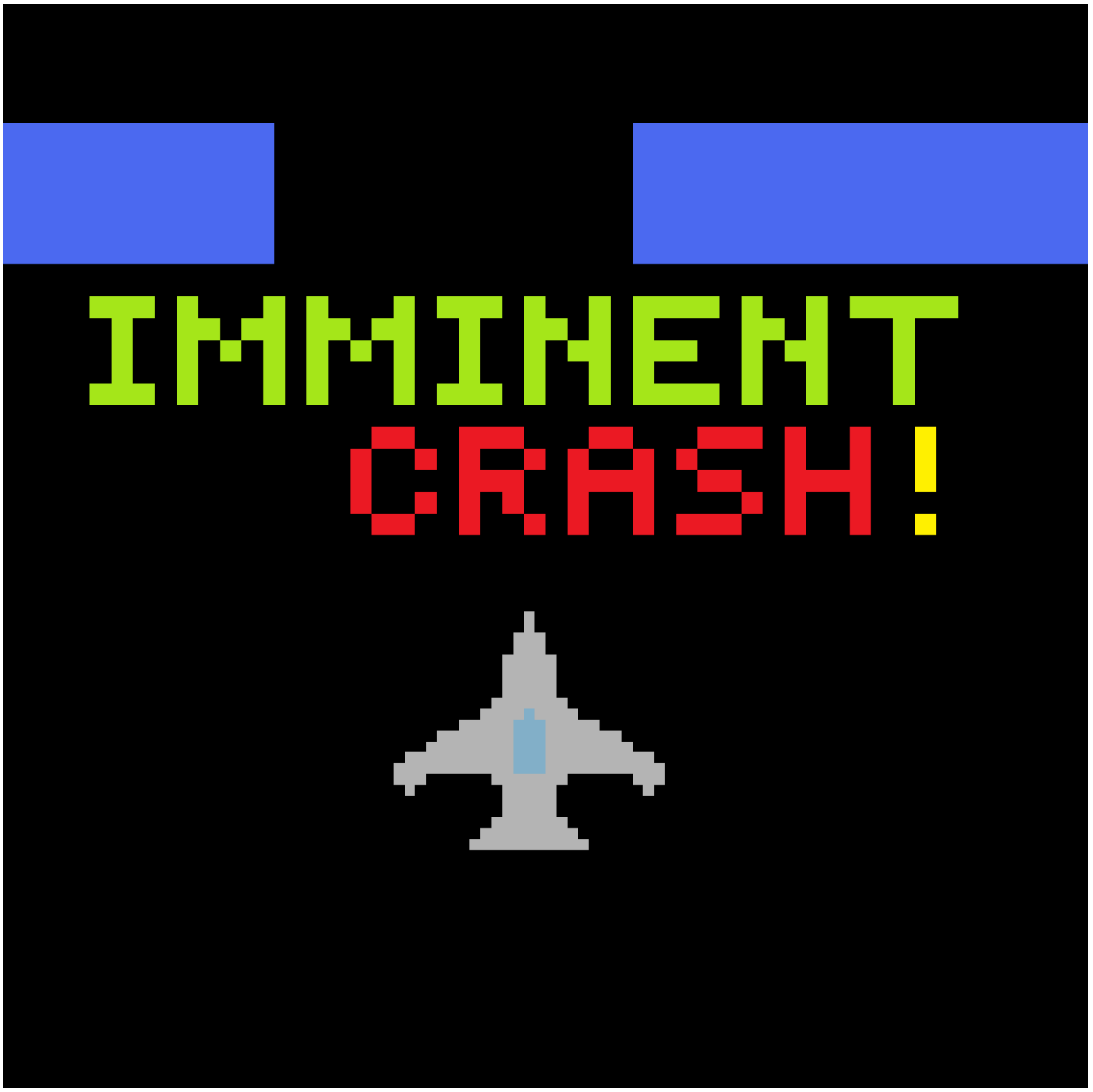


Proyecto Microcontrolador - Sistemas Electrónicos GIERM

Fernando Román Hidalgo
Marta Barroso Infante

ÍNDICE

1. INTRODUCCIÓN

Para superar esta segunda parte de la asignatura de Sistemas Electrónicos, se ha elegido como idea de proyecto realizar un videojuego para demostrar los conocimientos adquiridos durante el curso sobre microcontroladores.

El juego se ha titulado **"Imminent Crash!"**, inspirado en un videojuego antiguo muy conocido, el *1942*, donde un avión que es manejado por el jugador, debe esquivar y derrotar los aviones enemigos que intentan destruirlo. Para este trabajo se ha realizado una adaptación adecuada a las posibilidades de la placa con la que se trabaja, teniendo en cuenta las limitaciones del Educational Boosterpack.

2. REGLAS DEL JUEGO

El desarrollo del juego ocurre de manera muy sencilla. El jugador maneja un pequeño **avión** situado en la parte inferior de la pantalla, y podrá escoger entre **cinco niveles** de dificultad un mapa en el que se desarrollará la aventura.

Una vez escogido, haciendo uso de joystick, debe mover el avión a la izquierda o a la derecha tratando de **no chocarse** con los muros que van apareciendo por pantalla, pasando por unos pequeños huecos. Cada vez que se pase por uno de ellos, el jugador **acumula un punto**, sumándose así su puntuación hasta que se choque y por lo tanto, finalice la partida.

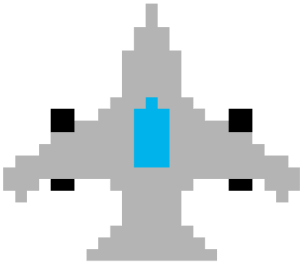
Cuando esto ocurra, tras aparecer **"Game over"** por pantalla, el jugador podrá saber cuál ha sido la puntuación obtenida en la partida, teniendo la posibilidad de cambiar de mapa, o bien reiniciar una nueva partida.

3. DESARROLLO DEL CÓDIGO

Para realizar el código de este trabajo, se ha intentado ir trabajando por partes, es decir, se han puesto varios objetivos como pueden ser el diseño del avión, la configuración de los mapas, la puntuación, etc, que se explicarán en los apartados posteriores con mayor detalle.

3.1 DISEÑO DEL AVIÓN Y SU MOVIMIENTO

El avatar escogido para el protagonista ha sido un caza, diseñado específicamente para este juego. Una vez terminado el boceto, para introducirlo en el programa se ha usado la función **“dibuja_avion”**.



En ella se dibuja el avión con sus respectivos detalles y colores en la posición elegida. Se ha dibujado **línea por línea** verticalmente usando un bucle for. Es importante destacar que se ha ido creando por partes separadas por colores, es decir, hay un bucle para pintar el avión *(coloravion)*, otro para la ventana *(colorventana)* y finalmente para las armas *(colorarmas)*.

Su movimiento horizontal viene codificado dentro del estado “juego”. Después de indicar la lectura del eje x del joystick, aparece la forma en la que se le ha dado velocidad de movimiento al caza. Cuando el joystick se mueve, significa que el avión debe desplazarse 3 píxeles a la derecha o a la izquierda.

Sin embargo, este desplazamiento provoca un duplicado del avión indeseado en la pantalla. Para que esto no suceda, cada vez que el caza se mueve se pinta por completo del color del fondo haciendo uso también de la función “dibuja_avión” la posición anterior del avión y, a continuación, se pinta nuevamente en la posición en la que debe encontrarse.

3.2 PASO POR LAS PANTALLAS

El siguiente paso fue darle al juego una apariencia adecuada con las pantallas por las que va pasando el jugador en el transcurso del juego. Para realizar esto se ha hecho una máquina de estados llamada **“estados”** y se va pasando de uno a otro según las indicaciones que aparezcan por pantalla.

```
enum pantallas{inicio,mapas,preprejuego,prejuego,juego,premuerte,muerte};
```

3.2.1 INICIO

Esta es la pantalla que aparece justo al empezar el juego, **“inicio”**, y su única función es mostrar el título del juego en el centro de la pantalla con unos bordes grises haciendo de marco en los bordes. Además, aparece por pantalla gracias al puerto serie una frase de bienvenida con los controles y las indicaciones para seleccionar el mapa.

```
UARTprintCR("  - Press 'm' for MUTE");
UARTprintCR("  - Press 's' to turn on SOUND");
UARTprintCR("  - Press 'r' to RESET highscore");
UARTprintCR("  - Press 'h' to see your HIGHSCORE");
UARTprintCR("  - Press 'space' to open the MAP
MENU");
```

3.2.2 MAPAS

Se debe pulsar la tecla correspondiente al paisaje deseado y en el estado **“mapas”**, además de aparecer la configuración de cada uno, comenzará una cuenta atrás por pantalla para que el jugador pueda prepararse para comenzar la partida. Esto viene acompañado de una secuencia de sonidos y colores en el led RGB.
3… 2… 1… GO!

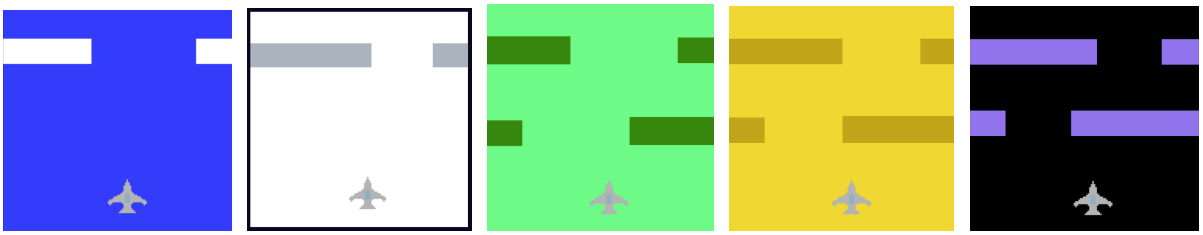
```
UARTprintCR("  - Press '1' for OCEAN");
UARTprintCR("  - Press '2' for SNOW ");
UARTprintCR("  - Press '3' for FOREST");
UARTprintCR("  - Press '4' for DESERT");
UARTprintCR("  - Press '5' for SPACE");
```

3.2.3 JUEGO

Antes de empezar, por motivos de optimización de código, se pasa a un estado previo llamado **“prejuego”**, donde se establece el color de fondo y la dificultad según el nivel que se haya elegido. Una vez realizado esto, comienza la partida pasando al estado **“juego”**.

En el estado **“juego”**, se desarrolla la partida mientras el jugador va moviendo de derecha a izquierda el avión y va acumulando puntos mientras supera con éxito los obstáculos. En este estado se pintan los muros con huecos de forma aleatoria para que no haya una secuencia de la situación de los huecos. Este proceso junto con la dificultad de cada nivel se explicará en uno de los apartados posteriores con mayor detenimiento.

Cuando el jugador no es capaz de pasar por uno de los huecos y por lo tanto el avión se choca con uno de los muros, la partida se para, indicando que ha llegado a su fin y se pasa al estado de **“premuerte”**.



3.2.4 MUERTE

Tristemente si ha aparecido esta pantalla significa que el jugador ha perdido la partida, apareciendo “Game over”. Antes de que finalice el juego, hay un estado previo en el que se actualizan todas las variables necesarias para iniciar una nueva partida de forma correcta. El jugador puede volver a iniciar una partida con el mismo nivel de dificultad dándole a “Restart”, y también puede cambiar de nivel seleccionando o volver a la pantalla de configuración con las indicaciones que aparecen en pantalla en el estado **“muerte”**.

Además de todo esto, es únicamente en el momento en el que se pierde cuando el jugador puede saber qué puntuación ha obtenido en la partida. Por puerto serie se manda esta información al monitor, y se mostrará el número de obstáculos superados con *“Score”*.

3.3 DESARROLLO DE LA PARTIDA

Como se ha mencionado anteriormente, todos los niveles siguen la misma filosofía: el avión debe pasar por los huecos y no chocarse. Sin embargo, lo que diferencia un mapa de otro es la dificultad, ya que se va modificando la anchura de los huecos por los que el avión debe pasar y el número de muros que aparecen por pantalla, quedando de forma resumida de la siguiente manera:

NIVELES	OCÉANO	NIEVE	BOSQUE	DESIERTO	ESPACIO
Muros por pantalla	1	1	2	2	2
Tamaño del hueco	Grande	Normal	Grande	Normal	Pequeño
Color del fondo	Azul oscuro	Blanco	Verde claro	Marrón claro	Negro
Color de los muros	Blanco	Gris	Verde oscuro	Marrón oscuro	Morado

3.3.1 DIBUJO DE MUROS Y HUECOS

Para mostrar los obstáculos, primero fue necesario pintar los muros por completo y realizar su movimiento avanzando por la pantalla. Todo este proceso aparece en el código en el estado **“juego”** de la máquina de estados principal.
Los muros son rectángulos horizontales que van de un lado a otro de la pantalla. Es importante tener en cuenta que en cada nivel los muros son de distintos colores, más o menos adecuados al color de fondo que hay en el mapa. En cada opción del menú de mapas vienen descritas las especificaciones de cada nivel según la tabla mostrada arriba.

Por lo tanto, para que parezca que los muros van descendiendo por la pantalla, se ha pintado del color del fondo las dos líneas superiores del muro, y se redibuja el muro esta

vez en su nueva posición. Esto se ha realizado con un *bucle for* y una variable llamada *“posbarra”.*

Para delimitar los huecos del muro se han usado dos variables, *“limizq”* y *“limder”*, que se generan de forma aleatoria al inicio de la partida y cada vez que el muro llega al final de la pantalla con la variable *“t”*. De esta forma, nunca sabremos dónde estará el hueco en el próximo muro.

Cada vez que la posición del muro alcanza el valor 128 significa que ha llegado al final de la pantalla y que debe empezar a salir otra por la parte superior de la misma, por lo que se actualiza su posición así como los límites, además de incrementar en 1 la puntuación.

3.3.2 COLISIÓN DEL AVIÓN CON LOS MUROS

Siguiendo en el mismo estado del juego, se realiza la colisión cuando el avión no pasa por el hueco.

Para ello, teniendo en cuenta la posición de la barra respecto del avión, de los límites del hueco y las dimensiones del avión, si la posición y de la barra coincide con la altura a la que está situado el avión y, además, la posición horizontal del avión sobrepasa los límites del hueco, se entiende que ha habido un choque y por tanto, la partida finaliza.

3.3.3 NÚMERO DE MUROS POR PANTALLA

Tal y como se muestra en la tabla superior, se puede apreciar que en todos los niveles no aparece el mismo número de obstáculos para que haya diferentes niveles de dificultad.

En la configuración de cada uno de los paisajes, además de especificar el color del fondo, de los muros, y del ancho de los huecos, aparece la variable **“nivel2”**. En los niveles en los que se iguala a 1, significa que la dificultad será mayor y que, por tanto, aparecerán dos muros por pantalla en vez de uno.

Para conseguirlo, en el estado del juego hay un apartado que se realizará solamente cuando el valor de nivel2 sea 1. En ese caso, cuando la primera barra ha superado la mitad de la pantalla, es cuando comenzará a aparecer la segunda por la parte superior, pintándose de forma análoga a la primera barra pero usando las variables *“posbarra2”, “limizq2”, “limder2” y “t2”*. Las colisiones y la puntuación se hacen del mismo modo con ambas barras.

4. RECURSOS UTILIZADOS

RECURSO	Joystick-x	UART-RX	UART-TX	LED-R	LED-G	LED-B	BUZZER
PIN	P1.0	(2)P1.1	(2)P1.2	(2)P2.1	P2.2	P2.4	P2.6

4.1 JOYSTICK

En el desarrollo del juego, que es el único momento en el que se hace uso del joystick, como el avión solamente se mueve en su **eje horizontal** para pasar por los huecos, solamente hará falta configurar el pin P1.0.

4.2 PUERTO SERIE

Por motivos de **optimización de la memoria**, se ha decidido realizar el menú del juego usando el puerto serie, para recibir por pantalla las instrucciones necesarias para jugar.

Para ello, se han usado las funciones secundarias de los botones 1 y 2 de la placa *(BIT1 y BIT2)*, permitiendo así recibir y transmitir mensajes. Se puede avanzar por las diferentes opciones que presenta el juego presionando las teclas que se especifiquen por pantalla, como para la configuración, la selección del mapa y para lo que el jugador quiera hacer una vez finalice su partida.

4.3 LED RGB

Para mayor detalle **visual**, se ha decidido hacer uso del led RGB *(BIT1, BIT2 y BIT4)*, que comenzará apagado.

Mientras se ve la *pantalla de inicio*, esta luz se encenderá de color azul. Después de esto, no se volverá a encender hasta que no llegue la *cuenta atrás* del comienzo del juego, donde habrá una secuencia de colores. Finalmente, cuando el avión se choque con un muro y se *pare el juego*, la luz se encenderá de color rojo, dando a entender que la partida ha llegado a su fin.

4.4 PIN PWM: BUZZER

El pin P2.6 tiene dos opciones de uso. La primera de ellas es el buzzer, y es la que se ha decidido implementar en el trabajo.

5. SONIDOS DEL JUEGO

Llegados a este punto del proyecto, al tener finalizado con éxito el juego básico, se ha decidido añadir pequeños detalles para que sea más realista y similar a un juego de verdad. Para poder introducir sonidos con éxito ha sido necesario investigar el ejemplo 6 de programación subido a Enseñanza Virtual por el profesorado de la asignatura.

Se han añadido varios sonidos que aparecen en diferentes momentos del juego, a modo de una pequeña banda sonora. Se han creado dos **melodías**, una para el *inicio* del juego, que sonará mientras se muestra la portada, y otra cuando se llega al *game over*. Además, también se escuchará una secuencia de notas en la *cuenta atrás* del inicio de la partida.

Para realizarlo, se ha usado la función **"toca_nota"** y se ha configurado el **PWM** con el **timer 1**. En las partes del código necesarias, se hace llamada a la función y se especifica la nota que debe sonar y su duración haciendo uso de la variable "tmus"

Además, aunque siempre se escuchará la melodía de inicio, en las diferentes opciones que aparecen en el menú posterior está la posibilidad de *silenciar el juego*. Cuando la variable "mute" tiene valor 1, significa que el buzzer estará desactivado, por lo que no se escuchará nada. Sin embargo, si tiene valor 0, se sabrá que el sonido está activado.

```
if(tmus<20)
      toca_nota(SI,mute);
else if(tmus<30)
      toca_nota(1,mute);
```

6. DETALLES DE LA PUNTUACIÓN

En ese mismo menú en el que se puede configurar el volumen del juego, también se puede ver la puntuación más alta que se ha obtenido en las partidas que se han jugado, almacenándose en la variable **"highscore"**.

Cada vez que se juega una partida nueva, la puntuación obtenida se compara con el valor del highscore que haya en ese momento. Si el valor obtenido es menor, simplemente aparecerá por pantalla, pero si el jugador se ha superado a sí mismo y obtiene más puntos, además de aparecer por pantalla, el valor del *highscore se actualizará* y por pantalla se mostrará un *mensaje de felicitación*.

6.1 ALMACENAMIENTO EN MEMORIA DEL HIGHSCORE

Como último reto, se ha intentado que además de registrarse un highscore cada vez que se juega, es decir, cada vez que se ejecuta el programa, que se asemeje más a un videojuego real haciendo que esta puntuación se **guarde** de forma **permanente** en la *memoria flash* de la placa.

Para ello, se ha usado la función **“guarda_flash_seg”**, proporcionada en uno de los ejemplos de programación subidos en la plataforma. Se han hecho algunas modificaciones para adaptarla adecuadamente al código.

Cuando conseguimos un nuevo highscore, este se almacena en la dirección de memoria *0x1000* y permanecerá intacta hasta que en cualquier momento, se obtenga una puntuación mayor. Además, se ha introducido la opción de resetearlo por si el jugador quiere comenzar desde cero, como se ha indicado anteriormente en el menú de configuración.

```
highscore=*Puntero3;
if(puntuacion>highscore){
      highscore=puntuacion;
      guarda_flash_seg(highscore,0);
      UARTprintCR("Congratulations! You have a new highscore.");
 }
```

7. ALCANCE DEL JUEGO

Como se ha repetido en muchos apartados del documento, la **memoria** de la placa ha sido la mayor limitación y dificultad encontrada en el desarrollo del trabajo.

Aún así, el juego es bastante atractivo y **fácil** de jugar. La intención desde un primer momento fue hacer un videojuego adaptado a las condiciones del material de trabajo, por lo que el objetivo planteado fue hacer un juego sencillo, visualmente atractivo y con capacidad de generar una pequeña **adicción** en el jugador, requisitos necesarios para que cualquier juego sea exitoso.

Este tipo de “minijuegos” forman parte de un tipo de entretenimiento que muchos jóvenes y adultos consumen en el día a día para rellenar momentos muertos, por lo que si este proyecto tuviera un alcance real y fuera publicado de verdad, su **éxito** sería claro.

8. CONCLUSIONES

Finalmente, con este trabajo se ha puesto a prueba el conocimiento real que posee el alumnado sobre la programación en lenguaje C y el manejo de todas las posibilidades que proporciona el microcontrolador y el Boosterpack. Además, se ha sabido optimizar el código y los recursos utilizados, para hacer un trabajo bastante completo sin superar el límite de la capacidad de la memoria.

Por destacar algún contratiempo en el proceso de creación, hubo dificultades en la adición de una segunda barra en los niveles más complicados. También en un primer momento, las partidas se desarrollaban de forma excesivamente lenta, además de tardar bastante en conseguir que el highscore se guardará de forma adecuada en la memoria flash de la placa.

Sin embargo, afortunadamente todos los errores fueron solucionados sin mucho problema para que el juego quedara tal y como está, perfecto.

9. CÓDIGO COMPLETO

```
#include <msp430.h>
#include "grlib.h"
#include "uart_STDIO.h"
#include "Crystalfontz128x128_ST7735.h"
#include "HAL_MSP430G2_Crystalfontz128x128_ST7735.h"
#include <stdio.h>
#include "partituras.h"

void conf_reloj(char VEL){
    BCSCTL2 = SELM_0 | DIVM_0 | DIVS_0;
    switch(VEL){
    case 16:
        if (CALBC1_16MHZ != 0xFF) {
            __delay_cycles(100000);
            DCOCTL = 0x00;
            BCSCTL1 = CALBC1_16MHZ;     /* Set DCO to 16MHz */
            DCOCTL = CALDCO_16MHZ;
        }
        break;
    }
    BCSCTL1 |= XT2OFF | DIVA_0;
    BCSCTL3 = XT2S_0 | LFXT1S_2 | XCAP_1;

}

void inicia_ADC(char canales){
    ADC10CTL0 &= ~ENC;//deshabilita ADC
    ADC10CTL0 = ADC10ON | ADC10SHT_3 | SREF_0|ADC10IE;
    //enciende ADC, S/H 64clk's, REF:VCC-GND, con INT
    ADC10CTL1 = CONSEQ_0 | ADC10SSEL_0 | ADC10DIV_0 | SHS_0 | INCH_0;
    //Modo simple, reloj ADC, sin subdivision, Disparo soft, Canal 0
    ADC10AE0 = canales; //habilita los canales indicados
    ADC10CTL0 |= ENC; //Habilita el ADC
}

int lee_ch(char canal){
    ADC10CTL0 &= ~ENC;//deshabilita el ADC
    ADC10CTL1&=(0x0fff);//Borra canal anterior
    ADC10CTL1|=canal<<12;//selecciona nuevo canal
    ADC10CTL0|= ENC;//Habilita el ADC
    ADC10CTL0|=ADC10SC;//Empieza la conversión
    LPM0;//Espera fin en modo LPM0
    return(ADC10MEM);//Devuelve valor leido
}


// DECLARACIONES
Graphics_Context g_sContext;
char *Puntero3= (char *)(0x1000); // Puntero que apunta a la direccion de memoria
donde guardamos el highscore
char cadena[20];
int
avionx=26,posx1=26,avionxant=26,posbarra=0,posbarra2=-10,ampliacion=0,nivel2=0,segu
nda=0;// Posicion del avion y de las barras, y parámetros de los niveles
char mute=0; // Variable para mutear el juego. De serie comienza con sonido
int puntuacion=0,highscore=0; // Puntuaciones
volatile char caracter=0;  // EL caracter que recibimos por puerto serie
```

```
int ejex,t=0,tms=0,t2=0,tmus=0; // Posicion del joystick y diferentes
temporizadores
char limizq,limder,limizq2,limder2; // Límites de las barras
long int
coloravion=GRAPHICS_COLOR_GRAY,colormuro,colorfondo,colorventana=0x0093D6FA,colorar
mas=GRAPHICS_COLOR_BLACK; // Colores que modificaremos en funcion del mapa
unsigned char estado=0,i; // Maquina de estados y la i para los bucles for
enum
pantallas{inicio,opciones,premapas,mapas,juego,premuerte,muerte,preprejuego,prejueg
o,prejuego2,prejuego3}; // Estados de la maquina de estados



int main(void) {

    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer
    conf_reloj(16); // Configuración del reloj

    FCTL2 = FWKEY + FSSEL_2 + 34; // Habilitamos el guardado en la flash

    // Timer 100 ms
    TA1CTL=TASSEL_2|ID_3| MC_1;         //SMCLK, DIV=8 (2MHz), UP
    TA1CCR0=19999;       //periodo=20000: 10ms
    TA1CCTL0=CCIE;      //CCIE=1


    TA0CTL=TASSEL_2| MC_1;  //SMCLK, DIV=1, UP
    TA0CCTL1=OUTMOD_7;      //OUTMOD=7

    inicia_ADC(BIT0); // Inicializamos el joystick solamente en el eje x



    //Pines del buzzer
    P2DIR|=BIT6;            //P2.6 salida
    P2SEL|= BIT6;           //P2.6 pwm
    P2SEL2&=~(BIT6+BIT7);   //P2.7 gpio
    P2SEL&=~(BIT7);         //P2.7 gpio

    // Pines y configuración para el puerto serie
    P1SEL2 = BIT1 | BIT2; //P1.1 RX, P1.2: TX
    P1SEL = BIT1 | BIT2;
    P1DIR = BIT0+ BIT2;

    /*------ Configuración de la USCI-A para modo UART:----------*/
    UCA0CTL1 |= UCSWRST; // Reset
    UCA0CTL1 = UCSSEL_2 | UCSWRST; //UCSSEL_2: SMCLK (16MHz) | Reset on
    UCA0CTL0 = 0; // 8bit, 1stop, sin paridad. NO NECESARIO
    UCA0BR0 = 139; // 16MHz/139=115108...
    UCA0CTL1 &= ~UCSWRST; //Quita reset
    IFG2 &= ~(UCA0RXIFG); //Quita flag
    IE2 |= UCA0RXIE; // y habilita int. recepcion

    // Pines del led RGB
    P2DIR|=BIT1|BIT4|BIT2; //Añadimos las direcciones del led RGB que vamos a usar
    P2OUT&=~(BIT1|BIT4|BIT2); // Inicializamos el led RGB apagado

    __bis_SR_register(GIE); // Habilita las interrupciones globalmente
```

```
    // Inicializamos Graphics Context
    Crystalfontz128x128_Init();
    Crystalfontz128x128_SetOrientation(LCD_ORIENTATION_UP); // Dejamos la
orientacion por defecto
    Graphics_initContext(&g_sContext, &g_sCrystalfontz128x128);

    Graphics_clearDisplay(&g_sContext); // Limpiamos la pantalla por lo que pudiera
haber de antes
    Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);// Ponemos el
fondo en negro

    Graphics_setFont(&g_sContext, &g_sFontCm16b); // Usamos este tamaño de fuente

    // Dibujamos la imagen de la portada

    for(i=0;i<6;i++){ // Bucle for para dibujar un marco rectangular de ancho 6
        Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GRAY); // El marco
será de color gris
        Graphics_Rectangle rectangulo1 = {1+i,1+i,127-i,127-i}; // Rectángulo que
abarcará toda la pantalla e ira disminuyendo su tamaño con el bucle for
        Graphics_drawRectangle(&g_sContext,&rectangulo1); // Dibujamos solo los
bordes del rectángulo, ya que si lo llenaramos taparía toda la pantalla
    }
    // Letras del título del juego
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_LIME_GREEN); // Color
para "Imminent"
    Graphics_drawString(&g_sContext,"Imminent", 12, 27, 50, TRANSPARENT_TEXT);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED); // Color para
"CRASH"
    Graphics_drawString(&g_sContext,"CRASH", 5, 45, 65, TRANSPARENT_TEXT);
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_YELLOW); // Color para
"!"
    Graphics_drawString(&g_sContext,"!", 1, 97, 65, TRANSPARENT_TEXT);


    // Mensaje que se mandará por el puerto serie al inicio del juego
    UARTprintCR("WELCOME TO IMMINENT CRASH! Try to dodge the walls moving your
plane left and right with the joystick.");
    UARTprintCR("Press 'enter' to continue");


    while(1){
        LPM0; // Entramos en modo de bajo consumo


        switch(estado)
        {
        case inicio: // Estado inicial del juego

            P2OUT|=BIT4; // Encendemos el led de color azul

            // Toca melodía de inicio del juego
            if(tmus<20)
                toca_nota(SI,mute); // Función para tocar una nota, si mute vale 1
no se tocará.
            else if(tmus<30)
                toca_nota(1,mute);
```

```
            else if(tmus<50)
                toca_nota(SOL,mute);
            else if(tmus<60)
                toca_nota(1,mute);
            else if(tmus<90)
                toca_nota(RE,mute);
            else if(tmus<120)
                toca_nota(1,mute);
            else if(tmus<140)
                toca_nota(SI,mute);
            else if(tmus<150)
                toca_nota(1,mute);
            else if(tmus<170)
                toca_nota(SOL,mute);
            else if(tmus<180)
                toca_nota(1,mute);
            else if(tmus<200)
                toca_nota(RE,mute);
            else {
                toca_nota(1);

                if(caracter==13) // Si el caracter recibido por puerto serie es el
enter
                {
                    tmus=0; // Ponemos el temporizador de la musica a 0

                    UARTprintCR("MENU");
                    UARTprintCR("  - Press 'm' for MUTE");
                    UARTprintCR("  - Press 's' to turn on SOUND ");
                    UARTprintCR("  - Press 'r' to RESET highscore ");
                    UARTprintCR("  - Press 'h' to see your HIGHSCORE ");
                    UARTprintCR("  - Press 'space' to open the MAP MENU ");
                    estado=opciones;

                }

            }
            break;

        case opciones: // Menu del juego, configuraciones...


            if(caracter=='m'){ // Si el caracter recibido es "m":
                mute=1; // Ponemos mute a 1, con lo que se silenciará toda la
música y sonidos del juego

                UARTprintCR("Now the game is muted"); // Mensaje por puerto serie
para saber que se ha silenciado

                caracter=0; // Asignamos el valor 0 a caracter para no entrar en un
bucle


            }
            if(caracter=='s'){ // Si el caracter recibido es "s":
                mute=0; // Ponemos mute a 0, con lo que toda la musica y sonidos
del juego sonarán de forma normal
```

```
                UARTprintCR("Now the game is unmuted"); // Mensaje por puerto serie
para saber que se ha activado el sonido

                caracter=0; // Asignamos el valor 0 a caracter para no entrar en un
bucle

            }

            if(caracter=='r'){ // Si el caracter recibido es "r":
                highscore=0;  // Ponemos la variable highscore a 0

                guarda_flash_seg(highscore, 0); // La guardamos en nuestra
dirección de memoria asignada (0x1000)

                UARTprintCR("Now the highscore is set to 0"); // Mensaje por puerto
serie para saber que se ha reseteado el highscore

                caracter=0;  // Asignamos el valor 0 a caracter para no entrar en
un bucle

            }
            if(caracter=='h'){ // Si el caracter recibido es "h":
                highscore=*Puntero3; // Asignamos a la variable highscore el valor
almacenado en memoria

                sprintf(cadena,"Your highscore is: %d",highscore); // Escribimos la
frase con nuestro highscore en el vector cadena

                UARTprintCR(cadena); // Mensaje por puerto serie que nos indica
nuestro highscore actual

                caracter=0;  // Asignamos el valor 0 a caracter para no entrar en
un bucle


            }
            if(caracter==' '){ // Si el caracter recibido es el espacio:

                estado=premapas; // Pasamos al  estado previo al menú de selección
de mapas
            }
            break;
        case premapas: // Estado previo al menú de selección de mapas

            UARTprintCR("CHOOSE YOUR MAP:"); // Se manda por puerto serie las
opciones de mapas existentes
            UARTprintCR("  - Press '1' for OCEAN");
            UARTprintCR("  - Press '2' for SNOW ");
            UARTprintCR("  - Press '3' for FOREST");
            UARTprintCR("  - Press '4' for DESERT");
            UARTprintCR("  - Press '5' for SPACE");

            estado=mapas; // Pasamos a la selección de mapas
            break;

        case mapas: // Estado de selección de mapas

            if(caracter=='1') // Si el mapa seleccionado es el océano
            {
```

```
                colorfondo=0x00277CFF; // Se asigna el color del fondo, azul
                colormuro=GRAPHICS_COLOR_WHITE; // Se asigna el color de los muros,
blanco
                ampliacion=20; // Se asigna el ancho del hueco, en este caso 20 ya
que es el nivel más fácil
                nivel2=0; // En este nivel sólo hay 1 barra por pantalla

                estado=preprejuego; // Pasamos al siguiente estado para comenzar la
partida
            }

            if(caracter=='2') // Si el mapa seleccionado es la nieve
            {
                colorfondo=GRAPHICS_COLOR_SNOW; // Se asigna el color del fondo,
blanco nieve
                colormuro=GRAPHICS_COLOR_DARK_GRAY; // Se asigna el color de los
muros, gris
                ampliacion=10; // Se asigna el ancho del hueco, en este caso 10 ya
que el nivel es fácil
                nivel2=0; // En este nivel sólo hay 1 barra por pantalla

                estado=preprejuego; // Pasamos al siguiente estado para comenzar la
partida
            }

            if(caracter=='3') // Si el mapa seleccionado es el bosque
            {
                colorfondo=GRAPHICS_COLOR_LIME_GREEN; // Se asigna el color del
fondo, verde claro
                colormuro=GRAPHICS_COLOR_FOREST_GREEN; // Se asigna el color de los
muros, verde oscuro
                ampliacion=14; // Se asigna el ancho del hueco, en este caso 14 ya
que es el nivel más fácil con 2 muros
                nivel2=1;  // En este nivel hay 2 barras por pantalla

                estado=preprejuego; // Pasamos al siguiente estado para comenzar la
partida
            }

            if(caracter=='4') // Si el mapa seleccionado es el desierto
            {
                colorfondo=GRAPHICS_COLOR_GOLD; // Se asigna el color del fondo,
dorado claro
                colormuro=GRAPHICS_COLOR_GOLDENRON; // Se asigna el color de los
muros, dorado oscuro
                ampliacion=10; // Se asigna el ancho del hueco, en este caso 10 ya
que incrementa la dificultad habiendo 2 muros
                nivel2=1;  // En este nivel hay 2 barras por pantalla

                estado=preprejuego; // Pasamos al siguiente estado para comenzar la
partida
            }

            if(caracter=='5') // Si el mapa seleccionado es el espacio
            {
                colorfondo=GRAPHICS_COLOR_BLACK; // Se asigna el color del fondo,
negro
                colormuro=0x00EA94FF; // Se asigna el color de los muros, morado
```

```
                ampliacion=4; // Se asigna el ancho del hueco, en este caso 4 ya
que es el nivel más dificil
                nivel2=1;  // En este nivel hay 2 barras por pantalla

                estado=preprejuego; // Pasamos al siguiente estado para comenzar la
partida

            }

            break;
        case preprejuego: // Estado realizado para optimizar código
            tmus=0; // Ponemos el temporizador de la musica a 0
            UARTprintCR("3..."); // Mensaje mostrado por puerto serie para saber
que inicia la cuenta atrás
            estado=prejuego; // Pasamos al siguiente estado
            break;

        case prejuego:

            P2OUT&=~(BIT4|BIT2); // Se apagan los leds azul y verde
            P2OUT|=(BIT1); // Se enciende el led en rojo, para empezar la cuenta
atrás

            toca_nota(MI>>1,mute); // Primera nota de la cuenta atrás

            if(tmus>=100){
                toca_nota(1,mute); // Añadimos un silencio
                UARTprintCR("2..."); // Mensaje mostrado por puerto serie de la
cuenta atrás
                tmus=0; // Ponemos el temporizador de la musica a 0
                // Inicializamos el avión en el centro de la pantalla
                Graphics_setBackgroundColor(&g_sContext, colorfondo); // Ponemos el
color de fondo correspondiente
                Graphics_clearDisplay(&g_sContext); // Borramos lo que hubiera
antes
                dibuja_avion(avionx,54,coloravion,colorventana,colorarmas); //
Dibujamos el avión
                // Inicializamos los limites de los muros
                limizq = t;
                limder = t+40+ampliacion;
                limizq2 = t2;
                limder2 = t2+40+ampliacion;
                estado=prejuego2; // Pasamos al siguiente estado
            }
            break;

        case prejuego2:


            toca_nota(1,mute); // Seguimos en silencio

            if(tmus>=100){
                toca_nota(MI>>1,mute); // Segunda nota de la cuenta atrás
                P2OUT|=(BIT1|BIT2); // Se enciende el led en amarillo, continuando
la cuenta atrás
                UARTprintCR("1..."); // Mensaje mostrado por puerto serie de la
cuenta atrás
                tmus=0; // Ponemos el temporizador de la musica a 0
                estado=prejuego3; // Pasamos al siguiente estado
```

```
            }
            break;

        case prejuego3:

            if(tmus>88)
                toca_nota(1,mute); // Tocamos un silencio

            if(tmus>=190){
                toca_nota(MI>>2,mute); // Última nota de la cuenta atrás
                P2OUT&=~(BIT4|BIT1); // Se apagan los leds azul y rojo
                P2OUT|=(BIT2); // Se enciende el led en verde, finalizando la
cuenta atrás
                tmus=0;  // Ponemos el temporizador de la musica a 0
                UARTprintCR("GO!"); // Mensaje mostrado por puerto serie indicando
que se ha finalizado la cuenta atrás

                estado=juego; // Pasamos al estado principal del juego
            }
            break;


        case juego: // Estado donde se desarrolla el juego
            if(tmus>50){ // Despues de poco tiempo

                P2OUT&=~(BIT4|BIT2|BIT1); // Apagamos el led
            }
            if(tmus>20){
                toca_nota(1,mute); // Deja de sonar música

            }


            ejex=lee_ch(0); // Asignamos a la variable ejex el valor del eje x del
joystick, que ira de 0 a 1024

            posbarra+=2; // Hacemos que el muro avance

            for(i=0;i<10;i++){ // Bucle for para dibujar el muro
                Graphics_setForegroundColor(&g_sContext, colormuro); // Lo pintamos
del color correspondiente
                Graphics_drawLine(&g_sContext,0,0+i+posbarra,limizq,0+i+posbarra);
                // Línea horizontal que va desde el inicio de la pantalla hasta el
limite izquierdo del hueco

Graphics_drawLine(&g_sContext,limder,0+i+posbarra,128,0+i+posbarra);
                // Línea horizontal que va desde el limite derecho del hueco hasta
el final de la pantalla
            }

            Graphics_setForegroundColor(&g_sContext, colorfondo); // Borramos los
dos pixeles anteriores del muro
            Graphics_drawLine(&g_sContext,0,posbarra-1,limizq,posbarra-1);
            Graphics_drawLine(&g_sContext,limder,posbarra-1,128,posbarra-1);
            Graphics_drawLine(&g_sContext,0,posbarra-2,limizq,posbarra-2);
            Graphics_drawLine(&g_sContext,limder,posbarra-2,128,posbarra-2);


            if(posbarra>128){ // Cuando la barra llegue al final de la pantalla
```

```
                posbarra=-10; // Se resetea la posición para que aparezca otra vez
desde el inicio de la pantalla
                // Se toman nuevos límites para el muro

                limizq=t;
                limder=t+40+ampliacion;
                t=t+5;
                puntuacion+=1; // Se incrementa en 1 la puntuación
            }

            if(posbarra>54 && nivel2==1){ // Si la primera barra pas ala mitad de
la pantalla, y es un nivel con dos barras:
                segunda=1; // Enable para que la segunda barra funcione
            }
            if(segunda==1){ // Si esta variable vale 1, se permite el movimiento de
la segunda barra
                posbarra2+=2; // Avanza la segunda barra dos pixeles
                for(i=0;i<10;i++){ // Se dibuja la segunda barra de la misma forma
que la primera
                    Graphics_setForegroundColor(&g_sContext, colormuro);

Graphics_drawLine(&g_sContext,0,0+i+posbarra2,limizq2,0+i+posbarra2);

Graphics_drawLine(&g_sContext,limder2,0+i+posbarra2,128,0+i+posbarra2);
                }

                Graphics_setForegroundColor(&g_sContext, colorfondo);
                // Se borran los dos píxeles anteriores
                Graphics_drawLine(&g_sContext,0,posbarra2-1,limizq2,posbarra2-1);
                Graphics_drawLine(&g_sContext,limder2,posbarra2-1,128,posbarra2-1);
                Graphics_drawLine(&g_sContext,0,posbarra2-2,limizq2,posbarra2-2);
                Graphics_drawLine(&g_sContext,limder2,posbarra2-2,128,posbarra2-2);

                // Cuando la segunda barra ha recorrido toda la pantalla, aparece
nuevamente arriba, de la misma forma que la primera
                if(posbarra2>128){
                    posbarra2=-10;
                    limizq2=t2;
                    limder2=t2+40+ampliacion;
                    t2=t2+33;
                    puntuacion+=1;

                }
            }

            // Movimiento del avión con el joystick
            if(ejex>800){
                // Si el joystick se inclina a la derecha más de 800
                posx1=posx1+3; // el avión se desplaza 3 píxeles a la derecha
            }
            if(ejex<=200){
                // Si el joystick se inclina a la izquierda menos de 200
                posx1=posx1-3;  // el avión se desplaza 3 píxeles a la izquierda
            }
            if(posx1<-25){
                // Si la posición del avión es más pequeña que -25, se saldría de
la pantalla
                posx1=-25;
            }
```

```
            if(posx1>75){
                // Si la posición del avión es mayor que 75, se saldría de la
pantalla
                posx1=75;
            }

            avionx=posx1; // Actualizamos la posición del avión

            if(avionx==avionxant){ } // Si el avión no se ha movido, no se redibuja
            else{ // Si el avión se ha movido
                dibuja_avion(avionxant,54,colorfondo,colorfondo,colorfondo); //
Pintamos del color del fondo la posición anterior del avión
                dibuja_avion(avionx,54,coloravion,colorventana,colorarmas); //
Dibujamos el avión en su nueva posición
                avionxant=avionx;
            }
            // Colisiones
            if(posbarra>81 && posbarra<111){ // Si la barra está a la altura del
avión
                if((avionx+28)<limizq || (avionx+48)>limder){ // Y el avión se sale
de los límites del hueco
                    tmus=0; // Reiniciamos el tiempo de la música
                    estado=premuerte; // Se pasa al siguiente estado, ya que el
jugador ha perdido la partida

                }
            }
            if(posbarra2>81 && posbarra2<111){ // Si la barra 2 está a la altura
del avión
                if((avionx+26)<limizq2 || (avionx+50)>limder2){ // Y el avión se
sale de los límites del hueco 2
                    tmus=0; // Reiniciamos el tiempo de la música
                    estado=premuerte; // Se pasa al siguiente estado, ya que el
jugador ha perdido la partida

                }
            }
            break;

        case premuerte: // Estado anterior a la muerte

            P2OUT&=~(BIT4|BIT2); // Apagamos los leds verde y azul
            P2OUT|=(BIT1); // Encendemos el led rojo indicando la muerte
            // Melodía de game over
            if(tmus<20)
                toca_nota(MI,mute);
            else if(tmus<30)
                toca_nota(1,mute);
            else if(tmus<50)
                toca_nota(RE,mute);
            else if(tmus<60)
                toca_nota(1,mute);
            else if(tmus<90)
                toca_nota(DO,mute);
            else if(tmus<100)
                toca_nota(1,mute);
            else{
                pantallamuerte(); // Sale "Game over" en color rojo por pantalla
```

```
                sprintf(cadena,"SCORE: %d",puntuacion); // Se muestra por puerto
serie la puntuación obtenida en la partida jugada
                UARTprintCR(cadena);

                highscore=*Puntero3; // Se lee la dirección de memoria donde está
almacenado el highscore

                if(puntuacion>highscore){ // Si la puntuación obtenida es mayor que
el highscore
                    highscore=puntuacion; // Actualizamos el highscore

                    guarda_flash_seg(highscore,0); // Lo guardamos en el espacio de
memoria asignado

                    UARTprintCR("Congratulations! You have a new highscore."); //
Mensaje por puerto serie de actualización del highscore
                }
                else{}
                // Reset de todas las variables necesarias para iniciar una nueva
partida
                segunda=0;
                avionx=26;
                avionxant=26;
                posx1=26;
                posbarra=-10;
                posbarra2=-10;
                limizq=t;
                limder=t+40+ampliacion;
                limizq2=t2;
                limder2=t2+40+ampliacion;
                caracter=0;
                // Menú final
                UARTprintCR("  - Press 'space' to restart");
                UARTprintCR("  - Press 'm' to open the map menu");
                UARTprintCR("  - Press 'o' to open options");
                puntuacion=0;


                estado=muerte; // Pasamos al último estado
            }
            break;

        case muerte:

            if(caracter==' ') // Si se pulsa el espacio se reiniciará la partida
            {
                P2OUT&=~BIT1; // Se apaga el led rojo
                caracter=0;  // Asignamos el valor 0 a caracter para no entrar en
un bucle
                tmus=0;
                UARTprintCR("3...");
                estado=prejuego; // Pasamos al estado de prejuego e inicia la
cuenta atrás
            }

            if(caracter=='m') // Si se pulsa la letra m volvemos al menú de mapas
            {
                P2OUT&=~BIT1; // Se apaga el led rojo
```

```
                caracter=0; // Asignamos el valor 0 a caracter para no entrar en un
bucle
                nivel2=0; // Reseteamos el nivel
                UARTprintCR("");
                estado=premapas; // Pasamos al menú de mapas
            }
            if(caracter=='o') // Si se pulsa la tecla o volvemos al menú de
opciones
            {
                P2OUT&=~BIT1; // Se apaga el led rojo
                nivel2=0; // Reseteamos el nivel
                UARTprintCR("");
                tmus=200; // Cuando vuelve al estado inicio no toca la melodía,
directamente pasa al estado opciones
                caracter=13; // Pasa directamente sin darle al enter
                estado=inicio; // Volvemos al inicio
            }
            break;

        }
    }
}
// Interrupciones
#pragma vector=ADC10_VECTOR // Interrupción generada por el convertidor ADC que
usamos para leer el joystick
__interrupt void ConvertidorAD(void)
{
    LPM0_EXIT; //Despierta al micro al final de la conversión
}
#pragma vector=TIMER1_A0_VECTOR // Interrupción generada por el timer 1
__interrupt void Interrupcion_T1(void)
{
    tms++;
    t++; // Incremento de la variable para los límites aleatorios en la barra 1
    t2++; // Incremento de la variable para los límites aleatorios en la barra 2
    if(t>88){
        t=0;}
    if(t2>78){
        t2=0;
    }
    if(tms>=1)
    {
        tms=0;
        tmus++; // Incremento de la variable que indica la duración de una nota
musical
        LPM0_EXIT;
    }
}


#pragma vector=USCIAB0RX_VECTOR // Interrupción generada por el puerto serie al
recibir un caracter
__interrupt void USCI0RX_ISR_HOOK(void)
{
    caracter=UCA0RXBUF; // Leo dato recibido
    LPM3_EXIT;
}

// Funciones usadas
```

```
char guarda_flash_seg(char dato, char direc) // Función para guardar en memoria
flash
{
    char *  Puntero = (char *) (0x1000); // Apunta a la direccion
    char buffer_ram[64];
    char i;
    if(direc<64)
    { //Comprueba area permitida
        if(Puntero[direc]!=0xFF)
        {
            //Guarda el bloque en RAM
            for (i=0;i<64;i++) buffer_ram[i]= Puntero[i];
            //Borra el bloque
            FCTL1 = FWKEY + ERASE;       // activa  Erase
            FCTL3 = FWKEY;               // Borra Lock (pone a 0)
            Puntero[0] = 0;             // Escribe algo para borrar el segmento
            //Sobreescribe posicion en RAM
            buffer_ram[direc]=dato;
            //Reescribe la flash
            FCTL1 = FWKEY + WRT;         // Activa WRT y borra ERASE
            for (i=0;i<64;i++) Puntero[i]= buffer_ram[i];
            FCTL1 = FWKEY;               // Borra bit WRT
            FCTL3 = FWKEY + LOCK;        //activa LOCK
            return 1;
        }
        else
        {
            //Escribe la flash (solo ese dato)
            FCTL3 = FWKEY;              // Borra Lock (pone a 0)
            FCTL1 = FWKEY + WRT;        // Activa WRT
            Puntero[direc]=dato;        //Escribo el dato
            FCTL1 = FWKEY;              // Borra bit WRT
            FCTL3 = FWKEY + LOCK;       //activa LOCK
            return 2;
        }
    }
    return 0;
}

void toca_nota(unsigned int nota,char mute) // Función para tocar una nota musical
{
    if(mute==1){ // Si mute es 1, toca un silencio
        TA0CCR0=1;
        TA0CCR1=1>>1;
    }
    else{ // De otra forma, toca la nota asignada
        TA0CCR0=nota;
        TA0CCR1=nota>>1;
    }
}



int pantallamuerte(){ // Función para la pantalla del game over
    Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
    Graphics_drawString(&g_sContext,"GAME OVER", 9, 17, 55, TRANSPARENT_TEXT);
    return 0;

}
```

```
int dibuja_avion(int posx,int posy,long int coloravion,long int colorventana,long
int colorarmas){
    // Función para dibujar y borrar el avión línea a línea
    int i;


    for(i=0;i<2;i++){ // Armas
        Graphics_setForegroundColor(&g_sContext, colorarmas);
        Graphics_drawLine(&g_sContext,31-i+posx,50+posy,31-i+posx,44+posy);
        Graphics_drawLine(&g_sContext,45+i+posx,50+posy,45+i+posx,44+posy);
    }


    Graphics_setForegroundColor(&g_sContext, coloravion);
    for(i=0;i<2;i++){  // Cuerpo del avión (se dibuja línea a línea simétricamente
en base a un centro)
        Graphics_drawLine(&g_sContext,26+posx,50+posy,26+posx,49+posy);
        Graphics_drawLine(&g_sContext,26+posx,50+posy,26+posx,49+posy);
        Graphics_drawLine(&g_sContext,27+posx,51+posy,27+posx,48+posy);
        Graphics_drawLine(&g_sContext,28+posx,50+posy,28+posx,48+posy);
        Graphics_drawLine(&g_sContext,29+posx,49+posy,29+posx,47+posy);
        Graphics_drawLine(&g_sContext,31-i+posx,49+posy,31-i+posx,46+posy);
        Graphics_drawLine(&g_sContext,33-i+posx,49+posy,33-i+posx,45+posy);
        Graphics_drawLine(&g_sContext,34+posx,49+posy,34+posx,44+posy);
        Graphics_drawLine(&g_sContext,35+posx,50+posy,35+posx,43+posy);
        Graphics_drawLine(&g_sContext,36+posx,56+posy,36+posx,39+posy);
        Graphics_drawLine(&g_sContext,37+posx,56+posy,37+posx,37+posy);
        Graphics_drawLine(&g_sContext,38+posx,56+posy,38+posx,35+posy);
        Graphics_drawLine(&g_sContext,39+posx,56+posy,39+posx,37+posy);
        Graphics_drawLine(&g_sContext,40+posx,56+posy,40+posx,39+posy);
        Graphics_drawLine(&g_sContext,41+posx,50+posy,41+posx,43+posy);
        Graphics_drawLine(&g_sContext,42+posx,49+posy,42+posx,44+posy);
        Graphics_drawLine(&g_sContext,43+posx+i,49+posy,43+i+posx,45+posy);
        Graphics_drawLine(&g_sContext,45+i+posx,49+posy,45+i+posx,46+posy);
        Graphics_drawLine(&g_sContext,47+posx,49+posy,47+posx,47+posy);
        Graphics_drawLine(&g_sContext,48+posx,50+posy,48+posx,48+posy);
        Graphics_drawLine(&g_sContext,49+posx,51+posy,49+posx,48+posy);
        Graphics_drawLine(&g_sContext,50+posx,50+posy,50+posx,49+posy);
    }
    for(i=0;i<3;i++){ // Cola del avión
        Graphics_drawLine(&g_sContext,35-i+posx,56+posy,35-i+posx,54+i+posy);
        Graphics_drawLine(&g_sContext,41+i+posx,56+posy,41+i+posx,54+i+posy);
    }
    Graphics_setForegroundColor(&g_sContext, colorventana); // Ventana del avión
    Graphics_drawLine(&g_sContext,37+posx,49+posy,37+posx,45+posy);
    Graphics_drawLine(&g_sContext,38+posx,49+posy,38+posx,44+posy);
    Graphics_drawLine(&g_sContext,39+posx,49+posy,39+posx,45+posy);

    return 0;

    // fin :)

}
```