Bomberman modo multijugador

Hristo Ivanov[*] , Alberto Lorente[+], Verónica Chamorro[+], Alberto A. Del Barrio[1+], Guillermo Botella[1+]

[1] Departamento de Arquitectura de Computadores y Automática, Facultad de Informática de la Universidad Complutense de Madrid
Madrid, España

* hristo.idgh@gmail.com,+ {albertol, verocham, abarriog, gbotella}@ucm.es

Resumen. Este trabajo presenta el proyecto Bomberman, realizado en la asignatura Sistemas Empotrados Distribuidos, perteneciente a la titulación del Máster en Ingeniería Informática de la Universidad Complutense de Madrid. En este trabajo se describe e implementa una adaptación del conocido juego Bomberman en modo multijugador (dos jugadores). En esta versión los dos jugadores tratarán de salir de un laberinto o derrotar a su contrincante para ganar. Este proyecto utiliza dos placas de desarrollo S3CEV40 representado a cada jugador, una Raspberry Pi 2, dos cables hembra-hembra de 9 pines y dos adaptadores a 9 pines-USB para conectar cada cable desde cada placa S3CEV40 a la Raspberry.

Palabras Clave: Sistemas empotrados distribuidos, raspberry. ARM, Sprites.

Abstract. This paper presents the Bomberman project, carried out in the Distributed Embedded Systems subject, which belongs to the Computer Science Master that is taught at the Complutense University of Madrid. This work describes and implements an adaptation of the well-known Bomberman game in multiplayer mode (for two players). In this version, the two players will try to escape from a labyrinth or to destroy his opponent to win. This project use two S3CEV40 boards to represent the players, a Raspberry Pi 2, two female-to-female 9 pin cables and two 9 pins-to-USB adapters to connect each board to the Raspberry.

Keywords: Distributed embedded systems, Raspberry, S3CEV40, ARM, Sprites.

1 Introducción

La asignatura Sistemas Empotrados Distribuidos (SED) es una asignatura que aparece en el plan del Máster de Ingeniería Informática de la Universidad Complutense de Madrid (UCM) [1], implantado durante el curso 2013/2014. La idea fundamental del máster es proporcionar conocimientos adicionales a los estudiantes que les haga más competitivos en el mundo laboral. Con el objetivo de conseguir estas competencias las asignaturas del máster, y en concreto SED, contienen una alta carga práctica, de tal forma que los alumnos puedan enfrentarse a problemas desde una perspectiva más allá de la teórica. Por ello, además de las sesiones de laboratorio, la asignatura cuenta con un proyecto final [9-11] en el que los estudiantes demuestran la adquisición de conocimientos en el área de los Sistemas Empotrados Distribuidos, también conocida como *Computación Ubícua* o *Pervasive Computing* [6-8,17].

La importancia de los sistemas distribuidos puede observarse en distintos ámbitos de la Informática actualmente: desde el diseño de circuitos [13-15] hasta el *Wearable Computing* [12,16] o el *Internet of Things* (IoT) [18-19]. Es por ello fundamental que los alumnos aprendan y experimenten con la problemática que tales sistemas presentan.

El proyecto descrito en este artículo consiste en una adaptación del conocido juego Bomberman en modo multijugador sobre la placa de desarrollo S3CEV40 [3-5], la cual se utiliza también en las prácticas de SED.

El resto del artículo se organiza de la siguiente manera: la Sección 2 especifica el proyecto; en las Secciones 3 y 4 se presentan los aspectos generales del hardware y el diseño e implementación del proyecto, respectivamente; mientras que en las Secciones 5 y 6 se detallan aspectos más concretos sobre el envío de mensajes y la lógica interna del juego; y en la Sección 7 se presentan algunas de las pruebas realizadas. Finalmente, en la Sección 8 se resumen las conclusiones y describen las posibles líneas de trabajo futuro.

2 Especificación del juego

La versión de Bomberman que ha sido implementada, trata de dos personajes intentado salir de un laberinto, o bien derrotando a su contrincante haciendo que un elemento llamado *bomba* explote en las proximidades del jugador. Dicho laberinto está compuesto de diferentes elementos; *paredes* (elementos indestructibles) y por *rocas fracturadas* (elementos destructibles). Cada jugador es capaz de eliminar estas rocas con las mencionadas bombas. Cuando el jugador así lo desee podrá colocar una bomba en la posición en la que se encuentre. Tras un cierto tiempo la bomba detonará rompiendo aquellas rocas fracturadas que tenga a su alrededor. Además, si cualquiera

de los jugadores se encontrase en el rango de explosión de la bomba, este jugador será derrotado.

Para ganar el juego, el jugador deberá romper las rocas necesarias hasta encontrar la salida oculta por las mismas. Una vez visible la salida tan solo deberá caminar hasta salir por ella para ganar el juego. Idénticamente un jugador ganará la partida automáticamente si su contrincante muere a consecuencia de una bomba.

Para añadir más emoción al juego el mapa se generará de forma aleatoria en cada partida. De esta forma cada partida contará con una nueva distribución de las rocas fracturadas, la salida y la colocación de los dos jugadores. Esta colocación de los jugadores siempre deberá permitir a los mismos comenzar la partida sin estar bloqueados entre rocas destructibles, es decir, permitirá al jugador colocar bombas sin verse afectadas por ellas.

Un comienzo correcto de la partida sería pues cuando un jugador tiene al menos 2 casillas libres a su alrededor. Esto le permitiría colocar en una la bomba y huir a la otra para no verse afectada por la misma.

Dado que se trata de un juego, el objetivo es que ambos jugadores tengan una experiencia de juego en “tiempo real” por parte del sistema. Esta experiencia deberá compaginarse con el hecho de que no habrá un componente central con toda la información del juego.

3 Hardware empleado

El proyecto se desarrolla utilizando dos placas de desarrollo S3CEV40 para cada jugador y una Raspberry Pi 2. Una representación general del sistema empleado sería la que muestra la Figura 1.

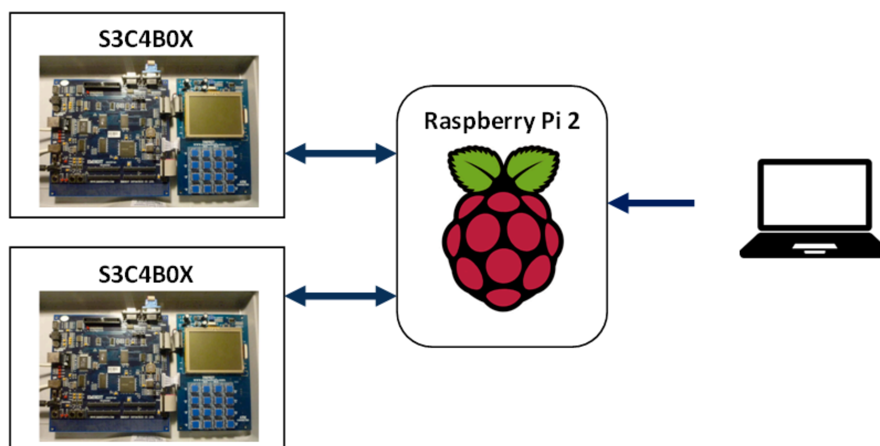

Figura 1. Esquema general del sistema

A su vez, cada elemento sería como los mostrados en las Figuras 2 y 3. Además, los conectores mostrados en la Figura 4 son necesarios para que las placas de desarrollo puedan comunicarse entre sí.

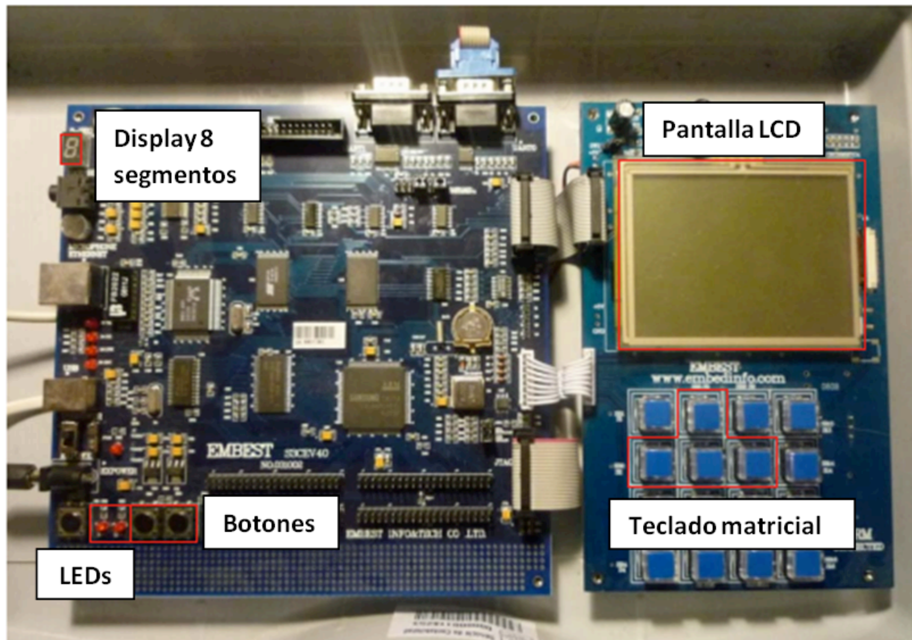

Figura 2. Placa de desarrollo S3CEV40 (*maletín*)

Figura 3. Raspberry Pi 2 que se utiliza como puente entre los dos maletines

Figura 4a. Dos cables hembra-hembra, de 9 pines.

Figura4b. Dos adaptadores a 9pin-USB para conectar los cables de la figura 4a a través de la UART de los maletines a la Raspberry.

La placa de desarrollo S3CV40 es un System on Chip (SoC) de Samsung basada en un procesador ARMTDMI. Dicha placa contiene varios periféricos, como puede observarse en la Figura 2, y contiene el microcontrolador S3C44B0X, que será el encargado de gestionar las interrupciones producidas por los periféricos.

Además, será necesario un equipo que funcione como host y desde el cual descargar el juego a los maletines y para conectarse a la Raspberry a través de SSH. Esto es así para poder ejecutar el código en python de la Raspberry.

4 Diseño e implementación

Dado que el componente central de la asignatura es la creación de sistemas distribuidos, el diseño del videojuego se centró principalmente en varias placas de desarrollo S3CEV40. Cada una tendrá una *copia* del juego de su jugador correspondiente. No habrá, por tanto, un sistema central en donde se almacene todo el estado del juego y sobre el cual los jugadores realicen acciones. Es decir, cada maletín es independiente y tan solo tiene un canal de entrada/salida de información a través de la UART.

Además, para mejorar la experiencia de juego se realizarán unos *sprites* en 2D que representarán a los jugadores, paredes, rocas fracturadas y salida. Para dar la sensación de movimiento, los jugadores tendrán varios *sprites* distintos.

Con el objetivo de comprender el funcionamiento del juego, se explican a continuación los elementos utilizados así como su función:

4.1 Botones

Están configurados mediante interrupciones y tienen 2 funcionalidades:

1. Inicializar el juego. El juego no comenzará hasta que los dos maletines hayan pulsado un botón para indicar que están preparados.

2. Colocar bombas. Cada jugador dispondrá de una bomba que se colocará cuando éste pulse un botón. Esta bomba se ubicará en la posición donde esté en ese momento el jugador. Concretamente, la bomba se depositará en la casilla donde esté situada la mayor parte del personaje.

4.2 LEDs

Los LEDs están configurados para informar visualmente al jugador del estado de la bomba.

- Si la bomba no está colocada los LEDs permanecen encendidos.
- Si la bomba está colocada los leds parpadean. Una vez explota la bomba vuelven a parpadear.

4.3 Teclado matricial

El teclado matricial no está configurado ni por interrupciones ni por espera activa. La forma de conocer qué tecla ha pulsado el jugador es mediante el uso de un temporizador. Cuando este temporizador llega a 0 realiza la lectura del registro correspondiente para conocer así cual ha sido la última pulsación del jugador. Este método es utilizado para mantener una sincronización de juego entre los dos maletines y para que un jugador no pueda realizar más acciones que otro, ya que ambos tienen la misma configuración de su temporizador.

Así el teclado matricial indica la dirección hacia la que se quiere mover el jugador. Siguiendo el estilo de las flechas de dirección de un teclado, las teclas del juego serían las que se muestran en la Figura 5.

Tecla 1 → Arriba
Tecla 4 → Izquierda
Tecla 5 → Abajo
Tecla 6 → Derecha

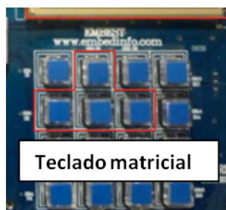

Figura 5. Teclado matricial para los movimientos

Como se puede ver en la Figura 5 las únicas teclas tienen efecto sobre el juego son las teclas 1, 4, 5 y 6. Se pueden entender como las flechas de dirección de un teclado convencional.

4.4 Display de 8 segmentos

Dicho display se utiliza a modo informativo para representar visualmente el último movimiento realizado por el jugador a través del teclado matricial.

4.5 Temporizadores o *timers*

En la implementación del juego se han utilizado dos timers distintos, cada uno configurado de forma idéntica en cada maletín.

Timer 0

Es el timer de juego principal. Configurado en modo *autoreload*, se utiliza para mantener la sincronización del juego. Este *timer* impide al jugador realizar más movimientos de los permitidos en un espacio de tiempo definido. Cada vez que este *timer* se active se encargará de leer la tecla pulsada por el jugador en el teclado matricial. Según sea la pulsación del jugador con la tecla 1, 4, 5 o 6 moverá el jugador en su sentido correspondiente. En otras palabras, cada vez que se active el *timer* 0 el jugador podrá llevar a cabo una única acción de movimiento. Además, mostrará en el display de 8 segmentos la tecla pulsada.

Timer 2

Este timer se encargará de gestionar las bombas. Cada vez que un jugador coloque una bomba pulsando un botón, éste activará el timer 2 en modo *manual update*. Cuando el timer llegue a 0 la bomba explotará, liberando así el camino de rocas fragmentadas o incluso pudiendo matar a los distintos jugadores.

4.6 Pantalla LCD

Se utiliza para mostrar todo el desarrollo del juego. Se encargará de dibujar el escenario con todas sus paredes y rocas fragmentadas, así como los dos jugadores y las bombas.

La pantalla LCD funciona mostrando el contenido de un array de enteros sin signo de 8 bits (*uint8*). Este array (llamado buffer en el código) tiene un tamaño de 320 píxeles de ancho por 240 de alto. Además, dentro de cada posición del array (8 bits) se representan dos píxeles distintos. Los 4 bits de las posiciones superiores representan el color del píxel izquierdo y los 4 bits de posiciones inferiores representan el color del píxel derecho.

1111,1110,1101,1100,

1011,1010,1001,1000,

0111,0110,0101,0100,

0011,0010,0001,0000

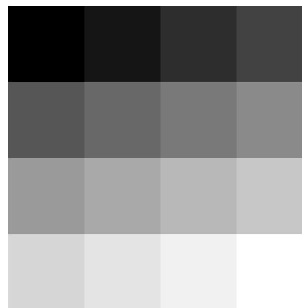

Figura 6. Codificación de escala de grises en la pantalla y su valor en binario.

Esto nos deja una profundidad de color en cada píxel de 4 bits. Con esta codificación podemos representar una escala de colores de gris (gray-scale) como la que puede verse en la Figura 6.

Para trabajar más fácilmente con el LCD, se ha dividido la pantalla de 320x240 píxeles en casillas de 16x16 dando como resultado una pantalla de 20x15 casillas lógicas de juego. Aprovechando estas casillas se pintarán los distintos componentes del juego: personajes, bombas, rocas, etc. Además, se han creado distintos sprites para pintar en estas casillas con el fin de mejorar la experiencia de juego.

La función visual de un juego de estas características es muy importante, por lo que para pintar el LCD no se recorre todo el array cada vez que se desea cambiar la visualización de la pantalla. Por el contrario, se utilizan refrescos locales, es decir, no se modifica el contenido al completo del array. Tan solo se refrescan casillas lógicas de 16x16 mencionadas en el apartado anterior. Para ello se utilizan las funciones *lcd_draw16x16* y *lcd_clear16x16* capaces de pintar los sprites y borrarlos más rápidamente, evitando aquellos píxeles que ya estén en blanco.

4.7 UART

La UART, o Universal Asynchronous Reciver-Transmitter, es el elemento utilizado para la comunicación entre las dos placas de desarrollo S3CEV40.

En nuestro proyecto se han configurado las dos UARTs presentes en la placa (UART0 y UART1). Sin embargo, para la comunicación específica entre los 2 maletines únicamente se utiliza la UART1. Ambas UARTs han sido configuradas mediante interrupciones en modo lectura (*Rx*). Además, se ha activado la cola FIFO (*Rx*) presente en la UART para evitar perder mensajes enviados de una placa a otra. El tratamiento de mensajes se hace a nivel de byte.

Por último, se ha definido un protocolo propio de transmisión en el cual se define cómo son los mensajes que se envían los jugadores para actualizar sus escenarios, es decir, la pantalla LCD. La Sección 5, especificará en profundidad dicho protocolo.

4.8 Raspberry Pi 2

Entre las dos placas S3CEV40, conectada por los cables hembra-hembra y los adaptadores, se coloca la Raspberry. Su función es únicamente reenviar los mensajes que le llegan a través de un puerto serie por el otro puerto serie.

El único código utilizado en la Raspberry tiene por finalidad la creación de dos puertos serie (*io1* e *io2*) y la creación de dos threads (*tr1* y *tr2*), los cuales serán los encargados de leer de un puerto serie el mensaje recibido, mostrarlo y reenviarlo por el otro puerto serie.

La Raspberry se ha utilizado fundamentalmente para mitigar los problemas de potencia de las UARTs de las placas S3CEV40. Sin la Raspberry se perdían o malinterpretaban varios mensajes tras cierto tiempo de juego, con lo que concluimos que la potencia de señal transmitida que transmiten las placas a través de sus UART no es lo suficientemente elevada como para que la otra placa detecte el mensaje transmitido con claridad. Además, el uso de esta placa se pensó como posible solución para escalar el juego a N jugadores, ya que con 2 jugadores los maletines consumían casi todos sus recursos con la lógica del juego y el refresco.

4.9 Sprites

Los sprites son conocidos en el mundo de los videojuegos como mapas de bits dibujados generalmente por hardware y utilizados para dotar al videojuego de un atractivo adicional.

Para nuestro proyecto se han generado distintos sprites para los distintos objetos presentes en la escena. Estos sprites tienen un tamaño de 16x16 píxeles y cada píxel o posición de esa matriz tiene un valor en [0,15] o en binario [0000,1111].

Figura 7. Sprites correspondientes al estado *Idle* (0), y el *movimiento* (2 y 3) del jugador.

Para representar el movimiento del jugador alternamos la presentación de los sprites en secuencia 0-1-0-2-0 según están representados en la Figura 7. Esto genera una sensación de movimiento al avanzar el personaje. Para distinguir un segundo jugador se han retirado los cuernos al sprite de la Figura 7. De esta forma obtenemos dos jugadores *demon* y *human*.

Una vez seleccionada la imagen generamos la matriz gracias a un programa externo creado por nosotros mismos. Este programa leerá el color establecido en cada píxel haciendo una conversión a la tonalidad de gris correspondiente. Además creará una matriz como la de la Figura 8 que copiaremos dentro de nuestro código fuente.

```
0b1110,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b0000,0b1110,
0b1110,0b1110,0b0000,0b0000,0b0000,0b1101,0b1101,0b1101,0b1101,0b1101,0b1101,0b0000,0b0000,0b0000,0b1110,0b1110,
0b1110,0b1110,0b1110,0b1110,0b1101,0b0011,0b0011,0b0011,0b0011,0b0011,0b0011,0b1101,0b1110,0b1110,0b1110,0b1110,
0b0000,0b1110,0b1110,0b1111,0b0011,0b0010,0b0001,0b0010,0b0010,0b0001,0b0010,0b0011,0b1111,0b1110,0b1110,0b0000,
0b0000,0b0000,0b1111,0b1111,0b0011,0b0001,0b1110,0b0001,0b0001,0b1110,0b0001,0b0011,0b1111,0b1111,0b0000,0b0000,
0b0000,0b0000,0b0000,0b1101,0b0100,0b0001,0b1110,0b0001,0b0001,0b1110,0b0001,0b0100,0b1101,0b0000,0b0000,0b0000,
0b0000,0b0000,0b0000,0b1101,0b1011,0b0010,0b0001,0b0010,0b0010,0b0001,0b0010,0b1011,0b1101,0b0000,0b0000,0b0000,
0b0000,0b0000,0b0000,0b0000,0b1101,0b1011,0b1011,0b1011,0b1011,0b1011,0b1011,0b1101,0b0000,0b0000,0b0000,0b0000,
0b0000,0b0000,0b0000,0b1101,0b0011,0b1101,0b1101,0b1101,0b1101,0b1101,0b1101,0b0011,0b1101,0b0000,0b0000,0b0000,
0b0000,0b0000,0b1101,0b0011,0b1011,0b0100,0b0011,0b1101,0b1101,0b0011,0b0100,0b1011,0b0011,0b1101,0b0000,0b0000,
0b0000,0b1101,0b0011,0b1011,0b1101,0b1011,0b0100,0b0100,0b0100,0b0100,0b1011,0b1101,0b1011,0b0011,0b1101,0b0000,
0b0000,0b1101,0b0100,0b0100,0b1101,0b1011,0b1011,0b1011,0b1011,0b1011,0b1011,0b1101,0b0100,0b0100,0b1101,0b0000,
0b0000,0b0000,0b1101,0b1101,0b1101,0b1100,0b1100,0b1100,0b1100,0b1100,0b1100,0b1101,0b1101,0b1101,0b0000,0b0000,
0b0000,0b0000,0b0000,0b0000,0b0000,0b1101,0b0101,0b1101,0b1101,0b0101,0b1101,0b0000,0b0000,0b0000,0b0000,0b0000,
0b0000,0b0000,0b0000,0b0000,0b1101,0b1110,0b1110,0b1101,0b1101,0b1110,0b1110,0b1101,0b0000,0b0000,0b0000,0b0000,
0b0000,0b0000,0b0000,0b0000,0b1101,0b1111,0b1111,0b1101,0b1101,0b1111,0b1111,0b1101,0b0000,0b0000,0b0000,0b0000
```

Figura 8. Matriz de 16x16 elementos enteros en donde cada uno representa el color de un píxel.

0b indica que los siguientes dígitos especifican un número en binario. Para el segundo jugador el sprite es el mismo, pero sin cuernos (Human).

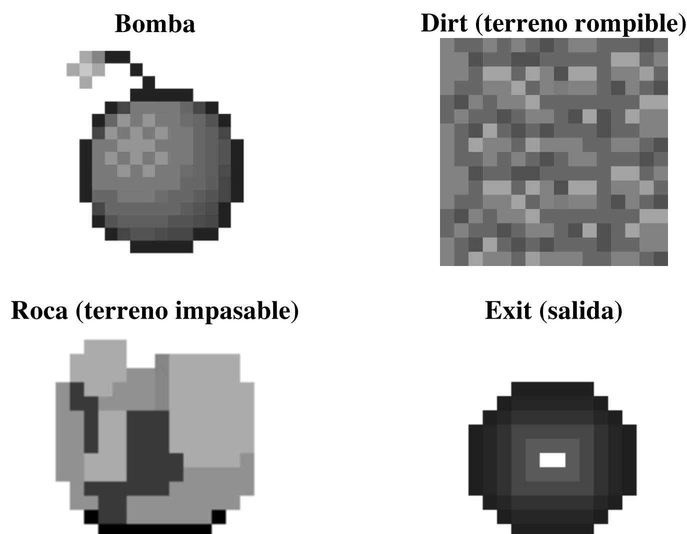

Figura 9. Otros sprites generados para la pantalla LCD.

Las matrices generadas como la de la Figura 8 serán añadidas como arrays en el código, representando todos los sprites de las Figura 7 y 9.

5 Protocolo

Las placas S3CEV40 se comunican por la UART1 enviando bytes de uno a otro para que actualicen el escenario presentado en la pantalla LCD. A continuación, se describen los diferentes mensajes que los jugadores pueden intercambiar.

Tabla 1. Mensaje correspondiente a la posición del jugador.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	0	0	PosX(8)	PosX(7)	PosY(7)	Sprite(1)	Sprite(0)
Byte 2	0	PosX(6)	PosX(5)	PosX(4)	PosX(3)	PosX(2)	PosX(1)	PosX(0)
Byte 3	0	PosY(6)	PosY(5)	PosY(4)	PosY(3)	PosY(2)	PosY(1)	PosY(0)

Dado que la pantalla tiene un tamaño de 320 x 240 píxeles, y el jugador puede estar en cualquiera de estos píxeles, necesitamos 9 bits para la *posición X* y 8 bits para la *posición Y*. Con los dos bits de *Sprite* indicamos el sprite que debe ser utilizado

para pintar el movimiento del jugador [0,2]. Toda esta información aparece resumida en la Tabla 1.

Tabla 2. Mensaje correspondiente a la posición de la bomba.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	0	1	PosX(4)	PosX(3)	PosX(2)	PosX(1)	PosX(0)
Byte 2	0	_	_	_	PosY(3)	PosY(2)	PosY(1)	PosY(0)

Dado que la bomba únicamente puede estar en las casillas de 16x16, tan solo son necesarios 5 bits (320/16 = 20) para la *posición X* y 4 bits (240/16 = 15) para la *posición Y*. Cuando el jugador pulsa un botón, este mensaje es enviado por una placa a la otra para que la segunda pinte en su pantalla LCD la bomba en su correspondiente posición. El contenido de este mensaje se detalla en la Tabla 2.

Tabla 3. Mensaje que contiene la semilla del juego.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	1	0	0	0	0	0	0
Byte 2	0	seed(6)	seed(5)	seed(4)	seed(3)	seed(2)	seed(1)	seed(0)

Para generar el mapa de juego de forma aleatoria se utiliza una semilla de juego. Cuando los dos jugadores pulsan el botón de inicio de juego ambos intercambian la semilla. Como se utilizan 7 bits para la semilla, tenemos 2^7 juegos posibles. La especificación de este mensaje se muestra en la Tabla 3.

Tabla 4. Mensaje de GameOver.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	1	0	0	0	0	0	1

El mensaje especificado en la Tabla 4 indica el fin del juego. Este mensaje se genera cuando el jugador ha muerto al explotar una bomba a su lado, por lo que este jugador ha perdido la partida. Por tanto, el jugador que recibe este mensaje ha ganado.

Tabla 5. Mensaje de GameWin.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	1	0	0	0	0	1	0

El mensaje mostrado en la Tabla 5 indica el fin del juego. Este mensaje se genera cuando el jugador alcanza la salida, por lo que este jugador gana la partida. Por el contrario, el jugador que recibe este mensaje ha perdido.

Tabla 6. Mensaje correspondiente a la explosión de la bomba.

Bit	7	6	5	4	3	2	1	0
Byte 1	1	1	1	PosX(4)	PosX(3)	PosX(2)	PosX(1)	PosX(0)
Byte 2	0	_	_	_	PosY(3)	PosY(2)	PosY(1)	PosY(0)

El mensaje "Explosión Bomba" se muestra en la Tabla 6. Dado que la bomba únicamente puede estar en las casillas de 16x16, tan solo son necesarios 5 bits (320/16 = 20) para la *posición X* y 4 bits (240/16 = 15) para la *posición Y*. Análogamente al mensaje "Posición Bomba" de la placa que controla la bomba será responsable de retransmitir a la otra placa que la bomba ha explotado. Este mensaje se transmitirá cuando el timer 1 llegue a 0 y será el responsable de eliminar la bomba de la pantalla LCD y de procesar si la bomba a afectado al jugador.

6 La lógica del juego

Para procesar los mensajes se utiliza una máquina de estados. El primer bit de cada byte está reservado. Cuando este bit toma el valor '1', indica el principio de un nuevo mensaje (comando), en otro caso este bit vale '0' (dato). Este primer bit nos permite sobrevenir la posible pérdida de bytes que pueda darse. Al recibir un byte con un '1' inicial empezamos la recepción de un nuevo mensaje. Por el contenido de este byte podemos identificar el mensaje (bits 6 y 5). Dependiendo del mensaje, esperamos un número variable de bytes que empiezan por un '0'. Si, por contrario, recibimos un byte con un '1' inicial, la recepción del mensaje actual se descarta y se procede al procesado del nuevo mensaje.
La máquina de estados quedaría entonces representada por la Figura 11.

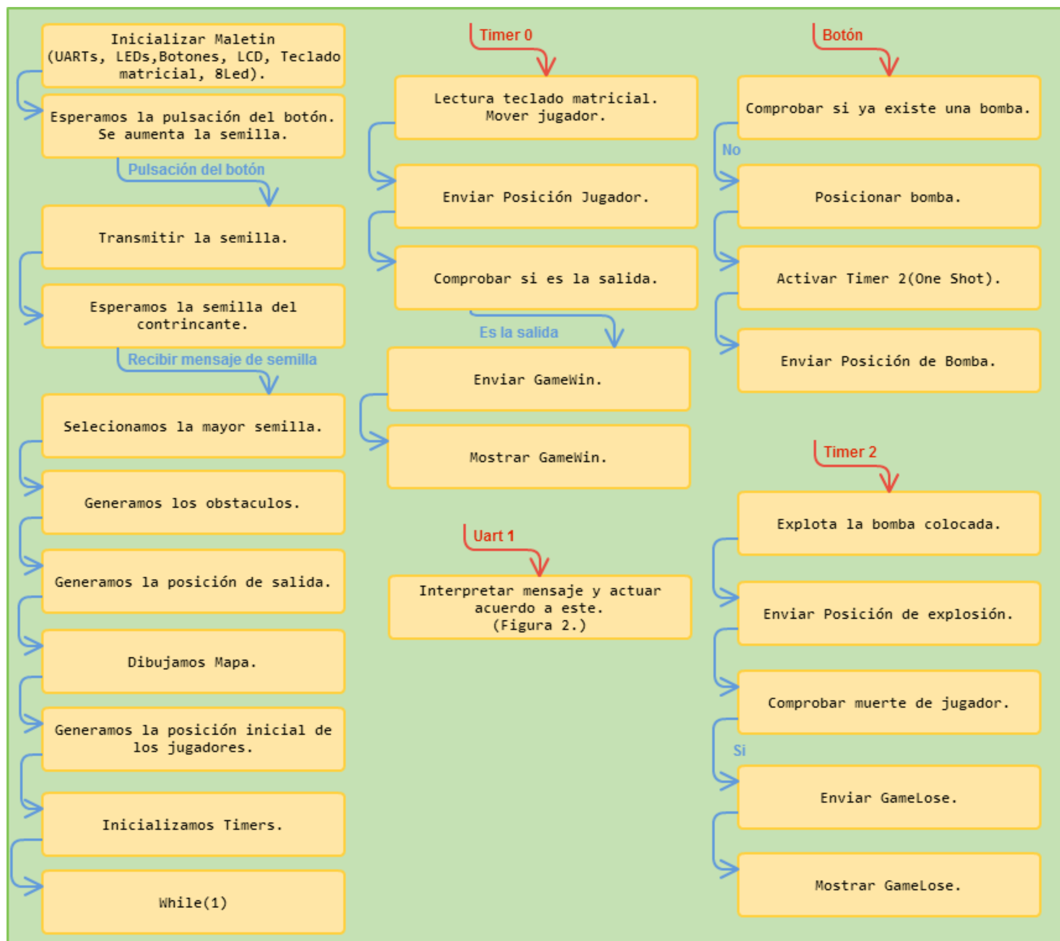

Figura 11. Lógica del juego.

Paralelamente se utiliza la UART para enviar mensajes a la otra placa. Estos mensajes recibirán un tratamiento distinto dependiendo de cuáles sean:

- En caso de recibir un mensaje de semilla la placa entenderá que el otro jugador está listo para empezar el juego. En caso de que ambos lo estén se generará el mapa en base a las semillas transmitidas entre ambos.
- En el caso de recibir un mensaje de "posición jugador" la placa se encargará de mover al jugador contrario en la pantalla LCD.
- En el caso de recibir un mensaje de "colocar bomba" la placa tan solo la pintará en la pantalla LCD.
- En el caso de recibir un mensaje de "explosión bomba" la placa se encargará de eliminar visualmente en la pantalla la bomba y las paredes rompibles adyacentes a la misma. También se encargará de procesar si la explosión de la bomba ha podido afectar al jugador. En el caso de que así sea la placa enviará un mensaje de

"GameOver" indicando que el jugador en su placa ha perdido y el contrario ha sido victorioso.

7 Experimentos

El desarrollo del videojuego y la experimentación han sido dos estados cíclicos en el desarrollo del proyecto. Además esta experimentación nos permitió tomar decisiones importantes de cara al desarrollo como la creación de una maquina de estados en cada placa o la sincronización a través de un timer en el teclado matricial y no por interrupciones.

Sin embargo, los experimentos más importantes y más satisfactorios se produjeron al final del proyecto. Al ser un videojuego el papel que más resalta es la visualización del mismo.

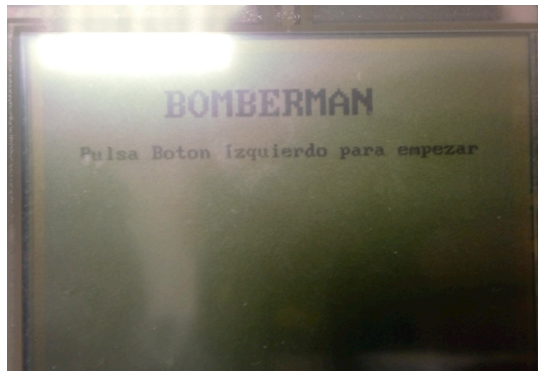

Figura 12. Inicio del videojuego.

En la Figura 12 puede verse cómo se escribe el mensaje inicial en la pantalla LCD, trabajando además con la intensidad del negro en las letras así como con su tamaño.

Figura 13. Juego en proceso.

En la Figura 13 puede observarse cómo se muestra el laberinto del juego durante una partida.

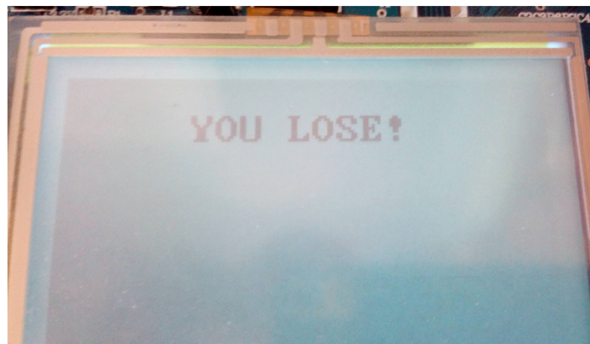

Figura 14. Fin del videojuego con derrota

Finalmente, en la Figura 14 podemos ver el mensaje que mostraría el LCD al jugador que hubiera perdido el juego.

8 Conclusiones

En este artículo se ha presentado una plataforma distribuida y heterogénea para implementar el juego del Bomberman para dos jugadores. Las placas de desarrollo S3CEV40 se han empleado para implementar a cada jugador, mientras que la Raspberry Pi 2 ha funcionado como puente para comunicar ambas placas.

Además de la lógica de juego, se ha desarrollado un protocolo de comunicación para informar a cada jugador sobre el estado del otro y así poder avanzar en la ejecución del juego.

Aunque la versión presentada es plenamente funcional, en el futuro podrían añadírsele las siguientes características:

- Creación de un menú de juego: una vez el juego se finaliza no hay ninguna forma de volver a relanzarlo. Todas las interrupciones y timers se desactivan una vez ha finalizado el juego.
- Sistema de puntuaciones: asignar un valor por cada bloque de tierra eliminado o por el tiempo en salir del laberinto. Para ello se pensó inicialmente en la utilización de la EEPROM como almacén de puntuaciones que pudiera mostrarse también desde el menú de juego.
- Escalar el juego a N jugadores. Habría que evaluar la carga computacional asociada para procesar los mensajes de todos los jugadores, y muy posiblemente descargar parte del control sobre la Raspberry Pi.

Referencias

1. Máster en Ingeniería Informática de la Universidad Complutense de Madrid, http://informatica.ucm.es/estudios/2016-17/master-ingenieriainformatica
2. https://github.com/AlbertoLorente92/master-ucm-SED. Repositorio de la asignatura y del Proyecto final donde está el juego.
3. Embest S3CEV40 EVB User Guide, http://www.vas.co.kr/products/support/S3CEV40_UserGuide.pdf
4. http://datasheets.chipdb.org/Samsung/S3C44B0X.pdf. Manual de la placa S3C44B0X de Samsung.
5. http://www.fdi.ucm.es/profesor/mendias/PSyD/PSyD.html. Manuales para el uso de la placa S3C44B0X y la pantalla LCD.
6. Chtourou, S. et al., "Evolution of robot programming, towards the Ubiquitous Computing era," *Individual and Collective Behaviors in Robotics (ICBR), 2013 International Conference on* , vol., no., pp.44,48, 15-17 Dec. 2013
7. Daoqing Sun, "Researches to the trusted ubiquitous computing," *Electronics, Communications and Control (ICECC), 2011 International Conference on* , vol., no., pp.433,437, 9-11 Sept. 2011
8. Kortuem, G. et al., "Smart objects as building blocks for the Internet of things," *Internet Computing, IEEE* , vol.14, no.1, pp.44,51, Jan.-Feb. 2010
9. D. Lora et al., "Sistema de Seguridad Basado en una Plataforma Heterogénea Distribuida", Enseñanza y Aprendizaje de Ingeniería de Computadores, 5: 29-38 (2015).
10. F. Parrales et al. "Una Orquesta Sinfónica como Ejemplo de Aplicación de un Sistema Empotrado Distribuido", Enseñanza y Aprendizaje de Ingeniería de Computadores, 5: 115-124 (2015).
11. I.M. Laclaustra et al. "Sistema Domótico Distribuido para Controlar el Riego y el Aire Acondicionado en el Hogar", Enseñanza y Aprendizaje de Ingeniería de Computadores, 6: 87-102 (2016).
12. R. Braojos et al., "Ultra-low power design of wearable cardiac monitoring systems," *Design Automation Conference (DAC), 2014 51st ACM/EDAC/IEEE*, San Francisco, CA, 2014, pp. 1-6.
13. A. A. Del Barrio et al., "A Distributed Clustered Architecture to Tackle Delay Variations in Datapath Synthesis," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 35, no. 3, pp. 419-432, March 2016.
14. A. A. Del Barrio et al., "A Distributed Controller for Managing Speculative Functional Units in High Level Synthesis," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 30, no. 3, pp. 350-363, March 2011.
15. Jesús Martín Alonso et al., 2016. A distributed HW-SW platform for fireworks. In Proceedings of the *Summer Computer Simulation Conference* (SCSC '16). Society for Computer Simulation International, Montreal, Article 17 , 7 pages.
16. A. Dias Junior et al., "Estimation of Blood Pressure and Pulse Transit Time Using Your Smartphone," *Digital System Design (DSD), 2015 Euromicro Conference on*, Funchal, 2015, pp. 173-180.
17. Marwedel, P. *Embedded system design*. Kluwer, 2003.
18. C. Perera et al., "Context Aware Computing for The Internet of Things: A Survey," in *IEEE Communications Surveys & Tutorials*, vol. 16, no. 1, pp. 414-454, First Quarter 2014.
19. A. Zanella et al., "Internet of Things for Smart Cities," in *IEEE Internet of Things Journal*, vol. 1, no. 1, pp. 22-32, Feb. 2014.