\n\r game 3 \r\n\r",sizeof("\r\n\r game 3 \r\n\r") );
1230.
1231.
1232.                                                                                        int*
    tab_c=malloc ( 2 * sizeof ( float* ) );
1233.
1234.
    int** tableau ;
1235.
    tableau = malloc ( DIMX * sizeof ( float* ) );
1236.                                                                                          int
    i;
1237.                                                                                          for
    ( i = 0; i < DIMX; i++ )
1238.                                                                                          {
1239.
    tableau[i] = calloc ( DIMY, sizeof ( float ) );
1240.                                                                                          }
1241.
1242.
1243.
    tab_c[0]=0;
1244.
    tab_c[1]=0;
1245.
1246.
    init_cart( 20,tableau);
1247.
    affiche_carte (2,20,tableau, tab_c);
1248.
    delay(3000);
1249.
1250.
    deplace_perso( 2,20,tableau,tab_c);
1251.                                                                                      break;
1252.
1253.                                                                          }
1254.
1255.
1256.           }
1257.
1258.
1259.           }
1260.
1261.     /* USER CODE END WHILE */
1262.
1263.     /* USER CODE BEGIN 3 */
1264.
1265.   /* USER CODE END 3 */
1266. }
1267.
1268. /**
1269.   * @brief System Clock Configuration
1270.   * @retval None
1271.   */
1272. void SystemClock_Config(void)
```

```
1273. {
1274.   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
1275.   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
1276.
1277.   /** Configure the main internal regulator output voltage
1278.   */
1279.   __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
1280.   /** Initializes the RCC Oscillators according to the specified parameters
1281.   * in the RCC_OscInitTypeDef structure.
1282.   */
1283.   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
1284.   RCC_OscInitStruct.HSIState = RCC_HSI_ON;
1285.   RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
1286.   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
1287.   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
1288.   RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL6;
1289.   RCC_OscInitStruct.PLL.PLLDIV = RCC_PLL_DIV3;
1290.   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
1291.   {
1292.     Error_Handler();
1293.   }
1294.   /** Initializes the CPU, AHB and APB buses clocks
1295.   */
1296.   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
1297.                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
1298.   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
1299.   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
1300.   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
1301.   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
1302.
1303.   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
1304.   {
1305.     Error_Handler();
1306.   }
1307. }
1308.
1309. /**
1310.   * @brief ADC Initialization Function
1311.   * @param None
1312.   * @retval None
1313.   */
1314. static void MX_ADC_Init(void)
1315. {
1316.
1317.   /* USER CODE BEGIN ADC_Init 0 */
1318.
1319.   /* USER CODE END ADC_Init 0 */
1320.
1321.   ADC_ChannelConfTypeDef sConfig = {0};
1322.
1323.   /* USER CODE BEGIN ADC_Init 1 */
1324.
1325.   /* USER CODE END ADC_Init 1 */
1326.   /** Configure the global features of the ADC (Clock, Resolution, Data Alignment and
    number of conversion)
1327.   */
1328.   hadc.Instance = ADC1;
1329.   hadc.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
1330.   hadc.Init.Resolution = ADC_RESOLUTION_12B;
1331.   hadc.Init.DataAlign = ADC_DATAALIGN_RIGHT;
1332.   hadc.Init.ScanConvMode = ADC_SCAN_DISABLE;
1333.   hadc.Init.EOCSelection = ADC_EOC_SEQ_CONV;
1334.   hadc.Init.LowPowerAutoWait = ADC_AUTOWAIT_DISABLE;
1335.   hadc.Init.LowPowerAutoPowerOff = ADC_AUTOPOWEROFF_DISABLE;
1336.   hadc.Init.ChannelsBank = ADC_CHANNELS_BANK_A;
1337.   hadc.Init.ContinuousConvMode = DISABLE;
1338.   hadc.Init.NbrOfConversion = 1;
1339.   hadc.Init.DiscontinuousConvMode = DISABLE;
1340.   hadc.Init.ExternalTrigConv = ADC_SOFTWARE_START;
1341.   hadc.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
```

```
1342.   hadc.Init.DMAContinuousRequests = DISABLE;
1343.   if (HAL_ADC_Init(&hadc) != HAL_OK)
1344.   {
1345.     Error_Handler();
1346.   }
1347.   /** Configure for the selected ADC regular channel its corresponding rank in the
    sequencer and its sample time.
1348.   */
1349.   sConfig.Channel = ADC_CHANNEL_4;
1350.   sConfig.Rank = ADC_REGULAR_RANK_1;
1351.   sConfig.SamplingTime = ADC_SAMPLETIME_4CYCLES;
1352.   if (HAL_ADC_ConfigChannel(&hadc, &sConfig) != HAL_OK)
1353.   {
1354.     Error_Handler();
1355.   }
1356.   /* USER CODE BEGIN ADC_Init 2 */
1357.
1358.   /* USER CODE END ADC_Init 2 */
1359.
1360. }
1361.
1362. /**
1363.   * @brief DAC Initialization Function
1364.   * @param None
1365.   * @retval None
1366.   */
1367. static void MX_DAC_Init(void)
1368. {
1369.
1370.   /* USER CODE BEGIN DAC_Init 0 */
1371.
1372.   /* USER CODE END DAC_Init 0 */
1373.
1374.   DAC_ChannelConfTypeDef sConfig = {0};
1375.
1376.   /* USER CODE BEGIN DAC_Init 1 */
1377.
1378.   /* USER CODE END DAC_Init 1 */
1379.   /** DAC Initialization
1380.   */
1381.   hdac.Instance = DAC;
1382.   if (HAL_DAC_Init(&hdac) != HAL_OK)
1383.   {
1384.     Error_Handler();
1385.   }
1386.   /** DAC channel OUT2 config
1387.   */
1388.   sConfig.DAC_Trigger = DAC_TRIGGER_NONE;
1389.   sConfig.DAC_OutputBuffer = DAC_OUTPUTBUFFER_ENABLE;
1390.   if (HAL_DAC_ConfigChannel(&hdac, &sConfig, DAC_CHANNEL_2) != HAL_OK)
1391.   {
1392.     Error_Handler();
1393.   }
1394.   /* USER CODE BEGIN DAC_Init 2 */
1395.
1396.   /* USER CODE END DAC_Init 2 */
1397.
1398. }
1399.
1400. /**
1401.   * @brief I2C1 Initialization Function
1402.   * @param None
1403.   * @retval None
1404.   */
1405. static void MX_I2C1_Init(void)
1406. {
1407.
1408.   /* USER CODE BEGIN I2C1_Init 0 */
1409.
1410.   /* USER CODE END I2C1_Init 0 */
```

```
1411.
1412.   /* USER CODE BEGIN I2C1_Init 1 */
1413.
1414.   /* USER CODE END I2C1_Init 1 */
1415.   hi2c1.Instance = I2C1;
1416.   hi2c1.Init.ClockSpeed = 400000;
1417.   hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
1418.   hi2c1.Init.OwnAddress1 = 0;
1419.   hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
1420.   hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
1421.   hi2c1.Init.OwnAddress2 = 0;
1422.   hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
1423.   hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
1424.   if (HAL_I2C_Init(&hi2c1) != HAL_OK)
1425.   {
1426.     Error_Handler();
1427.   }
1428.   /* USER CODE BEGIN I2C1_Init 2 */
1429.
1430.   /* USER CODE END I2C1_Init 2 */
1431.
1432. }
1433.
1434. /**
1435.   * @brief TIM2 Initialization Function
1436.   * @param None
1437.   * @retval None
1438.   */
1439. static void MX_TIM2_Init(void)
1440. {
1441.
1442.   /* USER CODE BEGIN TIM2_Init 0 */
1443.
1444.   /* USER CODE END TIM2_Init 0 */
1445.
1446.   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
1447.   TIM_MasterConfigTypeDef sMasterConfig = {0};
1448.
1449.   /* USER CODE BEGIN TIM2_Init 1 */
1450.
1451.   /* USER CODE END TIM2_Init 1 */
1452.   htim2.Instance = TIM2;
1453.   htim2.Init.Prescaler = 0;
1454.   htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
1455.   htim2.Init.Period = 65535;
1456.   htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1457.   htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1458.   if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
1459.   {
1460.     Error_Handler();
1461.   }
1462.   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1463.   if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
1464.   {
1465.     Error_Handler();
1466.   }
1467.   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
1468.   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
1469.   if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
1470.   {
1471.     Error_Handler();
1472.   }
1473.   /* USER CODE BEGIN TIM2_Init 2 */
1474.
1475.   /* USER CODE END TIM2_Init 2 */
1476.
1477. }
1478.
1479. /**
1480.   * @brief TIM3 Initialization Function
```

```
1481.   * @param None
1482.   * @retval None
1483.   */
1484. static void MX_TIM3_Init(void)
1485. {
1486.
1487.   /* USER CODE BEGIN TIM3_Init 0 */
1488.
1489.   /* USER CODE END TIM3_Init 0 */
1490.
1491.   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
1492.   TIM_MasterConfigTypeDef sMasterConfig = {0};
1493.
1494.   /* USER CODE BEGIN TIM3_Init 1 */
1495.
1496.   /* USER CODE END TIM3_Init 1 */
1497.   htim3.Instance = TIM3;
1498.   htim3.Init.Prescaler = 0;
1499.   htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
1500.   htim3.Init.Period = 65535;
1501.   htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1502.   htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1503.   if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
1504.   {
1505.     Error_Handler();
1506.   }
1507.   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1508.   if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
1509.   {
1510.     Error_Handler();
1511.   }
1512.   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
1513.   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
1514.   if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
1515.   {
1516.     Error_Handler();
1517.   }
1518.   /* USER CODE BEGIN TIM3_Init 2 */
1519.
1520.   /* USER CODE END TIM3_Init 2 */
1521.
1522. }
1523.
1524. /**
1525.   * @brief TIM4 Initialization Function
1526.   * @param None
1527.   * @retval None
1528.   */
1529. static void MX_TIM4_Init(void)
1530. {
1531.
1532.   /* USER CODE BEGIN TIM4_Init 0 */
1533.
1534.   /* USER CODE END TIM4_Init 0 */
1535.
1536.   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
1537.   TIM_MasterConfigTypeDef sMasterConfig = {0};
1538.   TIM_IC_InitTypeDef sConfigIC = {0};
1539.
1540.   /* USER CODE BEGIN TIM4_Init 1 */
1541.
1542.   /* USER CODE END TIM4_Init 1 */
1543.   htim4.Instance = TIM4;
1544.   htim4.Init.Prescaler = 0;
1545.   htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
1546.   htim4.Init.Period = 65535;
1547.   htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1548.   htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1549.   if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
1550.   {
```

```
1551.     Error_Handler();
1552.   }
1553.   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1554.   if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
1555.   {
1556.     Error_Handler();
1557.   }
1558.   if (HAL_TIM_IC_Init(&htim4) != HAL_OK)
1559.   {
1560.     Error_Handler();
1561.   }
1562.   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
1563.   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
1564.   if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
1565.   {
1566.     Error_Handler();
1567.   }
1568.   sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
1569.   sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
1570.   sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
1571.   sConfigIC.ICFilter = 0;
1572.   if (HAL_TIM_IC_ConfigChannel(&htim4, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
1573.   {
1574.     Error_Handler();
1575.   }
1576.   /* USER CODE BEGIN TIM4_Init 2 */
1577.
1578.   /* USER CODE END TIM4_Init 2 */
1579.
1580. }
1581.
1582. /**
1583.   * @brief TIM9 Initialization Function
1584.   * @param None
1585.   * @retval None
1586.   */
1587. static void MX_TIM9_Init(void)
1588. {
1589.
1590.   /* USER CODE BEGIN TIM9_Init 0 */
1591.
1592.   /* USER CODE END TIM9_Init 0 */
1593.
1594.   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
1595.   TIM_MasterConfigTypeDef sMasterConfig = {0};
1596.
1597.   /* USER CODE BEGIN TIM9_Init 1 */
1598.
1599.   /* USER CODE END TIM9_Init 1 */
1600.   htim9.Instance = TIM9;
1601.   htim9.Init.Prescaler = 0;
1602.   htim9.Init.CounterMode = TIM_COUNTERMODE_UP;
1603.   htim9.Init.Period = 65535;
1604.   htim9.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1605.   htim9.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1606.   if (HAL_TIM_Base_Init(&htim9) != HAL_OK)
1607.   {
1608.     Error_Handler();
1609.   }
1610.   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1611.   if (HAL_TIM_ConfigClockSource(&htim9, &sClockSourceConfig) != HAL_OK)
1612.   {
1613.     Error_Handler();
1614.   }
1615.   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
1616.   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
1617.   if (HAL_TIMEx_MasterConfigSynchronization(&htim9, &sMasterConfig) != HAL_OK)
1618.   {
1619.     Error_Handler();
1620.   }
```

```
1621.   /* USER CODE BEGIN TIM9_Init 2 */
1622.
1623.   /* USER CODE END TIM9_Init 2 */
1624.
1625. }
1626.
1627. /**
1628.   * @brief TIM11 Initialization Function
1629.   * @param None
1630.   * @retval None
1631.   */
1632. static void MX_TIM11_Init(void)
1633. {
1634.
1635.   /* USER CODE BEGIN TIM11_Init 0 */
1636.
1637.   /* USER CODE END TIM11_Init 0 */
1638.
1639.   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
1640.
1641.   /* USER CODE BEGIN TIM11_Init 1 */
1642.
1643.   /* USER CODE END TIM11_Init 1 */
1644.   htim11.Instance = TIM11;
1645.   htim11.Init.Prescaler = 31;
1646.   htim11.Init.CounterMode = TIM_COUNTERMODE_UP;
1647.   htim11.Init.Period = 65535;
1648.   htim11.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
1649.   htim11.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
1650.   if (HAL_TIM_Base_Init(&htim11) != HAL_OK)
1651.   {
1652.     Error_Handler();
1653.   }
1654.   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
1655.   if (HAL_TIM_ConfigClockSource(&htim11, &sClockSourceConfig) != HAL_OK)
1656.   {
1657.     Error_Handler();
1658.   }
1659.   /* USER CODE BEGIN TIM11_Init 2 */
1660.
1661.   /* USER CODE END TIM11_Init 2 */
1662.
1663. }
1664.
1665. /**
1666.   * @brief USART2 Initialization Function
1667.   * @param None
1668.   * @retval None
1669.   */
1670. static void MX_USART2_UART_Init(void)
1671. {
1672.
1673.   /* USER CODE BEGIN USART2_Init 0 */
1674.
1675.   /* USER CODE END USART2_Init 0 */
1676.
1677.   /* USER CODE BEGIN USART2_Init 1 */
1678.
1679.   /* USER CODE END USART2_Init 1 */
1680.   huart2.Instance = USART2;
1681.   huart2.Init.BaudRate = 115200;
1682.   huart2.Init.WordLength = UART_WORDLENGTH_8B;
1683.   huart2.Init.StopBits = UART_STOPBITS_1;
1684.   huart2.Init.Parity = UART_PARITY_NONE;
1685.   huart2.Init.Mode = UART_MODE_TX_RX;
1686.   huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
1687.   huart2.Init.OverSampling = UART_OVERSAMPLING_16;
1688.   if (HAL_UART_Init(&huart2) != HAL_OK)
1689.   {
1690.     Error_Handler();
```

```
1691.   }
1692.   /* USER CODE BEGIN USART2_Init 2 */
1693.
1694.   /* USER CODE END USART2_Init 2 */
1695.
1696. }
1697.
1698. /**
1699.   * @brief GPIO Initialization Function
1700.   * @param None
1701.   * @retval None
1702.   */
1703. static void MX_GPIO_Init(void)
1704. {
1705.   GPIO_InitTypeDef GPIO_InitStruct = {0};
1706.
1707.   /* GPIO Ports Clock Enable */
1708.   __HAL_RCC_GPIOC_CLK_ENABLE();
1709.   __HAL_RCC_GPIOH_CLK_ENABLE();
1710.   __HAL_RCC_GPIOA_CLK_ENABLE();
1711.   __HAL_RCC_GPIOB_CLK_ENABLE();
1712.
1713.   /*Configure GPIO pin : B1_Pin */
1714.   GPIO_InitStruct.Pin = B1_Pin;
1715.   GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
1716.   GPIO_InitStruct.Pull = GPIO_NOPULL;
1717.   HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);
1718.
1719.   /*Configure GPIO pin : PB11 */
1720.   GPIO_InitStruct.Pin = GPIO_PIN_11;
1721.   GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
1722.   GPIO_InitStruct.Pull = GPIO_NOPULL;
1723.   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
1724.
1725.   /* EXTI interrupt init*/
1726.   HAL_NVIC_SetPriority(EXTI15_10_IRQn, 0, 0);
1727.   HAL_NVIC_EnableIRQ(EXTI15_10_IRQn);
1728.
1729. }
1730.
1731. /* USER CODE BEGIN 4 */
1732. void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart){
1733.           HAL_UART_Receive_IT(huart,recieved,1);
1734.           if (recieved[0]=='X'){
1735.                     NVIC_SystemReset(); //software reset
1736.           }
1737. }
1738.
1739. //Improvement N°3 : an IR receiver transmitter module decoder
1740. //Callback for the Infra-red module using NEC Transmission Protocol
1741.
1742. void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
1743. {
1744.   if(GPIO_Pin == GPIO_PIN_11)
1745.   {
1746.     if (__HAL_TIM_GET_COUNTER(&htim11) > 8000)//if falling edge after 800µs (meaning
    the start of the signal)
1747.     {
1748.       tempCode = 0;
1749.       bitIndex = 0;
1750.     }
1751.     else if (__HAL_TIM_GET_COUNTER(&htim11) > 1700) // a 562.5µs pulse burst followed
    by a 1687.5µs space, total transmission >1700µs
1752.     {
1753.       tempCode |= (1UL << (31-bitIndex));   // write 1
1754.       bitIndex++;
1755.     }
1756.     else if (__HAL_TIM_GET_COUNTER(&htim11) > 1000) // a 562.5µs pulse burst followed
    by a 562.5µs space, total transmission >1000µs
1757.     {
```

```
1758.       tempCode &= ~(1UL << (31-bitIndex));  // write 0
1759.       bitIndex++;
1760.     }
1761.     if(bitIndex == 32)
1762.     {
1763.       cmdli = ~tempCode; // Logical inverted last 8 bits representing the logical
    inverse of the command (eg. command = 1)
1764.       cmd = tempCode >> 8; // Second last 8 bits,
1765.       if(cmdli == cmd) // Check for errors of transmission
1766.       {
1767.         code = tempCode; // If no bit errors
1768.
1769.        switch (code)
1770.         {
1771.
1772.
1773.
1774.           case (-2139114421):
1775.                                           controller=0;
1776.             break;
1777.
1778.           case (-2139121561):
1779.                                           controller=1;
1780.             break;
1781.
1782.           case (-2139124621):
1783.                                           controller=2;
1784.             break;
1785.
1786.           case (-2139112126):
1787.                                           controller=3;
1788.             break;
1789.
1790.           case (-2139125641):
1791.                                           controller=4;
1792.             break;
1793.
1794.           case (-2139120541):
1795.                                           controller=5;
1796.             break;
1797.
1798.           case (-2139116206):
1799.                                           controller=6;
1800.             break;
1801.
1802.           case (-2139119266):
1803.                                           controller=7;
1804.             break;
1805.
1806.           case (-2139118246):
1807.                                           controller=8;
1808.             break;
1809.
1810.           case (-2139117226):
1811.                                           controller=9;
1812.             break;
1813.
1814.           default :
1815.             break;
1816.         }
1817.
1818.       }
1819.     bitIndex = 0;
1820.     }
1821.    __HAL_TIM_SET_COUNTER(&htim11, 0);
1822.   }
1823. }
1824. /* USER CODE END 4 */
1825.
1826. /**
```

```
1827.   * @brief  This function is executed in case of error occurrence.
1828.   * @retval None
1829.   */
1830. void Error_Handler(void)
1831. {
1832.   /* USER CODE BEGIN Error_Handler_Debug */
1833.                                 /* User can add his own implementation to report the HAL
    error return state */
1834.                                 __disable_irq();
1835.                                 while (1) {
1836.                                 }
1837.   /* USER CODE END Error_Handler_Debug */
1838. }
1839.
1840. #ifdef  USE_FULL_ASSERT
1841. /**
1842.   * @brief  Reports the name of the source file and the source line number
1843.   *         where the assert_param error has occurred.
1844.   * @param  file: pointer to the source file name
1845.   * @param  line: assert_param error line source number
1846.   * @retval None
1847.   */
1848. void assert_failed(uint8_t *file, uint32_t line)
1849. {
1850.   /* USER CODE BEGIN 6 */
1851.   /* User can add his own implementation to report the file name and line number,
1852.      ex: printf("Wrong parameters value: file %s on line %d\r\n\r", file, line) */
1853.   /* USER CODE END 6 */
1854. }
1855. #endif /* USE_FULL_ASSERT */
1856.
1857.
```