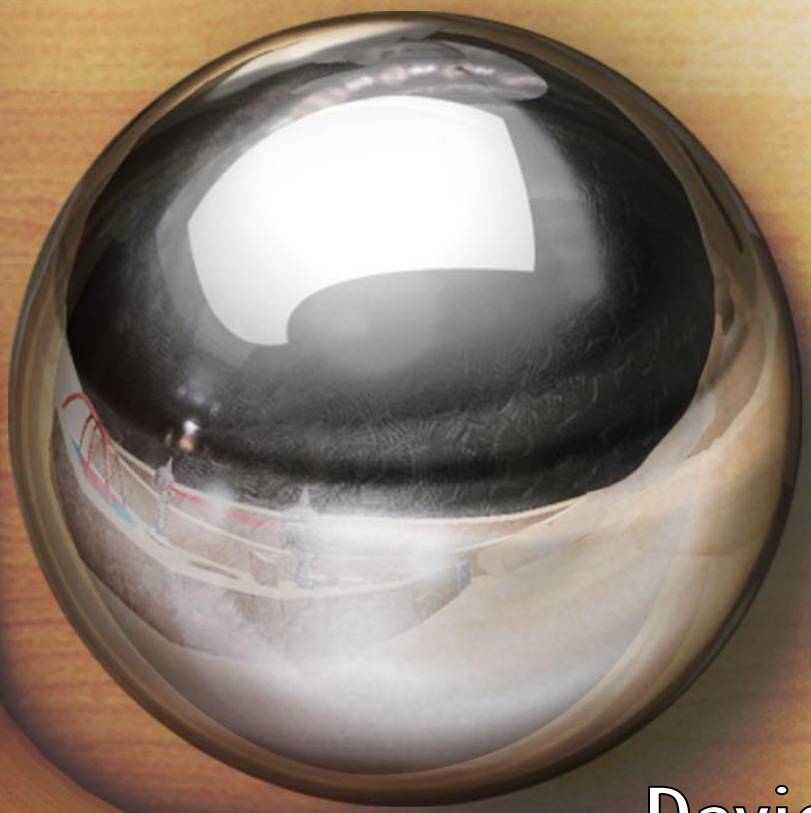
The Ultimate Labyrinth



David Romero Pastor,
Víctor Borrero López

Índice

1. Introducción

El proyecto elegido consiste en la creación de una versión del clásico juego en el que una bola, controlada por el jugador, debe alcanzar una meta superando para ello una serie de obstáculos que aportarán al juego un cierto nivel de dificultad, emoción y entretenimiento.

Para su desarrollo se han utilizado el microcontrolador MSP430G2553 como unidad de cálculo y proceso, el Educational Boosterpack MKII como aporte de periféricos y la placa de conexionado LAUNCH-BPII facilitada que cumple la función de nexo entre los pines de las dos placas anteriormente mencionadas.

La versión propia del juego consta, respecto a los aspectos de jugabilidad que se observarán en la pantalla, de:

- Una bola que se rige por unas ciertas leyes similares a las físicas (gravedad, cantidad de movimiento, rozamiento…) y que es controlada mediante el joystick/acelerómetro integrado en el Educational Boosterpack.
- Unos obstáculos que en este caso consisten en unos bloques o muros que impiden el paso de la bola a través de ellos y unos agujeros que hacen que la partida acabe si la bola cae en ellos. Además existen unos márgenes lo largo de todo el borde de la pantalla, que impedirán que la bola desaparezca.
- Una baldosa de comienzo y otra de meta al inicio y al final de cada nivel respectivamente.

En apartados posteriores se detallarán todos los aspectos de las componentes mencionada así como del resto de elementos de los que consta el juego.

2. Solución adoptada

Como se ha comentado en la introducción, el Educational Boosterpack MKII aporta los periféricos usados para interactuar con el programa.

Los periféricos de los que se hace uso son:

- El joystick/acelerómetro que, debido al bajo número de pines disponibles en el microprocesador para ser usados como entrada o salida, son excluyentes. Es decir, es posible el uso de ambos y con la misma función en el programa pero no simultáneamente. Es usado como periférico de entrada e interacción principal ya que mediante su lectura se modela el movimiento de la bola.
- Los dos botones, que simplemente cumplen la función de paso de la pantalla de 'inicio' a la de 'juego' o de la de 'fin' a la de 'inicio'. Pueden usarse uno u otro o los dos indiferentemente.

- El buzzer, que será usado para emitir un corto pitido cuando la bola choque con un muro o margen.
- La pantalla, que mostrará y permitirá al usuario conocer los distintos elementos y fases del juego en cada momento. Es por tanto el periférico de salida central e imprescindible.

3. Esquema eléctrico de montaje y planos del hardware implementado

Como se ha comentado anteriormente tan sólo se usan tres elementos en el montaje físico del proyecto o hardware: microcontroladorMSP430G2553, Educational Boosterpack MKII y placa de adaptación LAUNCH-BPII.Por tanto y, al no haber más elementos eléctricos o mecánicos implicados, se aportan a continuación diversos documentos que ayudarán al lector a comprender la naturaleza y conexionado de todos elementos.

- Microcontrolador MSP430G2553:
  - Link del datasheet completo: http://www.ti.com/lit/ds/symlink/msp430g2353.pdf
  - Páginas clave del datasheet, incluyen descripción, características e información referente al esquema eléctrico del modelo:

MIXED SIGNAL MICROCONTROLLER

FEATURES

- Low Supply-Voltage Range: 1.8 V to 3.6 V
- Ultra-Low Power Consumption
  - Active Mode: 230 µA at 1 MHz, 2.2 V
  - Standby Mode: 0.5 µA
  - Off Mode (RAM Retention): 0.1 µA
- Five Power-Saving Modes
- Ultra-Fast Wake-Up From Standby Mode in Less Than 1 µs
- 16-Bit RISC Architecture, 62.5-ns Instruction Cycle Time
- Basic Clock Module Configurations
  - Internal Frequencies up to 16 MHz With Four Calibrated Frequency
  - Internal Very-Low-Power Low-Frequency (LF) Oscillator
  - 32-kHz Crystal
  - External Digital Clock Source
- Two 16-Bit Timer_A With Three Capture/Compare Registers
- Up to 24 Touch-Sense-Enabled I/O Pins
- Universal Serial Communication Interface (USCI)
  - Enhanced UART Supporting Auto Baudrate Detection (LIN)
  - IrDA Encoder and Decoder
  - Synchronous SPI
  - I^2C™
- On-Chip Comparator for Analog Signal Compare Function or Slope Analog-to-Digital (A/D) Conversion
- 10-Bit 200-ksps Analog-to-Digital (A/D) Converter With Internal Reference, Sample-and-Hold, and Autoscan (See Table 1)
- Brownout Detector
- Serial Onboard Programming, No External Programming Voltage Needed, Programmable Code Protection by Security Fuse
- On-Chip Emulation Logic With Spy-Bi-Wire Interface
- Family Members are Summarized in Table 1
- Package Options
  - TSSOP: 20 Pin, 28 Pin
  - PDIP: 20 Pin
  - QFN: 32 Pin
- For Complete Module Descriptions, See the *MSP430x2xx Family User's Guide* (SLAU144)

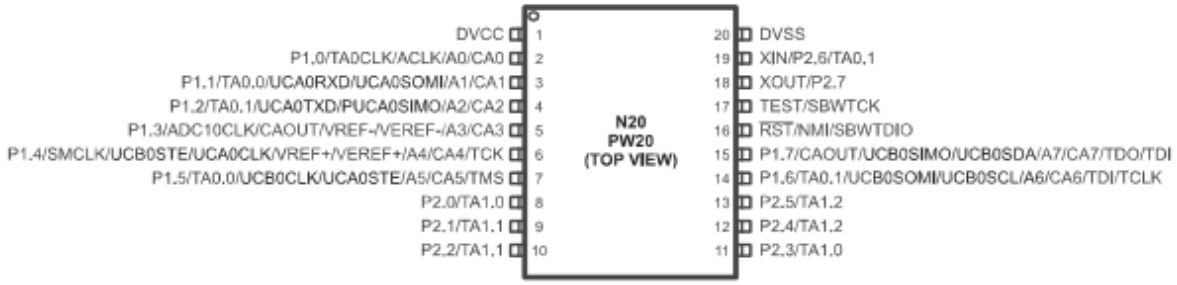
DESCRIPTION

The Texas Instruments MSP430 family of ultra-low-power microcontrollers consists of several devices featuring different sets of peripherals targeted for various applications. The architecture, combined with five low-power modes, is optimized to achieve extended battery life in portable measurement applications. The device features a powerful 16-bit RISC CPU, 16-bit registers, and constant generators that contribute to maximum code efficiency. The digitally controlled oscillator (DCO) allows wake-up from low-power modes to active mode in less than 1 µs.

The MSP430G2x13 and MSP430G2x53 series are ultra-low-power mixed signal microcontrollers with built-in 16-bit timers, up to 24 I/O touch-sense-enabled pins, a versatile analog comparator, and built-in communication capability using the universal serial communication interface. In addition the MSP430G2x53 family members have a 10-bit analog-to-digital (A/D) converter. For configuration details see Table 1.

Typical applications include low-cost sensor systems that capture analog signals, convert them to digital values, and then process the data for display or for transmission to a host system.

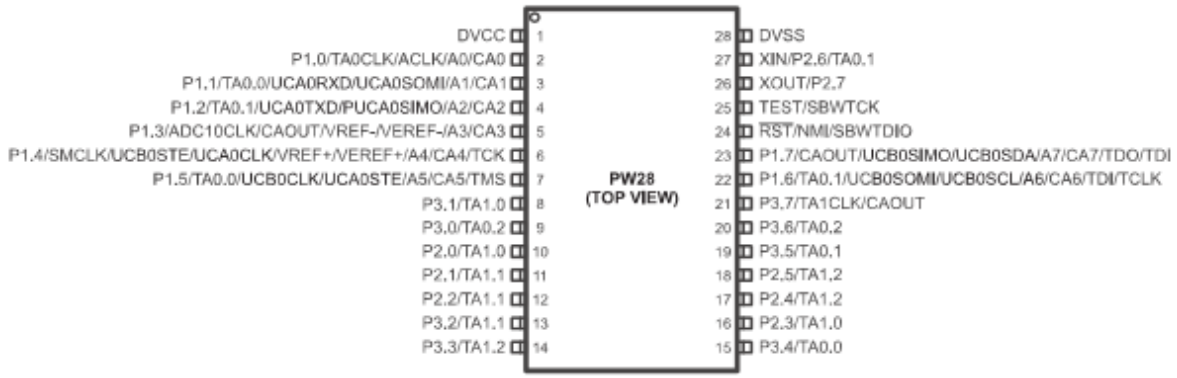
Device Pinout, MSP430G2x13 and MSP430G2x53, 20-Pin Devices, TSSOP and PDIP



NOTE: ADC10 is available on MSP430G2x53 devices only.

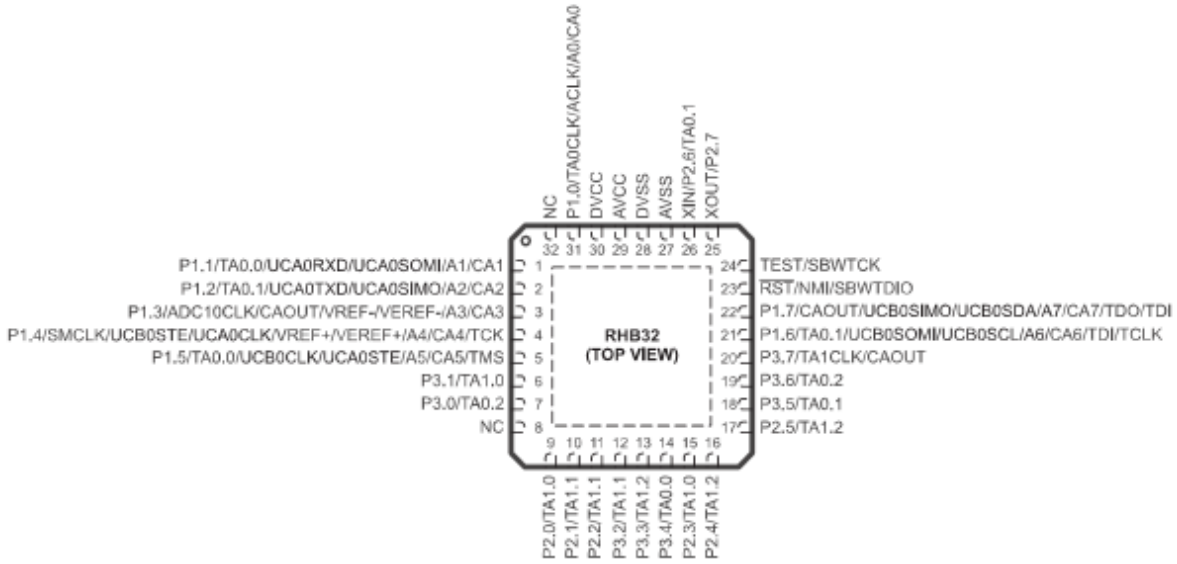
NOTE: The pulldown resistors of port P3 should be enabled by setting P3REN.x = 1.

Device Pinout, MSP430G2x13 and MSP430G2x53, 28-Pin Devices, TSSOP



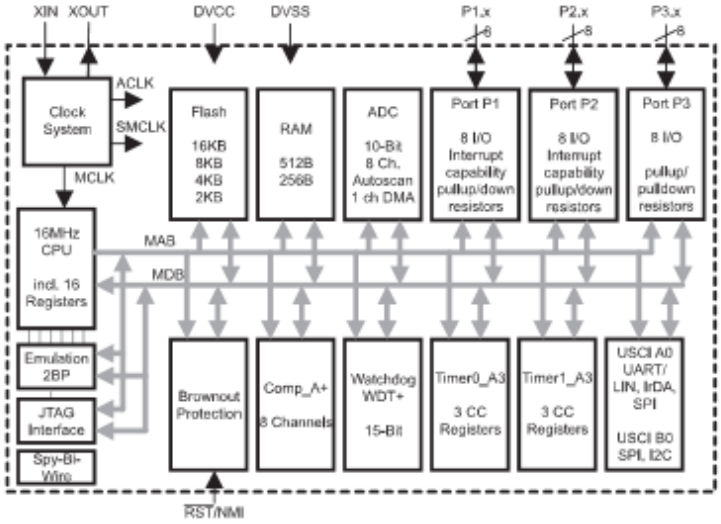
NOTE: ADC10 is available on MSP430G2x53 devices only.

Device Pinout, MSP430G2x13 and MSP430G2x53, 32-Pin Devices, QFN



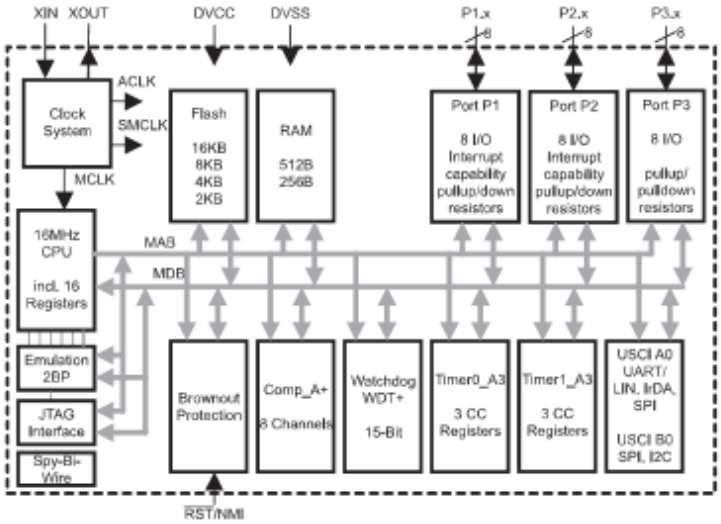
NOTE: ADC10 is available on MSP430G2x53 devices only.

Functional Block Diagram, MSP430G2x53



NOTE: Port P3 is available on 28-pin and 32-pin devices only.

Functional Block Diagram, MSP430G2x13



NOTE: Port P3 is available on 28-pin and 32-pin devices only.

- Educational Boosterpack MKII:
  - Link del datasheet completo: http://www.ti.com/lit/ug/slau599a/slau599a.pdf
  - Páginas clave del datasheet, incluyen descripción, características e información referente a los periféricos y al esquema eléctrico del modelo:

TEXAS INSTRUMENTS

User's Guide
SLAU599–August 2015

BOOSTXL-EDUMKII Educational BoosterPack™ Mark II Plug-in Module

The BOOSTXL-EDUMKII BoosterPack™ (see Figure 1) is an easy-to-use plug-in module that offers a high level of integration for developers to quickly add to LaunchPad™ designs. Various analog and digital inputs/outputs are at your disposal including an analog joystick, environmental and motion sensors, RGB LED, microphone, buzzer, color LCD display, and more.

This BoosterPack was developed with Energia in mind. Energia is an open-source community-developed coding environment, which is supported by a robust framework of intuitive APIs and easy-to-use software libraries for rapid firmware development. TI recommends Energia v12 or later. Learn more about Energia at www.energia.nu.

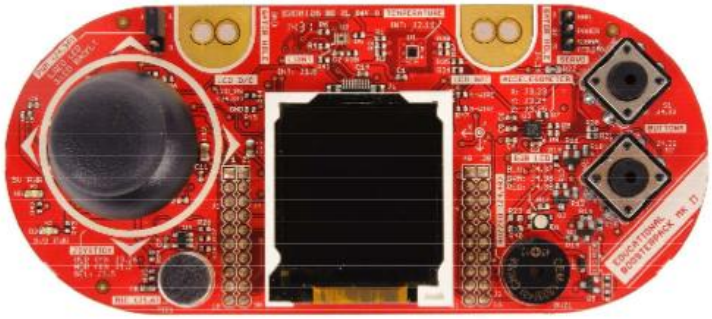


Figure 1. BOOSTXL-EDUMKII BoosterPack

1 Getting Started

1.1 Introduction

The BOOSTXL-EDUMKII BoosterPack is an easy-to-use plug-in module that offers a high level of integration for developers to quickly add to LaunchPad designs. Various analog and digital inputs and outputs are at your disposal including an analog joystick, environmental and motion sensors, RGB LED, microphone, buzzer, color LCD display, and more.

This BoosterPack was developed with Energia in mind. Energia is an open source, community developed coding environment, which is supported by a robust framework of intuitive APIs and easy-to-use software libraries for rapid firmware development. TI recommends Energia v12 or later. Learn more about Energia at www.energia.nu.

1.2 Key Features

- TI OPT3001 light sensor
- TI TMP006 temperature sensor
- Servo motor connector
- 3-axis accelerometer
- RGB multicolor LED
- Piezo buzzer
- Color 128x128 TFT LCD display
- Microphone
- 2-axis joystick with pushbutton
- User push buttons
- 40-pin BoosterPack standard for use with any LaunchPad kit

1.3 What's Included

1.3.1 Kit Contents

- 1 x BOOSTXL-EDUMKII BoosterPack Plug-in module
- 1 x Quick Start Guide

1.3.2 Software Examples

- MSP-EXP432P401R LaunchPad + BOOSTXL-EDUMKII demos (see Section 3)
  - BOOSTXL-EDUMKII_Accelerometer_MSP432P401R
  - BOOSTXL-EDUMKII_JoyStick_MSP432P401R
  - BOOSTXL-EDUMKII_LightSensor_MSP432P401R
  - BOOSTXL-EDUMKII_Temperature_MSP432P401R
  - BOOSTXL-EDUMKII_MicrophoneFFT_MSP432P401R

1.4 Next Steps: Looking Into the Provided Code

After the EVM features have been explored, the fun can begin. It's time to open an integrated development environment (IDE) and start looking at the code examples. Section 3 describes the example projects available to make it easy to understand the provided software. For more information on where to find and download an IDE, see Section 4.

2 Hardware

Figure 2 is an overview of the BOOSTXL-EDUMKII hardware.

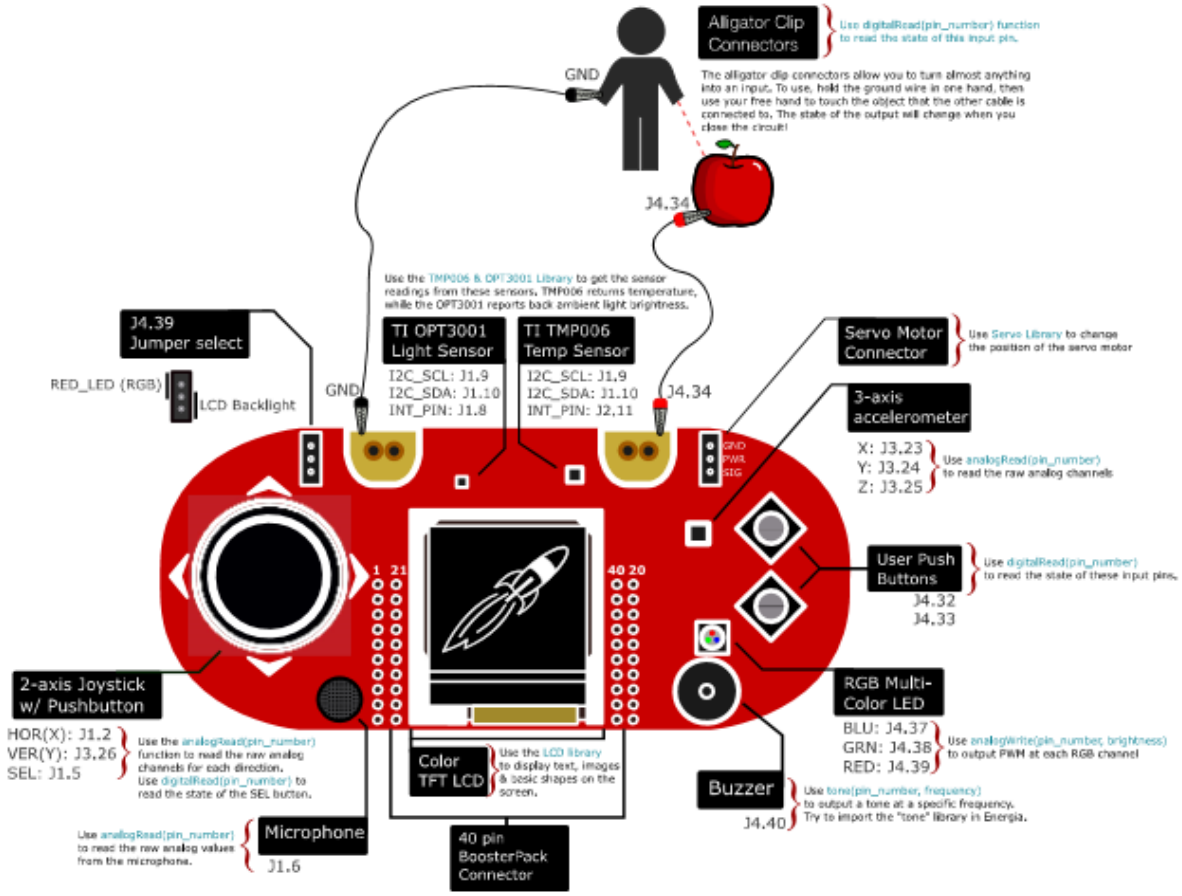


Figure 2. BOOSTXL-EDUMKII Overview

2.1 Hardware Features

2.1.1 BoosterPack Pinout

The Educational BoosterPack MKII adheres to the 40-pin LaunchPad and BoosterPack pinout standard (see Figure 3). A standard was created to aid compatibility between LaunchPad and BoosterPack tools across the TI ecosystem.

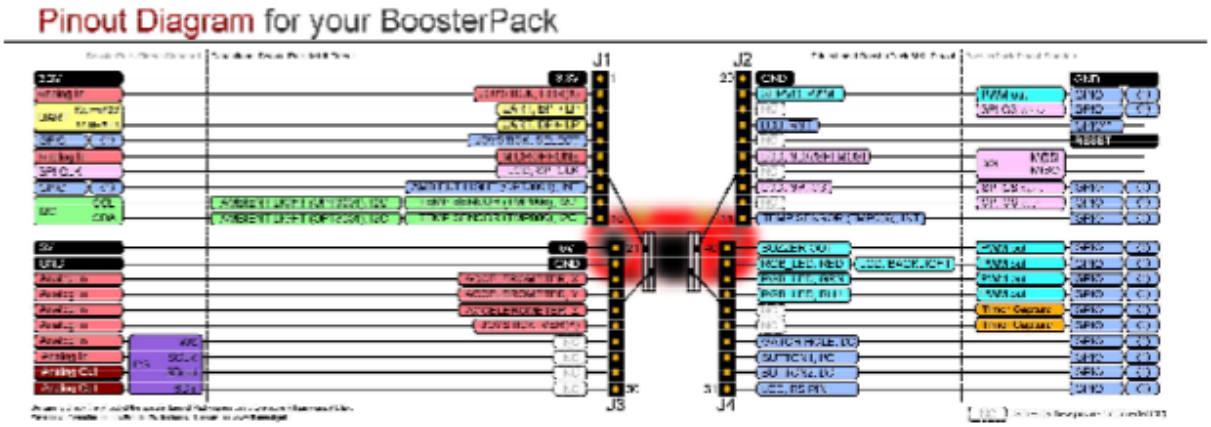


Figure 3. BoosterPack Pinout

The 40-pin standard is compatible with the 20-pin standard that is used by other LaunchPads like the MSP-EXP430G2. This allows for 20-pin LaunchPads to be used with 40-pin BoosterPacks with some limited functionality.

The BOOSTXL-EDUMKII supports BoosterPack stacking with its male/female BoosterPack headers. See how many BoosterPacks you can stack onto your LaunchPad to add more functionality like wireless and battery power.

More information about compatibility can also be found at http://www.ti.com/launchpad.

2.1.2 TI OPT3001 Light Sensor

The OPT3001 is a digital ambient light sensor (ALS) that measures the intensity of light as visible by the human eye. Covering the sensor with your finger or shining a flashlight on it will change the output of the OPT3001. The digital output is reported over an I²C- and SMBus-compatible, two-wire serial interface. The reference designator for the OPT3001 is U2.

More information on the OPT3001 light sensor can be found at http://www.ti.com/product/opt3001.

Table 1. OPT3001 Pinout

BoosterPack Header Connection	Pin Function
J1.8	OPT3001 interrupt
J1.9(1)	I²C SCL
J1.10(1)	I²C SDA

(1) Pin is multiplexed with the I²C communication lines of the TMP006.

2.1.3 TI TMP006 Temperature Sensor

The TMP006 is a digital infrared (IR) thermopile contactless temperature sensor that measures the temperature of an object without being in direct contact. Placing your hand over the sensor increases the sensor output. The digital output is reported over an I^2C- and SMBus-compatible two-wire serial interface. The reference designator for the TMP006 is U1.

More information on the TMP006 temperature sensor can be found at http://www.ti.com/product/tmp006.

Table 2. TMP006 Pinout

BoosterPack Header Connection	Pin Function
J1.9(1)	I^2C SCL
J1.10(1)	I^2C SDA
J2.11	TMP006 Interrupt

(1) Pin is multiplexed with the I^2C communication lines of the OPT3001.

2.1.4 Servo Motor Connector

The servo motor connector is a 3-pin header for the user to connect an external servo to be controlled. Users can connect a servo and control it through the application code. The reference designator for the servo motor connector is J8.

Table 3. Servo Motor Connector Pinout

BoosterPack Header Connection	Pin Function
J2.19	Servo Signal

NOTE: This kit does not include a servo motor and the user must provide one.

2.1.5 3-Axis Accelerometer

The Kionix KXTC9-2050 is a 3-axis analog accelerometer that measures g-forces. Moving the board along the axes will change the analog signal generated by the accelerometer. The reference designator for the accelerometer is U3.

More information on the 3-axis accelerometer can be found at http://www.kionix.com/product/KXTC9-2050.

Table 4. 3-Axis Accelerometer Pinout

BoosterPack Header Connection	Pin Function
J3.23	Accelerometer X-axis
J3.24	Accelerometer Y-axis
J3.25	Accelerometer Z-axis

2.1.6 RGB Multicolor LED

The Cree CLV1A-FKB RGB multicolor LED light output can make any color by mixing red, green, and blue. Each color channel can be individually modified by pulse width modulation (PWM) to achieve the desired color. The reference designator for the RGB LED is D1.

More information on the RGB multicolor LED can be found at http://www.cree.com/LED-Components-and-Modules/Products/High-Brightness/SMD-Color/SMD-Full-Color.

Table 5. RGB LED Pinout

BoosterPack Header Connection	Pin Function
J4.37	Blue channel
J4.38	Green channel
J4.39(1)	Red channel

(1) Pin is multiplexed with the LCD backlight pin through the jumper header J5.

2.1.7 Piezo Buzzer

The CUI CEM-1203(42) piezo buzzer can play various frequencies based on the user-provided PWM signal. You can even play different tones back to back to create a song. The reference designator for the piezo buzzer is BUZ1.

More information on the piezo buzzer can be found at http://www.cui.com/product/components/buzzers/audio-transducers/magnetic/cem-1203(42).

Table 6. Piezo Buzzer Pinout

BoosterPack Header Connection	Pin Function
J4.40	Buzzer input

2.1.8 Color 128x128-Pixel TFT LCD Display

The Crystalfontz CFAF128128B-0145T color 128x128-pixel TFT LCD supports display updates up to 20 frames per second (FPS) while only requiring a few lines to control the TFT LCD module through the SPI interface. This module has a color depth of 262K colors and a contrast ratio of 350. The reference designator for the color LCD is LCD1.

More information on the color LCD can be found at https://www.crystalfontz.com/product/cfaf128128b0145t-graphical-tft-128x128-lcd-display-module.

Table 7. Color LCD Pinout

BoosterPack Header Connection	Pin Function
J1.7	LCD SPI clock
J2.13	LCD SPI chip select
J2.15	LCD SPI MOSI
J4.31	LCD reset pin
J4.39(1)	LCD backlight

(1) Pin is multiplexed with the RGB LED red channel pin through the jumper header J5.

2.1.9 Microphone

The CUI CMA-4544PF-W electret microphone uses an OPA344 operational amplifier to boost the output of the microphone. The human ear can hear frequencies between 0 and 20 kHz and the operating range of the microphone is 20 Hz to 20 kHz. The reference designator for the microphone is MIC1.

More information on the microphone can be found at http://www.cui.com/product/components/microphones/electret-condenser-microphone/cma-4544pf-w.

Table 8. RGB LED Pinout

BoosterPack Header Connection	Pin Function
J1.6	Microphone Output

2.1.10 2-Axis Joystick With Pushbutton

The ITEAD studio IM130330001 2-axis joystick with pushbutton is simply two potentiometers, one for each axis. The select button is actuated when the joystick is pressed down. The analogRead statement reads the voltage present on the joystick axis to provide the position of the joystick to the application (for example, pushing the joystick to the left reads X = 0). The reference designator for the analog joystick is JS1.

More information on the analog joystick can be found at http://imall.itead.cc/playstation2-analog-joystick.html.

Table 9. Joystick Pinout

BoosterPack Header Connection	Pin Function
J1.2	Horizontal X-axis
J1.5	Select button
J3.26	Vertical Y-axis

2.1.11 User Pushbuttons

The user pushbuttons on the BOOSTXL-EDUMKII are connected to pullup resistors that drive the BoosterPack pin high until the button is pressed and the pin is driven low. The reference designators for the user pushbuttons are S1 and S2.

Table 10. User Pushbuttons Pinout

BoosterPack Header Connection	Pin Function
J4.32	S2 button
J4.33	S1 button

2.2 Power

The board was designed to be powered by the attached LaunchPad, and requires both 3.3V and 5V power rails. Some 20 pin LaunchPads like MSP-EXP430FR4133 may not provide the necessary 5V power, which will limit the functionality.

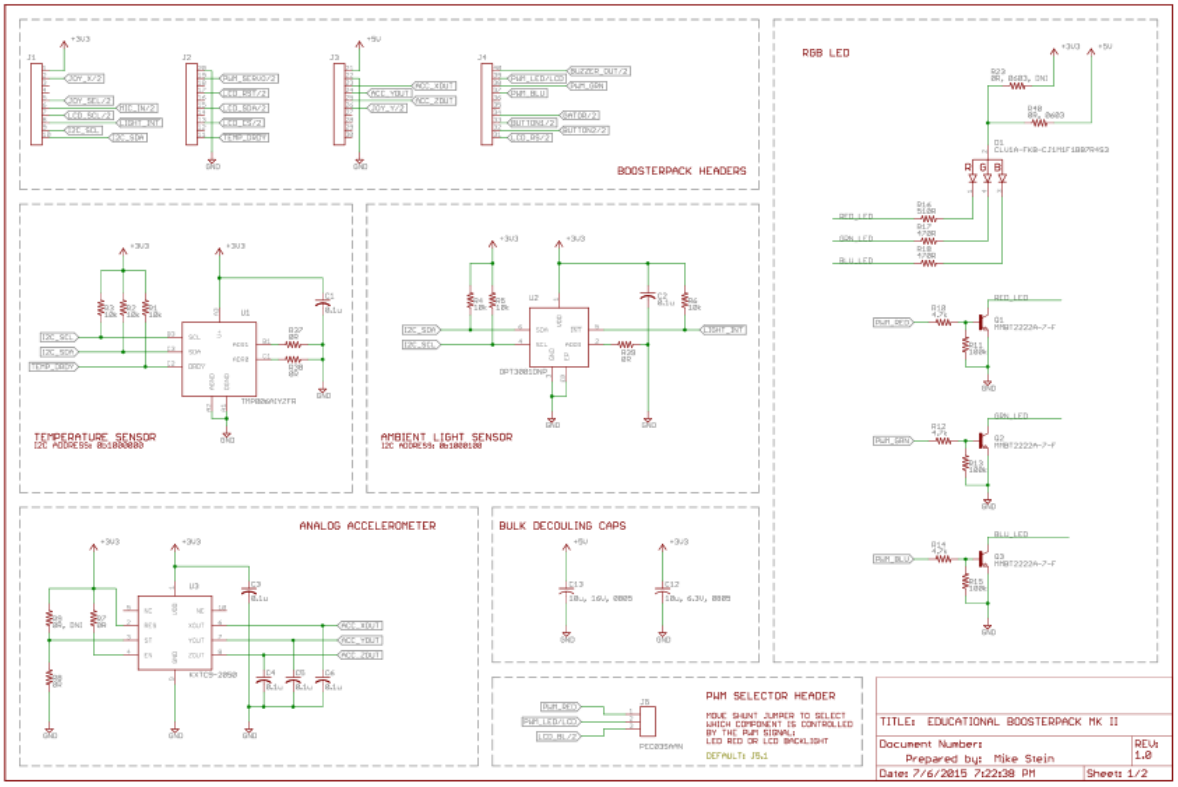


Figure 10. Schematics (1 of 2)

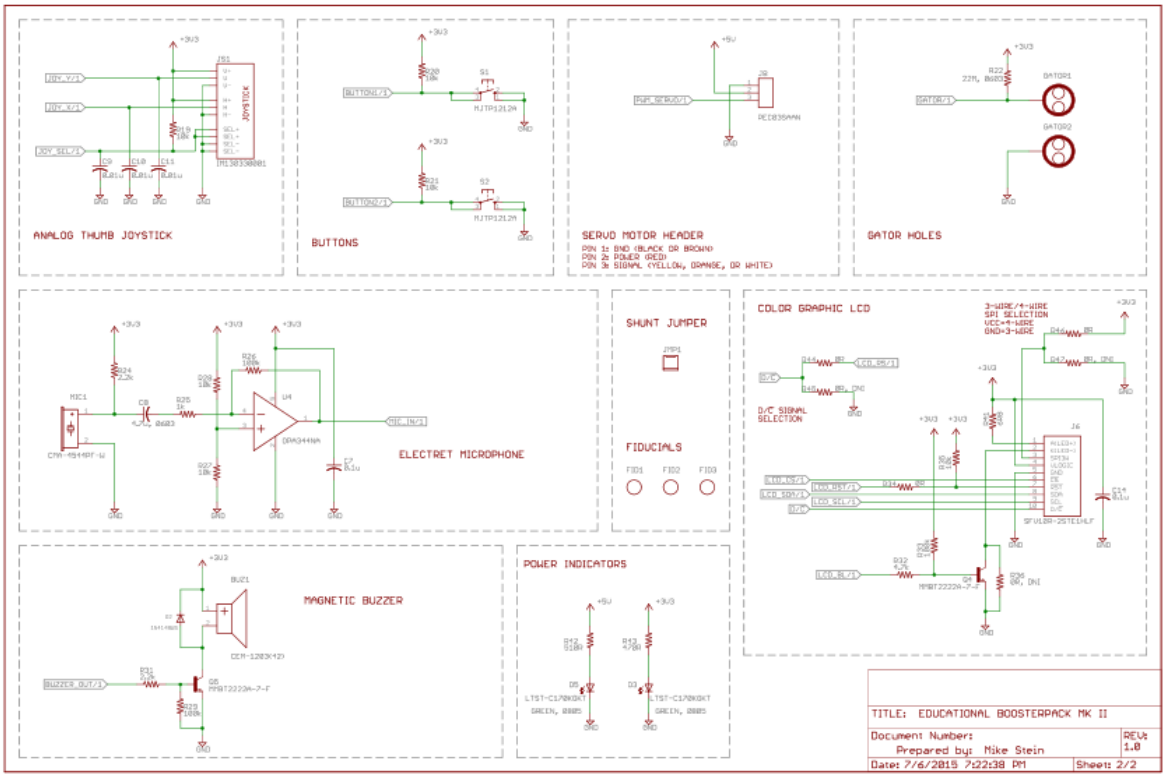
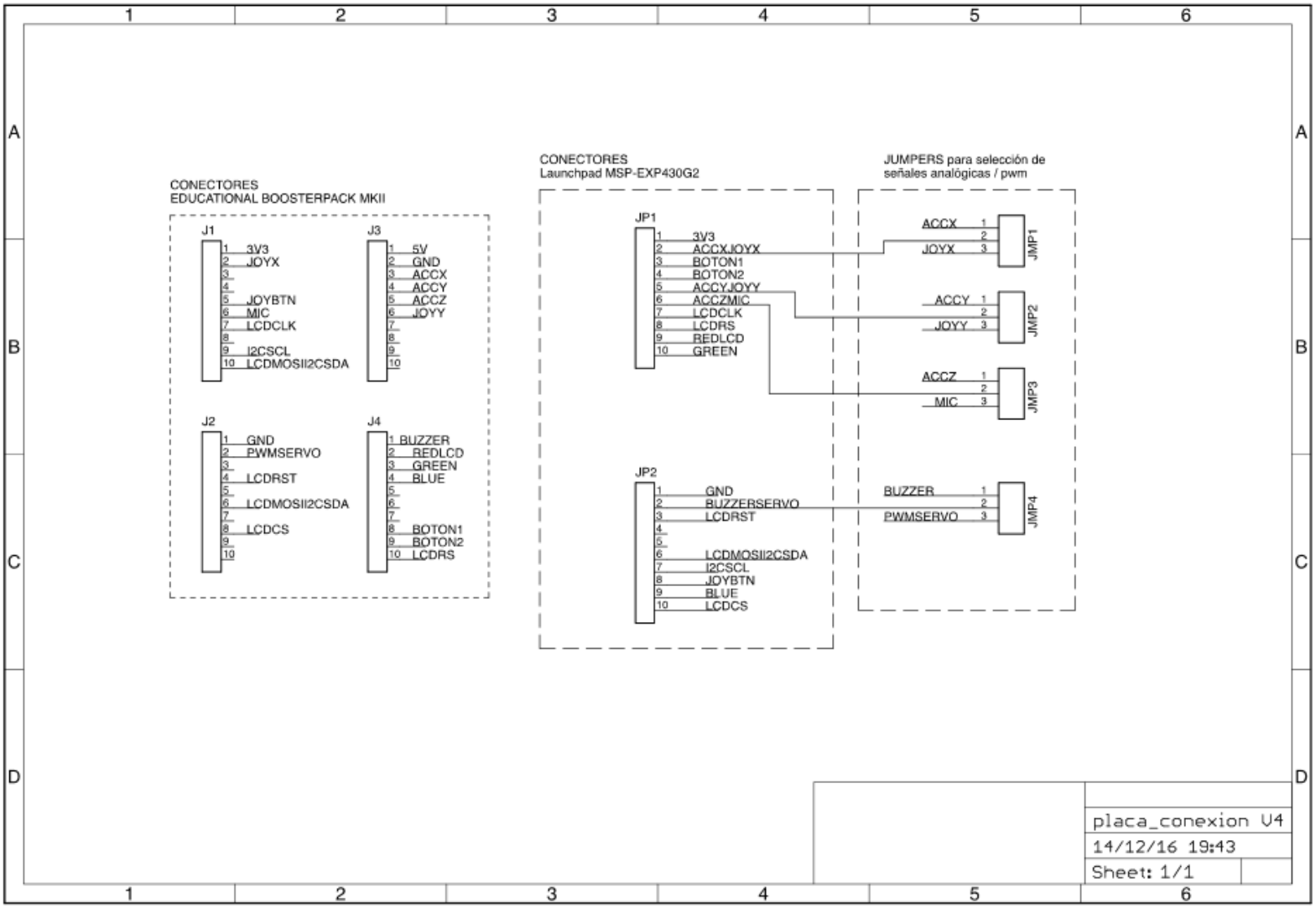


Figure 11. Schematics (2 of 2)

- Placa de adaptación LAUNCH-BPII:
  - Esquema de conexionado:



En este caso los jumpers de este elemento están colocados de manera que habilitan:

- Las componentes X e Y del acelerómetro o las componentes X e Y del

joystick, debiéndose usarse las parejas predeterminadas para mayor facilidad de manejo del periférico elegido (JMP1 y JMP2).
    - La posición del jumper JMP3 es indiferente, ya que no se usarán ninguno de los periféricos que conecta.
    - El uso del buzzer, excluyendo el uso de un servo (JMP4).

Para el conexionado final de los tres elementos simplemente habría que conectar el microcontrolador a la placa de adaptación y ésta al Boosterpack, teniendo especial atención de conectarlas con las placas en las direcciones adecuadas y no al revés, ya que esto último conllevaría un conexionado inadecuado y probablemente problemas y/o daños en el hardware.

4. Explicación del código

La estructura principal del código consiste en una máquina de estados que rige las distintas pantallas o fases del juego.

Los estados son:

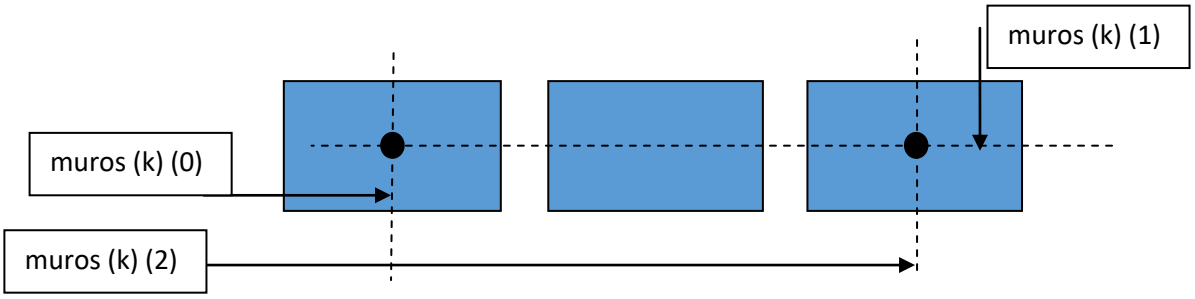
- Pantalla de inicio: en esta pantalla se muestra un texto sobre un fondo liso con el nombre del juego "The Ultimate Laberynth" y una instrucción para continuar a la siguiente pantalla parpadeando, "Press START". Para pasar al siguiente estado 'Actualización de la velocidad y dibujo del nivel' se ha de pulsar cualquiera de los botones del Boosterpack.

- Actualización de la velocidad y dibujo del nivel: en este estado se actualiza la velocidad de la bola (componentes X e Y) en función de los valores del acelerómetro/joystick leídos mediante el convertidor ADC. Además si el estado anterior fue 'Pantalla de inicio' o 'Pasa de nivel' se realiza el dibujo del nivel siguiente completo. Para realizar el mapa de la pantalla se han creado unas matrices de 7x7, donde cada elemento ocupa 16x16 píxeles de la pantalla. Se tendrán tres tipos: si el valor es 0, se dibuja el fondo; si es 1, se dibuja un muro y si es 2 se dibuja un agujero.

  Se pasa al siguiente estado 'Actualización de la posición' automáticamente.

- Actualización de la posición: en este estado se llevan a cabo la mayoría de las acciones que tienen que ver con el movimiento de la bola y su interacción con los elementos del juego:
    - En primer lugar se realiza la saturación de la posición de la bola al chocar con un margen o un muro, es decir, impide que la posición se incremente o decrezca, dependiendo del caso, para que la bola no "atraviese" estos elementos.

En concreto, para el choque con los muros, se ha empleado el uso de una matriz, que se comprobará para cada posición de la bola, llamada 'muros de 5x3', donde se almacena la posición x del bloque, la posición y, y en la tercera columna, la posición x más a la derecha, que nos permitirá colocar muros horizontales conformados por más de un bloque de manera consecutiva. En casos en los que el muro esté formado por un solo bloque, el valor de la tercera columna coincidiría con el de la primera. Veamos la siguiente figura de un muro formado por tres bloques:



Además de la saturación, también se lleva a cabo el efecto de rebote en estos elementos, cambiando el sentido la velocidad cuando la bola llega a estas posiciones límite y tiene una velocidad no nula. El buzzer emite un pequeño pitido cuando se produce un rebote.

- En segundo lugar lleva al programa al estado de 'Animación de fin de partida' cuando la bola "cae" en un agujero. Para realizar dicha "caída", en este caso, se ha usado un método similar al de las matrices 'muros'. Guardamos en una matriz 'agujeros' 5x2 la posición de cada agujero. De este modo, comparando la posición de la bola y la del agujero podemos establecer una condición de "muerte".
  Este estado también se encarga de pintar de nuevo los agujeros ya que la bola, al pasar cerca puede borrarlos parcialmente.
- En tercer lugar lleva al programa al estado 'Pasa de nivel' si la bola llega a la casilla de meta, siempre que la meta no pertenezca al último nivel diseñado, caso en el que el programa se dirigirá a 'Pantalla de victoria'.
  También se encarga de pintar de nuevo la meta ya que la bola, al pasar cerca puede borrarla parcialmente.

Si la bola no "cae" en un agujero ni llega a la meta el programa se dirigirá automáticamente de nuevo al estado 'Actualización de la velocidad y dibujo del nivel'.

- Animación de fin de partida: en este estado se lleva a cabo la "muerte" de la bola y consecuentemente el fin de la partida. La animación dicha consiste en que el agujero en el cual la bola "cae" se hace cada vez más grande, hasta

ocupar toda la pantalla, mientras que simultáneamente la bola se dirige al centro de ésta, trazando trayectorias rectilíneas. Una vez que la bola se sitúa en el centro el programa se dirigirá al estado 'Pantalla de fin'.

- Pantalla de fin: aparece un texto que dice "GAME OVER" parpadeando. Para volver a 'Pantalla de inicio' bastaría con pulsar uno de los dos botones aportados por el Boosterpack.

- Pasa de nivel: en este estado, como su propio nombre indica, se produce el cambio de nivel, de manera que una vez la bola ha conseguido situarse completamente dentro de la casilla de meta, el nivel actual empieza a desaparecer suavemente por la parte superior de la pantalla mientras que el nuevo nivel aparece por la parte inferior. Esto se realiza mediante un bucle que va leyendo las dos matrices simultáneamente y va disminuyendo la componente 'y' de cada elemento. La antigua matriz empieza en la posición normal y desaparece, la segunda está inicializada justo "debajo" de la pantalla y pasa a ocupar el centro de esta. Mientras que se realiza este movimiento, la bola desaparece de la pantalla, para aparecer en la casilla de salida del nuevo nivel una vez éste ya se ha establecido por completo.

- Pantalla de victoria: aparece el texto "YOU WIN!" durante unos segundos y a continuación se vuelve al estado 'Pantalla de inicio'.

A continuación se muestra el diagrama de bolas de la máquina de estados explicada para mayor claridad.

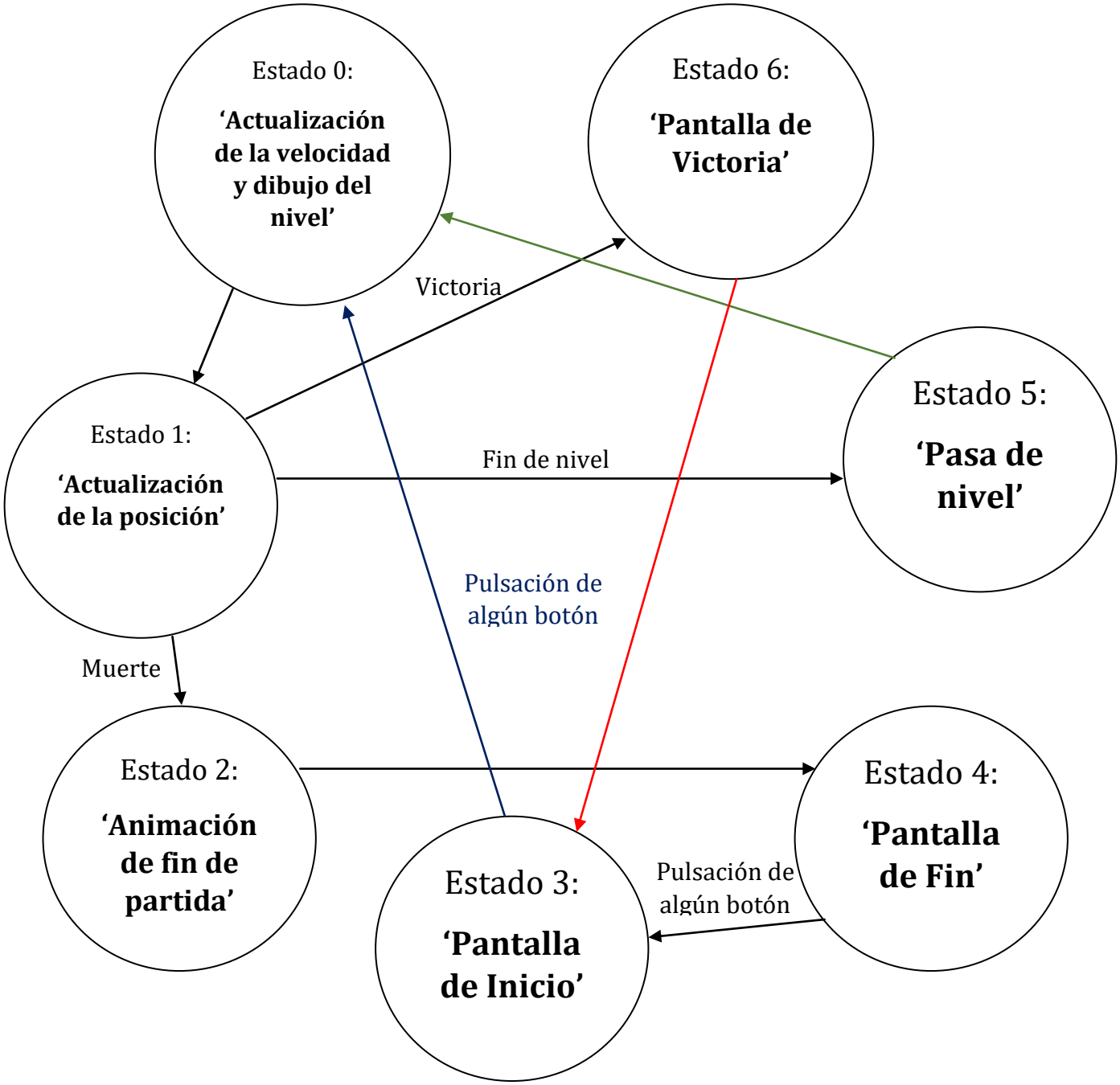


Imagen 1

5. Resultados obtenidos

A continuación se mostrarán una serie de imágenes y vídeos mediante los cuales se observa con mayor claridad el funcionamiento y la efectividad del juego.

En esta fotografía se puede ver la pantalla de inicio esperando a la pulsación de algún botón para comenzar el juego.

Imagen 2

En esta segunda imagen se puede apreciar el primer nivel, donde se distinguen claramente los agujeros (círculos negros), los muros (rectángulos marrones), los límites de la pantalla (margen azul), la casilla de inicio (cuadrado verte situado en la parte superior) y la casilla final de nivel (cuadrado verde situado en la parte inferior).

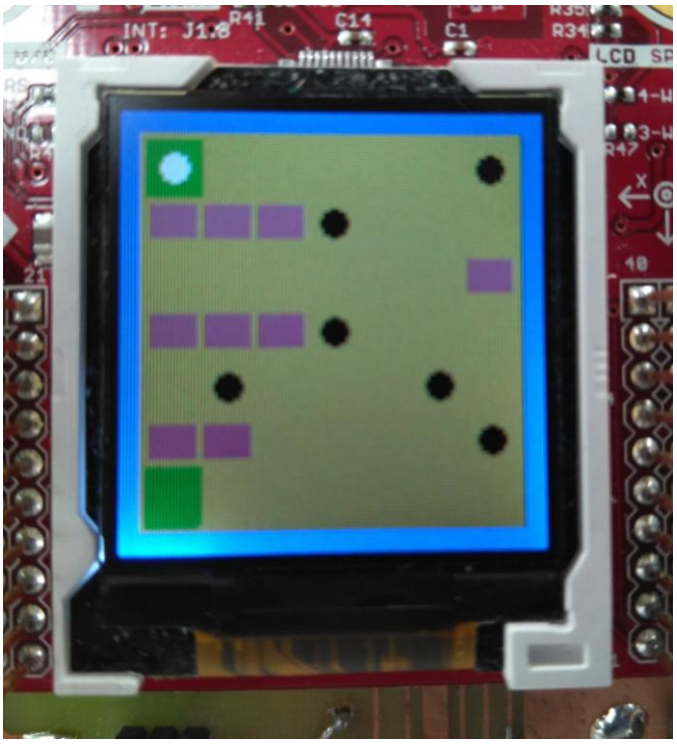


Imagen 3

En tercer lugar, se puede apreciar el cambio de nivel en una fase intermedia.

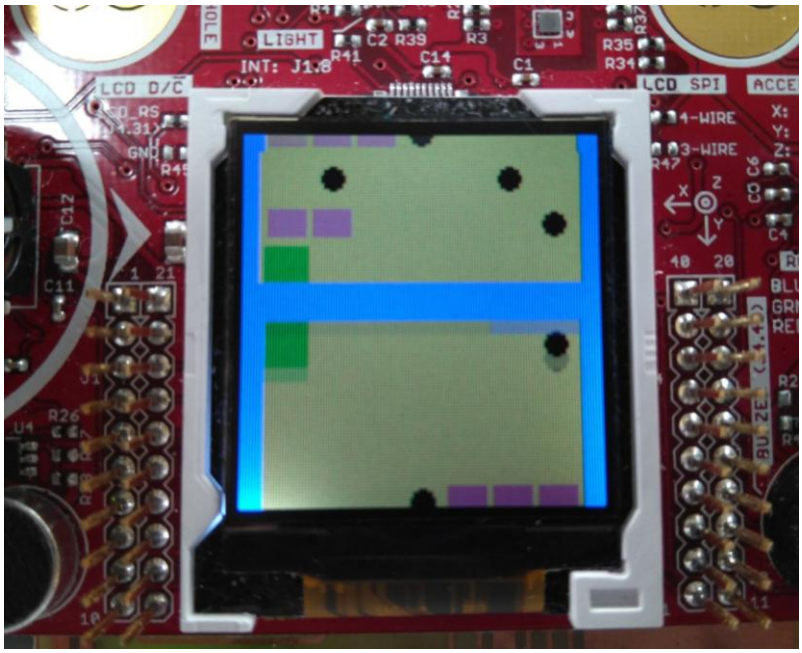


Imagen 4

En caso de que la bola “muera”, se produce la animación correspondiente, con su posterior pantalla de fin de partida. Ésta última se puede apreciar en la siguiente imagen:

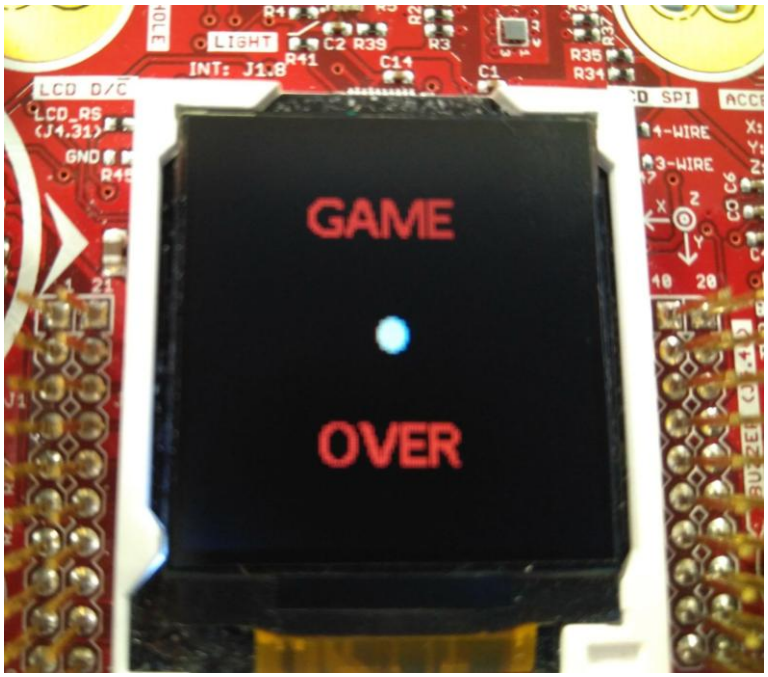


Imagen 5

Finalmente, una vez que se superen todos los niveles, aparecerá la siguiente pantalla, que indicará el haber superado el juego.

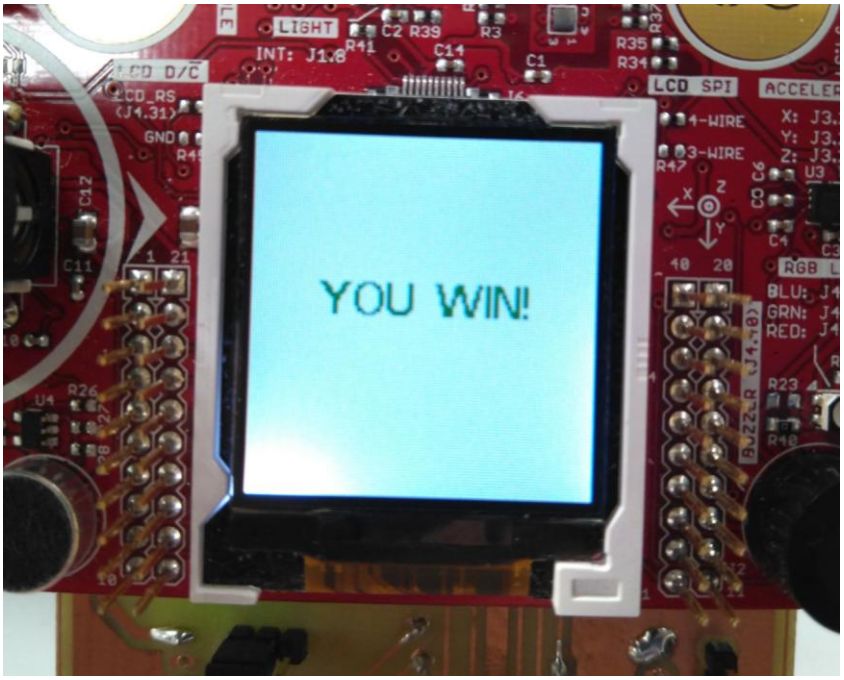


Imagen 6

Además, en el vídeo de muestra que acompaña el presente documento se puede ver el funcionamiento completo de nuestro juego y así apreciar claramente el tipo de movimiento implementado para la bola.

6. Código comentado

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


intlee_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
    }

voidinicia_ADC(char canales){
    ADC10CTL0 &= ~ENC;      //deshabilita ADC
```

```
ADC10CTL0 = ADC10ON | ADC10SHT_3 | SREF_0|ADC10IE; //enciende ADC, S/H lento,
REF:VCC, 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
}

voidconf_reloj(char VEL){
    BCSCTL2 = SELM_0 | DIVM_0 | DIVS_0;
switch(VEL){
case 1:
if (CALBC1_1MHZ != 0xFF) {
            DCOCTL = 0x00;
            BCSCTL1 = CALBC1_1MHZ;      /* Set DCO to 1MHz */
            DCOCTL = CALDCO_1MHZ;
        }
break;
case 8:

if (CALBC1_8MHZ != 0xFF) {
__delay_cycles(100000);
            DCOCTL = 0x00;
            BCSCTL1 = CALBC1_8MHZ;      /* Set DCO to 8MHz */
            DCOCTL = CALDCO_8MHZ;
        }
break;
case 12:
if (CALBC1_12MHZ != 0xFF) {
__delay_cycles(100000);
            DCOCTL = 0x00;
            BCSCTL1 = CALBC1_12MHZ;     /* Set DCO to 12MHz */
            DCOCTL = CALDCO_12MHZ;
        }
break;
case 16:
if (CALBC1_16MHZ != 0xFF) {
__delay_cycles(100000);
            DCOCTL = 0x00;
            BCSCTL1 = CALBC1_16MHZ;     /* Set DCO to 16MHz */
            DCOCTL = CALDCO_16MHZ;
        }
break;
default:
if (CALBC1_1MHZ != 0xFF) {
DCOCTL = 0x00;
            BCSCTL1 = CALBC1_1MHZ;      /* Set DCO to 1MHz */
            DCOCTL = CALDCO_1MHZ;
        }
break;

    }
    BCSCTL1 |= XT2OFF | DIVA_0;
    BCSCTL3 = XT2S_0 | LFXT1S_2 | XCAP_1;
}

Graphics_Context g_sContext;
```

```
char
i,j,k=0,kmax=0,l,aux,z=0,zmax=0,muerte=0,fin=0,salto,kmuerte,victoria=0,nivel
=0, ini=0, t=0, pulso=0; //Variables auxiliares
int espera=0;
constchar radio=4,ancho_margen=6,ancho=6,largo=8,radio_peq=3; //Dimensiones
constantes
int x,y; //Lectura del joystick/acelerómetro
char estado=3; //Estado de la máquina de estados. Comenzamos en el estado
‘Pantalla de Inicio’.

char mat[7][7]; //Matriz auxiliar (para cambiar de nivel)

/*Las siguientes matrices servirán para establecer el mapa de la pantalla
para cada nivel, de manera que el valor 0 será fondo, el valor 1 será muro y
el valor 2 será agujero. Cada elemento de la matriz ocupará16x16 píxeles.*/
char mat1[7][7]={{0,0,0,0,0,0,2}, //Matriz del primer nivel
                 {1,1,1,2,0,0,0},
                 {0,0,0,0,0,0,1},
                 {1,1,1,2,0,0,0},
                 {0,2,0,0,0,2,0},
                 {1,1,0,0,0,0,2},
                 {0,0,0,0,0,0,0}
};
char mat2[7][7]={{0,0,0,0,0,0,2}, //Matriz del segundo nivel
                 {0,0,0,0,0,0,0},
                 {0,0,0,0,0,0,0},
                 {1,1,0,2,1,1,1},
                 {0,2,0,0,0,0,0},
                 {0,0,0,0,0,0,2},
                 {0,1,1,1,1,0,0}
};
char muros[5][3]; //Matriz que guarda las posiciones de los muros
char agujeros[5][2]; //Matriz que guarda las posiciones de los agujeros
int vel_x, vel_y; //Velocidades de la bola

constint Pos_ini_x=16, Pos_ini_y=16;  //Posiciones iniciales de la bola
int Posicion_Bola_x=16, Prev_pos_x=16; //Posición del eje x de la bola
int Prev_pos_y=16, Posicion_Bola_y=16; //Posición del eje y de la bola

constint limite1=999,limite2=9999; //PWM para el pitido del buzzer


intmain(void) {

    WDTCTL = WDTPW | WDTHOLD;   // Stop watchdog timer
    conf_reloj(16);
Crystalfontz128x128_Init();

/* Set default screen orientation */
Crystalfontz128x128_SetOrientation(LCD_ORIENTATION_UP);

/* Initializes graphics context */
Graphics_initContext(&g_sContext, &g_sCrystalfontz128x128);
Graphics_setFont(&g_sContext, &g_sFontCmss18b); //Tipo y tamaño de la fuente

    inicia_ADC(BIT0); //Leemos la salida del convertidor ADC

        P1OUT=BIT1+BIT2; //Declaramos los pines P1.1 y P1.2 como entrada
         P1REN=BIT1+BIT2; //Colocamos las resistencias pull-up a dichos pines
```

```
        TA1CCTL0=CCIE;          //CCIE=1
TA1CTL=TASSEL_1| MC_1;  //ACLK, DIV=1, UP
TA1CCR0=499;           //periodo=5ms

//Declaración del BIT6 como PWM para el buzzer
P2DIR|=BIT6;
        P2SEL|=BIT6;
        P2SEL2&=~(BIT6+BIT7);
        P2SEL&=~(BIT7);
        TA0CCTL1=OUTMOD_7;
        TA0CTL=TASSEL_2| MC_1;
TA1CCTL0=CCIE;
        TA1CTL=TASSEL_1| MC_1;
        TA0CCR0=0;
        TA0CCR1=0;

__bis_SR_register(GIE); //Habilitación de las interrupciones

//Inicialización de la matriz auxiliar como primer nivel (mat1)
for(i=0; i<7; i++){
for(j=0; j<7; j++){
            mat[i][j]=mat1[i][j];
}
    }

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

//Leemos las salidas del joystick/acelerómetro del ADC y las almacenamos en x
e y
            x=lee_ch(0);
            y=lee_ch(3);

/*****************MÁQUINA DE ESTADOS DEL MOVIMIENTO*********************/
switch (estado){



/****ESTADO 0:ACTUALIZACIÓN DE VELOCIDAD Y DIBUJO COMPLETO DEL NIVEL****/
case 0:
//Sólo se pinta el nivel completo cuando se viene del estado de 'Pantalla de
inicio' o ‘Pasa de nivel’. Para ello usamos la variable 'ini'.
if(ini==1){
                        ini=0;

Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GOLDENROD);
Graphics_clearDisplay(&g_sContext);
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLUE);
//Se dibuja el margen
for(i=0;i<ancho_margen;i++){
Graphics_Rectangle marco = {
                                    0+i,
                                    0+i,
                                    127-i,
                                    127-i
                            };
 Graphics_drawRectangle(&g_sContext,&marco);
```

```
}
//Se dibujan los elementos de la matriz
for (i=0; i<7; i++){
for(j=0; j<7; j++){
/***BALDOSA DE INICIO Y DE META****/
if((i==0 && j==0) || (i==6 && j==0)){
Graphics_Rectangle cuadrado ={
                                                    (j+1)*16-8,
                                                    (i+1)*16-8,
                                                    (j+1)*16+8,
                                                    (i+1)*16+8
                                                                };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GREEN);
Graphics_fillRectangle(&g_sContext,&cuadrado);
                                            }
/****FONDO******/
elseif(mat[i][j]==0){
Graphics_Rectangle cuadrado ={
                                                    (j+1)*16-8,
                                                    (i+1)*16-8,
                                                    (j+1)*16+8,
                                                    (i+1)*16+8
                                                                };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GOLDENROD);
Graphics_fillRectangle(&g_sContext,&cuadrado);
}
/*****MUROS******/
/*La matriz 'muros' servirá para almacenar las posiciones de los bloques de
muros. Se guardan la posición "x" inicial, la final y la posición "y".*/
elseif(mat[i][j]==1){
if(mat[i][j-1]==1 && j!=0)
                                                muros[aux][2]=(j+1)*16;
else{
                                                muros[k][0]=(j+1)*16;
                                                muros[k][1]=(i+1)*16;

muros[k][2]=(j+1)*16;
                                                aux=k;
k++;
                                            }
if(j==0)
                                                muros[aux][0]=8;

if(j==6)
muros[aux][2]=119;
                                            kmax=k;

Graphics_Rectangle cuadrado1 ={
(j+1)*16-8,
                                                    (i+1)*16-8,
                                                    (j+1)*16+8,
                                                    (i+1)*16+8
                                            };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GOLDENROD);
Graphics_fillRectangle(&g_sContext,&cuadrado1);

Graphics_Rectangle cuadrado ={
                                                (j+1)*16-(largo-1),
(i+1)*16-(ancho-1),
                                                (j+1)*16+(largo-2),
                                                (i+1)*16+(ancho-2)
```

```
                                                                    };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BROWN);
Graphics_fillRectangle(&g_sContext,&cuadrado);
}
/*****AGUJEROS*****/
/*La matriz 'agujeros' servirá para almacenar las posiciones de los agujeros.
Se guardan la posición 'x' e 'y'.*/
elseif(mat[i][j]==2){

Graphics_Rectangle cuadrado ={
                                                    (j+1)*16-8,
                                                    (i+1)*16-8,
(j+1)*16+8,
                                                    (i+1)*16+8
                                                                };

                                                agujeros[z][0]=(j+1)*16;
                                                agujeros[z][1]=(i+1)*16;
z++;
                                                zmax=z;

Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GOLDENROD);
Graphics_fillRectangle(&g_sContext,&cuadrado);
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
Graphics_fillCircle(&g_sContext,(j+1)*16,(i+1)*16,4);

}
                                        }

                                    }
                    }

                    TA0CCR0=0;
                    TA0CCR1=0;

/*ACTUALIZACIÓN DE VELOCIDADES*/
/*Dicha actualización se realizará de manera que, en función del valor dado
por el acelerómetro/joystick, la velocidad sea una u otra.*/

//Velocidad en el eje x
if(x>514-10 && x<514+10)
{
if(vel_x > 0) {
                            vel_x-=1;
if(vel_x<0){
                                vel_x=0;
                             }
                        }

elseif(vel_x < 0){
                            vel_x+=1;
if(vel_x > 0)
                                vel_x=0;
}
                    }
elseif(x>=514+10 && x<514+10+150)
                        vel_x+=1;
elseif(x>=514+10+150)
                         vel_x+=2;
```

```
elseif(x<=514-10 && x>514-10-150)
                        vel_x-=1;
elseif(x<=514-10-150)
                        vel_x-=2;

//Velocidad en el eje y
if(y>507-10 && y<507+10)
                    {
if(vel_y > 0) {
                            vel_y-=1;
if(vel_y<0){
                                vel_y=0;
                             }
                        }

elseif(vel_y < 0){
                            vel_y+=1;
if(vel_y > 0)
                                vel_y=0;
                        }
                    }
elseif(y>=507+10 && y<507+10+150)
                        vel_y-=1;
elseif(y>=507+10+150)
                        vel_y-=2;
elseif(y<=507-10 && y>507-10-150)
                        vel_y+=1;
elseif(y<=507-10-150)
                        vel_y+=2;

/******SATURACIÓN DE LA VELOCIDAD*********/
/*Saturamos la velocidad a un valor de 6.*/
if(vel_x>6)
                        vel_x=6;
elseif(vel_x<-6)
                        vel_x=-6;

if(vel_y>6)
                        vel_y=6;
elseif(vel_y<-6)
                        vel_y=-6;

                    estado=1; //Pasamos al estado 1: 'Actualización de la
posición'.
break;

/**********ESTADO 1:ACTUALIZACIÓN DE LA POSICIÓN*********************/
/*El rebote se producirá de manera que al incidir en uno de los límites, ya
sea margen o muro, la velocidad que lleve tome sentido contrario. Además, se
producirá un pequeño pitido si la posición actual de la bola es diferente de
la anterior, de manera que no pite siempre que entre en contacto con un muro
o margen.*/
case 1:
//Sumamos a la posición de la bola en cada eje su velocidad correspondiente.
                    Posicion_Bola_x+=vel_x;
                    Posicion_Bola_y+=vel_y;

/**SATURACIÓN DE LA POSICIÓN CON LOS MÁRGENES*****/
 /***Saturación de la posición con el margen de la derecha***/
```

```
if (Posicion_Bola_x<=(ancho_margen+radio)){
                        Posicion_Bola_x=(ancho_margen+radio);
if(vel_x!=0){
                            vel_x=-(vel_x);
if(Prev_pos_x!=Posicion_Bola_x){
                                TA0CCR0=limite2;
                                TA0CCR1=limite1;
                            }
                        }
                    }
/*Saturación de la posición con el margen de la izquierda*/
elseif(Posicion_Bola_x>=127-(ancho_margen+radio)){
Posicion_Bola_x=127-(ancho_margen+radio);
if(vel_x!=0){
                            vel_x=-(vel_x);
if(Prev_pos_x!=Posicion_Bola_x){
                                TA0CCR0=limite2;
                                TA0CCR1=limite1;
                            }
                        }
                    }
/***Saturación de la posición con el margen de debajo***/
if(Posicion_Bola_y<=(ancho_margen+radio)){
                        Posicion_Bola_y=(ancho_margen+radio);
if(vel_y!=0){
                            vel_y=-(vel_y);
if(Prev_pos_y!=Posicion_Bola_y){
                                TA0CCR0=limite2;
                                TA0CCR1=limite1;
                            }
                        }
                    }
/***Saturación posición con el margen de arriba***/
elseif(Posicion_Bola_y>=127-(ancho_margen+radio)){
                        Posicion_Bola_y=127-(ancho_margen+radio);
if(vel_y!=0){
                            vel_y=-(vel_y);
if(Prev_pos_y!=Posicion_Bola_y){
                                TA0CCR0=limite2;
                                TA0CCR1=limite1;
                            }
                        }
                    }

/**MUERTE EN AGUJERO**/
/*Se comprueba si coincide con la bola para cada posición*/
for(k=0;k<zmax;k++){
if ((Posicion_Bola_x>(agujeros[k][0]-radio) &&
Posicion_Bola_x<agujeros[k][0]+radio) &&
(Posicion_Bola_y<agujeros[k][1]+radio && Posicion_Bola_y>agujeros[k][1]-
radio))
                        {
                            muerte=1; //Variable auxiliar que se pone a '1'
                            kmuerte=k;
//Se establece kmuerte, que servirá para la animación de fin de partida
                            k=zmax;
//Una vez la bola ha muerto, se finaliza el bucle
}
```

```
//Se pintan de Nuevo los agujeros para evitar que la bola los borre al pasar
cerca
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
Graphics_fillCircle(&g_sContext,agujeros[k][0],agujeros[k][1],radio);
}

/**SATURACIÓN DE LA POSICIÓN CON LOS MUROS*****/
//Para ello haremos uso de la matriz 'muros', se comprueba para cada posición
de la bola
for(k=0;k<kmax;k++){
//Choque por el lateral derecho del muro
if ((Posicion_Bola_x-(largo+radio)<=muros[k][2] &&
Posicion_Bola_x>muros[k][2]) && (Posicion_Bola_y<=muros[k][1]+ancho &&
Posicion_Bola_y>=muros[k][1]-ancho)){
                            Posicion_Bola_x=muros[k][2]+(largo+radio);
if(vel_x!=0){
                                vel_x=-vel_x;
if(Prev_pos_x!=Posicion_Bola_x){
                                    TA0CCR0=limite2;
                                    TA0CCR1=limite1;
                                }
                            }
                            k=kmax; //Una vez se ha producido un
choque/rebote, se dejan de leer las matrices y se sale del bucle for.
                        }
//Choque por el lateral izquierdo del muro
elseif ((Posicion_Bola_x+(largo+radio)>=muros[k][0] &&
Posicion_Bola_x<muros[k][0]) && (Posicion_Bola_y<=muros[k][1]+ancho &&
Posicion_Bola_y>=muros[k][1]-ancho)){
                            Posicion_Bola_x=muros[k][0]-(largo+radio);
if(vel_x!=0){
                                vel_x=-vel_x;
if(Prev_pos_x!=Posicion_Bola_x){
                                    TA0CCR0=limite2;
                                    TA0CCR1=limite1;
                                }
                            }
                            k=kmax;
                        }
//Choque por el lado inferior del muro
elseif ((Posicion_Bola_y-(ancho+radio)<=muros[k][1] &&
Posicion_Bola_y>muros[k][1]) && (Posicion_Bola_x<=muros[k][2]+largo &&
Posicion_Bola_x>=muros[k][0]-largo)){
                            Posicion_Bola_y=muros[k][1]+(ancho+radio);
if(vel_y!=0){
                                vel_y=-vel_y;
if(Prev_pos_y!=Posicion_Bola_y){
                                    TA0CCR0=limite2;
                                    TA0CCR1=limite1;
                                }
                            }
                            k=kmax;
                        }
//Choque por el lado superior del muro
elseif ((Posicion_Bola_y+(ancho+radio)>=muros[k][1] &&
Posicion_Bola_y<muros[k][1]) && (Posicion_Bola_x<=muros[k][2]+largo &&
Posicion_Bola_x>=muros[k][0]-largo)){
                            Posicion_Bola_y=muros[k][1]-(ancho+radio);
 if(vel_y!=0){
```

```
                                vel_y=-vel_y;
if(Prev_pos_y!=Posicion_Bola_y){
                                    TA0CCR0=limite2;
                                    TA0CCR1=limite1;
                                }
                            }
                            k=kmax;
                        }
                    }

//Se pintan las casillas de salida y meta.

Graphics_Rectangle cuadrado ={
                              16-8,
                              16-8,
                              16+8,
                              16+8
                                                  };
Graphics_Rectangle cuadrado1 ={
                              16-8,
                              127-16-8,
                              16+8,
                              127-16+8
                                                  };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GREEN);
Graphics_fillRectangle(&g_sContext,&cuadrado);
Graphics_fillRectangle(&g_sContext,&cuadrado1);

/*Si la bola se encuentra dentro de la casilla de meta, se pasa al estado
'Pasa de Nivel' si estamos en el nivel 1, y si no, se dirige al estado
'Pantalla de Victoria', ya que sólo se han creado dos niveles.*/

if((Posicion_Bola_x<16+4 && Posicion_Bola_x>16-4) && (Posicion_Bola_y<127-
16+4 && Posicion_Bola_y>127-16-4)){
if(nivel>=1)
                                victoria=1;
else{
                                fin=1;
                    //Se resetea la posición del eje x e y.
                                Posicion_Bola_x=16;
                                Posicion_Bola_y=16;
             //Se iniciará el siguiente nivel con velocidad 0 en ambos ejes.
                                vel_x=0;
                                vel_y=0;
                            }
                        }
/*En caso de que no se produzca ni 'muerte', ni 'victoria', ni 'cambio de
nivel', nos dirigimos al estado 0:'Actualización de la velocidad y dibujo del
nivel'.*/
                    estado=0;

if(muerte==1){ //Si la bola ha muerto, entonces:
                        estado=2;
//Se pasa al estado 2: 'Animación de fin de partida'.
                        muerte=0;   //Se resetea 'muerte'.
                    }
if(fin==1){ //Si se ha superado el primer nivel, entonces:
                        estado=5;  //Se pasa al estado 5: 'Pasa de Nivel'.
                         fin=0;     //Se resetea 'fin'.
```

```
                    }
if(victoria==1){
//Si se han superado ambos niveles, entonces:
                        estado=6;
//Se pasa al estado 6: 'Pantalla de Victoria'.
                        victoria=0; //Se resetea 'victoria'.
                        espera=0;   //Se resetea 'espera'.
                    }
break;

/************ESTADO 2:ANIMACIÓN DE FIN DE PARTIDA************/
case 2:
/*  Se van dibujando cuadrados alrededor del agujero en el que la bola ha
'muerto', mientras que la bolase dirige hacia el centro, trazando primero un
desplazamiento horizontal y luego vertical. Usamos la variable 'espera' para
que la animación del dibujo de los cuadrados negros sea apreciable.*/

for(i=0; i<112; i++){
//Se dibujan los cuadrados alrededor del agujero donde se muere, el cual
conocemos usando 'kmuerte'.
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);

Graphics_Rectangle cuadrado ={
                                agujeros[kmuerte][0]-radio-i,
                                agujeros[kmuerte][1]-radio-i,
                                agujeros[kmuerte][0]+radio+i,
                                agujeros[kmuerte][1]+radio+i
                        };
Graphics_drawRectangle(&g_sContext,&cuadrado);

                    //Desplazamiento hacia el centro como e ha comentado
if(Posicion_Bola_x > 64)
                            Posicion_Bola_x--;
elseif(Posicion_Bola_x <64)
Posicion_Bola_x++;

elseif(Posicion_Bola_y > 64)
                            Posicion_Bola_y--;
elseif(Posicion_Bola_y <64)
                            Posicion_Bola_y++;
if(Posicion_Bola_x!=64 || Posicion_Bola_y!=64){
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
Graphics_fillCircle(&g_sContext,Prev_pos_x,Prev_pos_y,4);
                        }

Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_WHITE_SMOKE);
Graphics_fillCircle(&g_sContext,Posicion_Bola_x,Posicion_Bola_y,4);

                        Prev_pos_x=Posicion_Bola_x;
                        Prev_pos_y=Posicion_Bola_y;

while(espera<9999)
                            espera++;
                        espera=0;
                    }

/*Se asigna a la matriz auxiliar 'mat' la matriz 'mat1' (nivel 1), para que
una vez se reinicie el juego, lo haga desde el primer nivel.*/
 for(i=0; i<7; i++){
```

```
for(j=0; j<7; j++){
                            mat[i][j]=mat1[i][j];
}
                    }


                    estado=4; //Nos dirigimos al estado 4: 'Pantalla de Fin'.
break;

/****************ESTADO 3:PANTALLA DE INICIO******************/
case 3:
/*Se escribe en el instante 't=0' (y 'espera'=0) el título del juego.
Aumentando la variable 't' se consigue que el texto "Press Start" parpadee,
dibujándolo cuando t=9, y si no, se dibuja un cuadrado del color del
fondo que tape las letras. Cuando t=19, se resetea a t=0. */
if(t==0){
if(espera==0){
Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
Graphics_clearDisplay(&g_sContext);
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
Graphics_drawString(&g_sContext,
"The Ultimate",
                                    15,
                                    15,
                                    20,
                                    TRANSPARENT_TEXT);

Graphics_drawString(&g_sContext,
"Labyrinth",
                                   15,
                                   30,
                                   50,
                                   TRANSPARENT_TEXT);
                            }

Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
Graphics_Rectangle square ={
                                      10,
                                      70,
                                      110,
                                      100
                            };
Graphics_fillRectangle(&g_sContext,&square);
                        }
elseif(t==9){
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLUE);
Graphics_drawString(&g_sContext,
"Press Start",
                                15,
                                20,
                                80,
                                TRANSPARENT_TEXT);
                        }
                        t++;
                        espera=1;
if(t==19)
t=0;
```

```
/*Si la variable 'pulso' está a cero, se espera que se pulse uno de los dos
botonespara proceder al cambio de estado. Esta variable 'pulso' cambia en la
interrupción del timer TA1CCR0, y si su valor es cero indica que no se
encuentra ningún botón pulsado con anterioridad.*/
if(pulso==0){
if(!(P1IN&BIT1) || !(P1IN&BIT2)){
estado=0;
//Se pasa al estado 0: 'Actualización de la velocidad y dibujo del nivel'.
                            ini=1;
//Se establece 'ini' a 1 para pintar el nuevo nivel.
                            t=0;        //Se resetea 't'y 'espera'.
                            espera=0;
                        }
                        }
break;


/*********************ESTADO 4:PANTALLA DE FIN******************/
case 4:
/*El proceso para conseguir el parpadeo de las palabras 'Game Over' es el
mismo que en el estado 3.*/
if(t==0){ //Se dibujan los dos cuadrados negros para tapar las letras y
simular el parpadeo.
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
Graphics_Rectangle square ={
                                    35,
                                    20,
                                    85,
                                    50
                            };
Graphics_fillRectangle(&g_sContext,&square);
Graphics_Rectangle square2 ={
                                    35,
                                    90,
                                    85,
                                    110
                            };
Graphics_fillRectangle(&g_sContext,&square2);

Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_WHITE_SMOKE);
Graphics_fillCircle(&g_sContext,64,64,4);
                    }
elseif(t>9){ //Se dibujan las palabras
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_RED);
Graphics_drawString(&g_sContext,
"GAME",
                            15,
                            38,
                            20,
                            TRANSPARENT_TEXT);

Graphics_drawString(&g_sContext,
"OVER",
                            15,
                            42,
                            90,
                            TRANSPARENT_TEXT);

                    }
```

```
                    t++;
if(t==19)
t=0;
/*Si se pulsa uno de los dos botones, se vuelve al estado 3: 'Pantalla de
Inicio.*/
if(!(P1IN&BIT1) || !(P1IN&BIT2)){
estado=3;
                        t=0;    //Se establece 't' a cero y 'pulso' a 1.
                        pulso=1;
                        Posicion_Bola_x=Pos_ini_x;
//Se coloca la bola en su posición inicial (x e y).
                        Posicion_Bola_y=Pos_ini_y;
                        Prev_pos_x=Posicion_Bola_x;
                        Prev_pos_y=Posicion_Bola_y;
                        vel_x=0; //Se resetean las velocidades.
                        vel_y=0;
                        k=0;    //Se resetean los índices 'k' y 'z', que
servirán para escribir las matrices 'muros' y 'agujeros' respectivamente.
                        z=0;
                        nivel=0;
//Se resetea'nivel' para comenzar en el primer nivel.
Graphics_clearDisplay(&g_sContext);
//Se limpia la pantalla.
                    }
break;

/*********************ESTADO 5:PASA DE NIVEL**********************/
case 5:
/*Mediante el bucle for con la variable 'l', se va dibujando el nivel 1,
restándole el valor de 'l' a las componentes verticales,desplazando así el
mapa del nivel 1 hacia arriba. Para el nivel 2 se realiza de la misma manera,
sólo que el mapa de éste comienza 128 píxeles debajo del extremo inferior de
la pantalla. De este modo, el mapa del nivel 1 va desapareciendo a la par que
el mapa del segundoaparece por debajo.*/
for(l=0;l<=128;l+=5){
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLUE);
for(i=0;i<ancho_margen;i++){
Graphics_Rectangle marco = {
                                    0+i,
                                    0+i-l,
                                    127-i,
                                    127-i-l
                                };
Graphics_drawRectangle(&g_sContext,&marco);
                        }
for(i=0;i<ancho_margen;i++){
Graphics_Rectangle marco = {
                                    0+i,
                                    0+i+127-l,
                                    127-i,
                                    127-i+127-l
                                };
Graphics_drawRectangle(&g_sContext,&marco);
                        }

for (i=0; i<7; i++){
for(j=0; j<7; j++){
if((i==0 && j==0) || (i==6 && j==0)){
 Graphics_Rectangle cuadrado ={
```

```
                                                    (j+1)*16-8,
                                                    (i+1)*16-8-l,
                                                    (j+1)*16+8,
                                                    (i+1)*16+8-l
                                                                    };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GREEN);
Graphics_fillRectangle(&g_sContext,&cuadrado);
                                            }
elseif(mat[i][j]==0){
Graphics_Rectangle cuadrado ={
                                                    (j+1)*16-8,
                                                    (i+1)*16-8-l,
                                                    (j+1)*16+8,
                                                    (i+1)*16+8-l
                                                                    };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GOLDENROD);
Graphics_fillRectangle(&g_sContext,&cuadrado);
                                            }
elseif(mat[i][j]==1){
Graphics_Rectangle cuadrado ={
                                                       (j+1)*16-8,
                                                       (i+1)*16-8-l,
                                                       (j+1)*16+8,
                                                       (i+1)*16+8-l
                                                };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GOLDENROD);
Graphics_fillRectangle(&g_sContext,&cuadrado);

Graphics_Rectangle cuadrado1 ={
                                                    (j+1)*16-(largo-1),
                                                    (i+1)*16-(ancho-1)-l,
                                                    (j+1)*16+(largo-2),
(i+1)*16+(ancho-2)-l
                                                                       };

Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BROWN);
Graphics_fillRectangle(&g_sContext,&cuadrado1);
                                            }
elseif(mat[i][j]==2){
Graphics_Rectangle cuadrado ={
                                                    (j+1)*16-8,
                                                    (i+1)*16-8-l,
                                                    (j+1)*16+8,
                                                    (i+1)*16+8-l
                                                                    };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GOLDENROD);
Graphics_fillRectangle(&g_sContext,&cuadrado);
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
Graphics_fillCircle(&g_sContext,(j+1)*16,(i+1)*16-l,4);
                                            }

if((i==0 && j==0) || (i==6 && j==0)){
Graphics_Rectangle cuadrado ={
                                                    (j+1)*16-8,
                                                    (i+1)*16-8+127-l,
                                                    (j+1)*16+8,
                                                    (i+1)*16+8+127-l
                                                                    };
 Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GREEN);
```

```
Graphics_fillRectangle(&g_sContext,&cuadrado);
                                        }
elseif(mat2[i][j]==0){
Graphics_Rectangle cuadrado ={
                                                    (j+1)*16-8,
                                                    (i+1)*16-8+127-l,
                                                    (j+1)*16+8,
                                                    (i+1)*16+8+127-l
                                                                    };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GOLDENROD);
Graphics_fillRectangle(&g_sContext,&cuadrado);
                                        }
elseif(mat2[i][j]==1){
Graphics_Rectangle cuadrado ={
                                                       (j+1)*16-8,
                                                       (i+1)*16-8+127-l,
                                                       (j+1)*16+8,
                                                       (i+1)*16+8+127-l
                                                };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GOLDENROD);
Graphics_fillRectangle(&g_sContext,&cuadrado);

Graphics_Rectangle cuadrado1 ={
                                                   (j+1)*16-(largo-1),
                                                   (i+1)*16-(ancho-1)+127-l,
                                                   (j+1)*16+(largo-2),
(i+1)*16+(ancho-2)+127-l
                                                                       };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BROWN);
Graphics_fillRectangle(&g_sContext,&cuadrado1);
                                        }
elseif(mat2[i][j]==2){
Graphics_Rectangle cuadrado ={
                                                    (j+1)*16-8,
                                                    (i+1)*16-8+127-l,
                                                    (j+1)*16+8,
                                                    (i+1)*16+8+127-l
                                                                    };
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GOLDENROD);
Graphics_fillRectangle(&g_sContext,&cuadrado);
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_BLACK);
Graphics_fillCircle(&g_sContext,(j+1)*16,(i+1)*16+127-l,4);
}
                                    }
                                }
                    }

                    estado=0;
//Se pasa al estado 0: 'Actualización de la velocidad y dibujo del nivel'.
                    ini=1;
//Establecemos 'ini' a 1 para volver a pintar el nivel.
                    k=0;    //Reseteamos los índices k y z.
                    z=0;
                    nivel++; //Aumentamos el nivel.

/*Se asigna a la matriz auxiliar 'mat' la matriz 'mat2' (nivel 2), para que
ahora las comprobaciones de los choques y rebotes sean con los elementos
del nivel 2.*/
 for(i=0; i<7; i++){
```

```
for(j=0; j<7; j++){
                            mat[i][j]=mat2[i][j];
}
                    }
break;

/************************ESTADO 6:PANTALLA DE VICTORIA*****************/
case 6:
//Se dibuja el texto durante unos segundos.
if(espera==0){
Graphics_setBackgroundColor(&g_sContext, GRAPHICS_COLOR_WHITE);
Graphics_clearDisplay(&g_sContext);
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_GREEN);
Graphics_drawString(&g_sContext,
"YOU WIN!",
15,
                                25,
                                50,
                                TRANSPARENT_TEXT);
                    }
                    espera++;
if(espera==59){
//Transcurridos unos segundos, actualizamos la matriz auxiliar 'mat' para
comenzar con el nivel 1.
for(i=0; i<7; i++){
for(j=0; j<7; j++){
                                mat[i][j]=mat1[i][j];
}
                        }
                        estado=3;
//Nos dirigimos al estado 3: 'Pantalla de Inicio'.
                        espera=0;
                        t=0;
                        pulso=1;
                        Posicion_Bola_x=Pos_ini_x;
//Se coloca la bola en su posición inicial ("x" e "y").
                        Posicion_Bola_y=Pos_ini_y;
                        Prev_pos_x=Posicion_Bola_x;
                        Prev_pos_y=Posicion_Bola_y;
                        vel_x=0;
//Se iniciará el siguiente nivel con velocidad 0 en ambos ejes.
                        vel_y=0;
                        k=0;
                        z=0;
                        nivel=0;
//Al comenzar a jugar de nuevo, el nivel será el primero.
Graphics_clearDisplay(&g_sContext);
                    }
break;
}
/*Se dibuja la bola sólo en los estados 0 ('Actualización de la velocidad y
dibujo del nivel)y 1 ('Actualización de la posición'), y se borra, pintando
del color del fondo la anterior posición de la bola.*/

if(estado==0 || estado==1){

Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_DARK_GOLDENROD);
Graphics_fillCircle(&g_sContext,Prev_pos_x,Prev_pos_y,radio);
```

```
Graphics_setForegroundColor(&g_sContext, GRAPHICS_COLOR_WHITE);
Graphics_fillCircle(&g_sContext,Posicion_Bola_x,Posicion_Bola_y,radio);

            Prev_pos_x=Posicion_Bola_x;
            Prev_pos_y=Posicion_Bola_y;
            }
    }
    }

#pragma vector=ADC10_VECTOR
__interruptvoidConvertidorAD(void)
    {
        LPM0_EXIT;  //Despierta al micro al final de la conversión
    }

#pragma vector=TIMER1_A0_VECTOR
__interruptvoidTIMER1_A0_ISR_HOOK(void)
    {
        LPM0_EXIT;  //Despierta al micro al final del tiempo

if(P1IN&BIT1 && P1IN&BIT2 && t==1)
//Si no se encuentra ninguno de los botones pulsados, establecemos 'pulso' a
cero.
            pulso=0;
          }
```

7. Bibliografía

http://www.ti.com/lit/ds/symlink/msp430g2353.pdf

http://www.ti.com/lit/ug/slau599a/slau599a.pdf

Apuntes de clase y prácticas.