Departamento de Ingeniería Electrónica
E.T.S.I. de Telecomunicación
Universidad Politécnica de Madrid

Enunciado del proyecto estándar de Sistemas Digitales II (SDG2)

ArkanoPi: una versión para la Raspberry Pi del popular videojuego arcade desarrollado por Atari

Fernando Fernández Martínez (coordinador)
fernando.fernandezm@upm.es
Alberto Boscá Mojena (coordinador administrativo)
alberto.bosca@upm.es

Curso 2016-2017

Grado en Ingeniería de Tecnologías y Servicios de Telecomunicación

Última revisión: 1, Febrero, 2017

Contenido

1. Aviso inicial 5
2. Introducción 5
2.1. Sobre la evaluación del proyecto 6
2.2. Proyectos innovadores 6
2.3. Sobre la organización del documento 6
3. Especificación de requisitos del sistema 7
3.1. Objetivo general 8
4. Subsistema Hardware 10
4.1. Matriz de leds 10
4.2. Pulsadores 11
4.3. Observaciones adicionales sobre el HW 11
5. Subsistema Software 12
5.1. Esquema de procesos 12
5.2. Interrupciones 14
5.3. Proceso principal y máquina de estados 15
5.4. Estado S1 - wait_start 15
5.5. Estado S2 - wait_push 15
5.6. Estado S3 - wait_end 17
6. Desarrollo recomendado 17
6.1. Sesiones 1, 2 y 3: Versión 1.0 del juego 18
6.2. Sesión 4: Versión 2.0 del juego 19
6.3. Sesiones 5 y 6: Versión 3.0 del juego 20
6.4. Sesión 7: Versión 4.0 del juego 22
6.5. Sesión 8: Versión 5.0 del juego 23
6.6. Sesiones 9 y 10: Versión 6.0 del juego 23
6.7. Sesiones 11-12: Mejoras y puesta a punto del sistema final 24
6.8. Mejoras con puntuación predefinida 24
6.9. Otras mejoras 25
6.10. Fechas importantes 27
6.11. Entregas electrónicas del código previstas 29
7. Material complementario 30
8. ANEXOS 31
8.1. Proyecto inicial: arkanoPi_1 31
8.2. Aspectos básicos de diseño 32
8.3. arkanoPi_1.c 37

8.4. arkanoPi_1.h 40
8.5. arkanoPiLib.c 41
8.6. arkanoPiLib.h 44

Índice de figuras

Figura 1. Captura de pantalla de Arkanoid, nueva y mejorada versión de Breakout lanzada por Taito en los 80. 7
Figura 2. Detalle del subsistema HW de visualización propuesto (ejemplo para 8 filas y 8 columnas). 10
Figura 3. Detalle del esquema de procesos propuesto para el proyecto. 12
Figura 4. Máquina de estados propuesta para el sistema. 15
Figura 5. Captura de una ventana de terminal que permite observar el desarrollo de una partida para la versión 1.0 del juego, a implementar en las 3 primeras sesiones. 18
Figura 6. Detalle del programa para la validación del HW de visualización: estados mostrados por pantalla a intervalos de 1 segundo. 21
Figura 7. Niveles de valoración de las posibles mejoras a implementar. 26
Figura 8. Calendario de fechas importantes para la asignatura. 28
Figura 9. Detalle del objeto pantalla y relación con los objetos raqueta, pelota y ladrillos. 33
Figura 10. Modelo propuesto de posibles trayectorias que podrá seguir la pelota durante el juego. 34
Figura 11. Modelo tentativo de rebotes contra la raqueta. 35

Índice de tablas

Tabla 1. Detalle de las diferentes conexiones al puerto de salida. 11
Tabla 2. Resumen del trabajo previo, objetivos, recursos y resultados de las tres primeras sesiones. 19
Tabla 3. Resumen del trabajo previo, objetivos, recursos y resultados de la cuarta sesión. 20
Tabla 4. Resumen del trabajo previo, objetivos, recursos y resultados de las sesiones 5 y 6. 22
Tabla 5. Resumen del trabajo previo, objetivos, recursos y resultados de la séptima sesión. 22
Tabla 6. Resumen del trabajo previo, objetivos, recursos y resultados de la octava sesión. 23
Tabla 7. Resumen del trabajo previo, objetivos, recursos y resultados de las sesiones 9 y 10. 24
Tabla 8. Resumen del trabajo previo, objetivos, recursos y resultados de las sesiones 11, 12 y 13. 24
Tabla 9. Mejoras con puntuación predefinida. 25
Tabla 10. Otras mejoras. 27
Tabla 11. Actualización necesaria de la trayectoria de la pelota tras rebote con alguna pared. 34

1. Aviso inicial

Cuando aborde la lectura de este documento, hágalo con tranquilidad y detenimiento. Frases o comentarios que no se entiendan en una primera lectura pueden encerrar avisos y recomendaciones que le serán útiles a lo largo del desarrollo.

No se preocupe si no alcanza a comprender todos los términos, conceptos y detalles que se discuten. Todos ellos se irán aclarando a medida que avance en la lectura de éste y otros documentos (i.e. tutoriales y demás recursos de apoyo, que permitirán profundizar en determinados aspectos del diseño). Por supuesto, asuma que necesitará varias lecturas y una reflexión a fondo sobre todo esto.

Finalmente, preste especial atención a todas las referencias explícitas a aspectos que se indican como obligatorios para incluir en el sistema y en la memoria correspondiente: no quiere decir que algo no referenciado explícitamente no tenga que ser tratado, sino que nuestra experiencia demuestra que algunos de esos aspectos no son considerados por un cierto número de alumnos, lo que da lugar a desagradables sorpresas en los exámenes.

2. Introducción

SDGII tiene por objetivos docentes los siguientes: en primer lugar, la aplicación y consolidación de los conocimientos sobre sistemas basados en microprocesadores o microcontroladores adquiridos en SDGI; y en segundo lugar, y como ampliación del primero, la adquisición y aplicación de conocimientos sobre programación de sistemas autónomos o empotrados (embedded) basados en un microprocesador (incluyendo hardware y software).

Con esos propósitos, la asignatura plantea el **desarrollo de un sistema electrónico complejo basado en un microcontrolador** partiendo de una descripción y unas especificaciones básicas **conforme a un caso real de diseño** que el alumno podrá consultar en este mismo documento.

El curso está organizado en sesiones de laboratorio orientadas a la implementación de un nuevo módulo o una nueva versión del sistema final a implementar. En particular, cada sesión planteará al alumno la consecución de un **hito** que corresponderá a un cierto nivel de desarrollo o madurez (funcionalidad) alcanzado por el prototipo.

Las primeras tres sesiones, en las que se presentarán los conceptos y las herramientas básicas necesarios para el desarrollo del proyecto propuesto, permitirán al alumno conseguir una **primera versión simplificada pero completamente funcional del sistema** (versión 1.0). Posteriormente, la consecución de cada nuevo hito, aplicando las herramientas y fundamentos adquiridos, significará la implementación de una nueva versión del prototipo (versión 2.0, 3.0, ...) al que se le irán añadiendo **nuevos elementos hardware** (pulsadores, displays, ...) **y software** (nuevos eventos y estados, temporización, ...) **que completarán y mejorarán su funcionalidad**.

Para ello, deberá seguir las instrucciones aquí incluidas, que implicarán diversas fases de diseño, análisis, implementación y medida de los circuitos y programas propuestos. Igualmente, se contará con todos los medios disponibles en el laboratorio B-043 así como también con la ayuda de los profesores y colaboradores docentes.

2.1. Sobre la evaluación del proyecto

El resultado del trabajo realizado debe quedar reflejado en una **memoria final** que contenga los detalles del proceso, así como los resultados obtenidos y todas aquellas cuestiones específicas que se indiquen en el presente enunciado o que sean notificadas a través de la plataforma Moodle de la asignatura. Tanto las instrucciones de entrega como la **plantilla** para la elaboración de esta memoria serán convenientemente publicadas y anunciadas a través de dicha plataforma.

De igual modo, el alumno deberá entregar electrónicamente, y haciendo uso de los medios habilitados a tal efecto en dicha plataforma, **una copia del código desarrollado correspondiente a cada versión haciendo notar la consecución o no del hito en cuestión.**

Salvo que se indique lo contrario, el proyecto propuesto contiene las **especificaciones mínimas obligatorias** que deben cumplir los sistemas y serán **valoradas con un máximo de 7 puntos**. **Esta nota máxima sólo se conseguirá si se cumplen todas las especificaciones y los resultados de las diferentes pruebas de evaluación continua son perfectos.** Adicionalmente a las especificaciones mínimas, se presentarán sugerencias de mejoras opcionales, dejando a los alumnos la libertad para que añadan nuevas mejoras o esquemas alternativos (ver sección 5.7). Con estas mejoras o montajes alternativos añadidos al prototipo básico, y dependiendo de su dificultad y realización, se podrán sumar puntos hasta alcanzar la máxima nota, 10 puntos. **Nótese que las mejoras añadidas sólo serán consideradas en caso de haber superado la evaluación final sobre la versión definitiva del juego (control oral y control escrito, ambos aprobados).**

2.2. Proyectos innovadores

Aquellos alumnos que deseen realizar un **proyecto innovador** basada en un **problema o un diseño propios** (o propuesto por un profesor), deberán hablar con alguno de los profesores de la asignatura y presentarle un listado de notas y una propuesta de proyecto donde describan, en 2 o 3 páginas:

- Objetivos del sistema propuesto.
- Recursos necesarios para llevarlo a cabo.
- Arquitecturas HW y SW propuestas para la resolución del problema.

Para poder abordar el proyecto será necesario contar con la aprobación de dicho profesor. No será admitido ningún proyecto (por muy complejo o perfecto que sea) que no se ajuste a estas normas.

2.3.Sobre la organización del documento

Los apartados siguientes del presente documento están organizados de la siguiente manera: en primer lugar se facilitará una descripción detallada de la especificación de requisitos que debe cumplir el prototipo final a implementar. A continuación, se detallarán las arquitecturas HW y SW del mismo, haciendo especial énfasis en la descomposición modular del sistema, en la interacción HW-SW y en los requisitos de tiempo real, todos ellos aspectos clave para abordar con éxito el proyecto. Finalmente se describirá la planificación recomendada para su desarrollo, organizada por sesiones de laboratorio para las cuales se indicarán tanto los objetivos a conseguir durante las mismas como los recursos disponibles para ello.

3. Especificación de requisitos del sistema

Uno de los sectores de mayor auge de la industria electrónica, tanto a nivel nacional como internacional, es el de los videojuegos. En España, buena prueba de ello la constituye el informe anual de la AEVI[1], Asociación Española de Videojuegos. Según los datos recogidos en su último informe, correspondiente al año 2015, los videojuegos facturaron durante ese año 1083 millones de euros en España (lo que supone un crecimiento del 8.7% con respecto al año anterior), dato que consolida a la industria del videojuego como líder del ocio audiovisual en España (se estiman 15 millones de jugadores en todo el país), superando al cine, con 571 millones de euros en facturación. Este auge es también evidente en los datos de empleo que arroja el sector, donde las contrataciones crecen anualmente por encima del 20%.

Estos datos subrayan la importancia de un sector cuyo crecimiento se estima garantizado a corto plazo (gracias, entre otras, a nuevas tendencias como la realidad virtual y los dispositivos inmersivos) y que constituye un auténtico motor de las industrias tecnológicas.

Para el proyecto estándar de este curso académico no hemos contemplado cotas tan ambiciosas como las anteriormente mencionadas. En su lugar hemos realizado un cierto ejercicio de retrospectiva que nos ha llevado hasta casi el mismo comienzo de la historia de los videojuegos: el lanzamiento al mercado el 13 de mayo de 1976 de *Breakout*, un videojuego arcade desarrollado por Atari (fundada en junio de 1972 por Nolan Bushnell y Ted Dabney). *Breakout* fue creado por el propio Bushnell junto con Steve Bristow, ambos influenciados por el videojuego de 1972 *Pong*, también de Atari y considerado por muchos el primer videojuego de la historia.

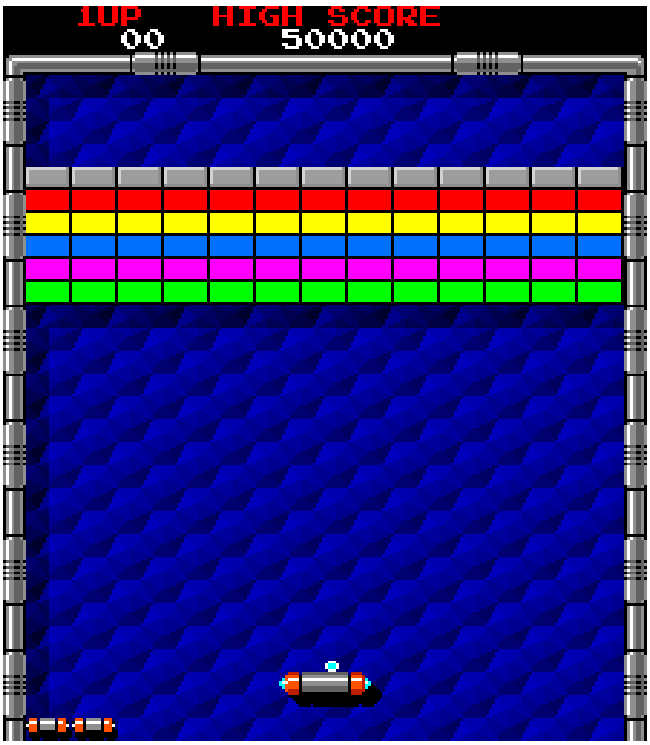

Figura 1. Captura de pantalla de Arkanoid, nueva y mejorada versión de Breakout lanzada por Taito en los 80.

[1] http://www.aevi.org.es/web/wp-content/uploads/2016/06/MEMORIA-ANUAL_2015_AEVI_-definitivo.pdf

La mecánica del *Breakout* era bien sencilla. Básicamente en la parte inferior de la pantalla una rayita simulaba una raqueta de front-tenis que el jugador podía desplazar de izquierda a derecha. En la parte superior se situaba una banda conformada por rectángulos que simulaban ser ladrillos. Una pelotita descendía de la nada y el jugador debía golpearla con la raqueta, entonces la pelota ascendía hasta pegar en el muro y los ladrillos tocados por la pelota desaparecían. La pelota volvía a descender y así sucesivamente. El objetivo del juego era terminar con la pared de ladrillos.

En la presente práctica supondremos que un cliente (el Departamento de Ingeniería Electrónica) nos ha contratado para desarrollar un prototipo de videojuego de bajo coste. Este prototipo estará inspirado en el modelo o la mecánica de juego anteriormente detallada pero, a diferencia del de 1976 para el que los creadores simplemente emplearon dispositivos discretos y lógica discreta, contaremos con la ventaja de disponer de una Raspberry Pi, un microcontrolador de referencia y con altas prestaciones. Dado que el propósito de este laboratorio es eminentemente docente (con especial énfasis en la interacción HW-SW, así como en los requisitos de tiempo real), el objetivo será implementar un prototipo funcional completo. Este enunciado constituye una guía para su diseño e implementación. Adicionalmente, el alumno dispone de un vídeo demostrativo sobre este proyecto en YouTube: https://youtu.be/BhlKW1HB3Oc.

3.1. Objetivo general

El objetivo del proyecto es mostrar la viabilidad de la idea de diseño básica desarrollando un prototipo plenamente funcional. El programa que ejecutará el micro se realizará en lenguaje C. El sistema digital basado en un microcontrolador será la plataforma ENT2004CF disponible en el laboratorio B-043 (construida en torno a una Raspberry Pi). Para más detalles relacionados con el sistema de desarrollo, consulte la publicación [1].

En los apartados siguientes se detallarán las arquitecturas HW y SW, haciendo énfasis en la descomposición modular del sistema, tarea clave para abordar con éxito el diseño de cualquier sistema HW o SW medianamente complejo.

El objetivo general se resume en desarrollar un “sistema digital basado en la plataforma ENT2004CF” con las siguientes características:

- El sistema deberá **informar al usuario mediante un mensaje por la terminal de que el juego puede comenzar**, para lo que bastará con accionar cualquiera de los pulsadores. La posterior acción de alguno de los pulsadores deberá servir al sistema para que este advierta al jugador del comienzo del juego.
- El juego se mostrará en una **matriz de leds de 7x10** , que actuará como display de visualización. En dicha matriz se deberá mostrar la **raqueta** (de dimensiones 1x3), **la pelota, y los ladrillos** (de 1x1 cada elemento). Para mostrar todos los elementos se llevará a cabo una adecuada excitación de los leds del display, empleándose 11 terminales digitales de salida de la plataforma ENT2004CF (7 terminales para las filas y 4 terminales para las columnas). De esta manera, el jugador observará cómo la pelota se desplaza por el display a una determinada velocidad constante.
- **La raqueta se posicionará en la fila inferior del área de juego**. Con objeto de poder eliminar todos los ladrillos, el jugador podrá modificar la posición de la raqueta horizontalmente, sin que esta rebase completamente los límites del área de juego en ningún momento y sin retardo en su movimiento apreciable. Para

ello **el jugador contará con dos pulsadores** cuya acción permitirá desplazar la raqueta en una posición (i.e. un led) hacia izquierda o derecha.

- **La raqueta no podrá detener la pelota pero sí que podrá modificar su trayectoria dentro del área de juego**. Para ello se aprovechará la detección del impacto de la pelota con la raqueta para definir una nueva trayectoria (i.e. nuevo ángulo de salida que adquiere la pelota tras su impacto con la raqueta) que dependerá de:
    - el ángulo de incidencia o entrada de la pelota: ángulo correspondiente a la trayectoria inicial o previa al choque de la pelota con la raqueta,
    - el lugar de impacto o posición de la raqueta en la que se produce el impacto con la pelota. Existirán tres posibles lugares de impacto, correspondientes a los tres "leds" de los que estará compuesta una raqueta.
- El juego comenzará con la pelota siempre en el centro del área de juego, pero aleatorizando para cada partida la trayectoria inicial asignada a la misma. **Durante el juego la pelota se desplazará a una velocidad constante** (e.g. la posición de la pelota se actualizará una vez cada segundo).
- **La detección de cada impacto de la pelota con alguno de los ladrillos será aprovechada para el borrado o la eliminación de dicho ladrillo.**
- Igualmente, **los límites superior, izquierdo y derecho** del área de juego constituirán superficies o **paredes contra las que la pelota deberá poder impactar**, saliendo esta rebotada con una nueva trayectoria en caso de que esto suceda. Esta nueva trayectoria dependerá, en este caso, única y exclusivamente de su ángulo de incidencia.
- Cuando el jugador no consiga golpear la pelota, rebasando esta la posición ocupada por su raqueta, y por tanto el límite inferior del área de juego, se **dará por finalizado el juego. Igualmente, el juego se dará también por concluido cuando el jugador consiga eliminar todos los ladrillos**. En ambos casos se informará al usuario del número de ladrillos eliminado mediante un mensaje por la terminal y, tras la acción por parte del usuario de cualquiera de los pulsadores, se devolverá al sistema a su estado inicial a la espera de que dé comienzo una nueva partida.
- Los mensajes destinados al usuario se mostrarán en la pantalla del ordenador mediante la ventana de terminal del entorno de desarrollo Eclipse.
- En la versión final, los retardos debidos a, por ejemplo, escribir por la terminal o gestionar las pulsaciones, no deben afectar a la correcta visualización de los elementos del juego. En particular, **no deben apreciarse parpadeos significativos en los leds del display** (requisitos de concurrencia y tiempo real).
- Será, igualmente, competencia del sistema SW, generar y procesar las señales necesarias para gobernar el HW externo.
- Los posibles rebotes mecánicos debidos al accionar un pulsador se suprimirán por SW. **Si se realiza una única pulsación el sistema debe interpretar una única pulsación** sin que le afecten los posibles rebotes en la señal que se reciba del pulsador.

- La duración de una pulsación (sea breve o sea larga) no debe dar lugar a que se detecten varias pulsaciones del mismo pulsador. El sistema debe ser capaz de detectar pulsaciones lo suficientemente breves como para desplazar la raqueta de la manera deseada.

4. Subsistema Hardware

4.1. Matriz de leds

Conceptualmente necesitamos un display o subsistema HW de visualización constituido por **una matriz de 7 filas y 10 columnas de leds**, subsistema que podremos implementar siguiendo un esquema similar al presentado en la Figura 2 (nótese que el esquema propuesto es para un display de dimensiones 8x8; el alumno deberá realizar las modificaciones necesarias para adaptarlo a las necesidades del proyecto).

Siguiendo dicho esquema, podemos comprobar cómo para encender el led ubicado en una fila y columna determinadas basta con poner a nivel bajo la fila (i.e. 0V) y a nivel alto la columna (i.e. 5V) correspondientes. Igualmente, excitada una cierta columna, para mantener apagado alguno de sus leds bastará con poner a nivel alto la fila correspondiente al mismo.

La ubicación de los ladrillos, la pelota y la raqueta dentro del área de juego permitirá determinar qué leds de la matriz deben permanecer encendidos o apagados en cada momento, consiguiéndose así la visualización de los elementos del juego en el display.

Como los leds estarán organizados en forma de matriz 7x10, se usarán 7 terminales del puerto de salida para iluminar o no cada uno de los 7 leds (o filas) que hay en cada columna, y otros 4 para seleccionar, por medio de un decodificador (nótese que el propuesto en el montaje a modo de ejemplo, el CD74HC238, decodificador de 3 a 8 líneas, no bastaría para cumplir con las especificaciones por lo que deberá encontrar una solución alternativa), qué columna iluminar en cada momento.

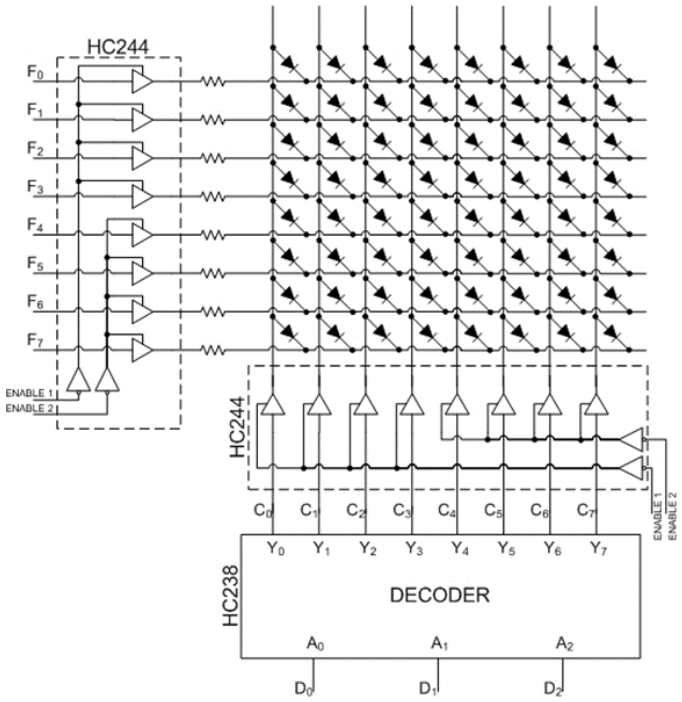

Figura 2. Detalle del subsistema HW de visualización propuesto (ejemplo para 8 filas y 8 columnas).

Puerto de salida	13	12	11	10	9	8	7	6	5	4	3	2	1
Conexión	-	-	ADC/DAC SPI_Dout	ADC SSTRB	-	GPIO-10 SPI_MOSI	GPIO-08 SPI_CS0	GPIO-07 FILA-5	GPIO-04 FILA-4	GPIO-03 FILA-3	GPIO-02 FILA-2	GPIO-01 FILA-1	GPIO-00 FILA-0
Puerto de salida	26	25	24	23	22	21	20	19	18	17	16	15	14
Conexión	-	GND	GND	-	-	GPIO-25 LIBRE	GPIO-24 LIBRE	GPIO-23 FILA-6	GPIO-22 COL-3	GPIO-18 COL-2	GPIO-17 COL-1	GPIO-14 COL-0	SPI_CLK

Tabla 1. Detalle de las diferentes conexiones al puerto de salida.

En la Tabla 1 se puede observar el detalle de la propuesta de pines del puerto de salida a emplear para las filas en verde y para las columnas en rojo respectivamente.

Para que el refresco columna a columna del display se produzca sin parpadeo apreciable, el subsistema SW:

- iluminará los leds apropiados de la primera columna durante 1 ms,
- posteriormente, iluminará sucesivamente las siguientes columnas durante el mismo periodo de tiempo,
- tras alcanzar e iluminar la última columna, volverá a la 1ª cíclicamente.

De esta manera, cada led se puede encender y apagar hasta 100 veces por segundo (i.e. 1 excitación cada 10 ms, 100 veces/seg), produciéndose la ilusión de que los leds de varias columnas están encendidos a la vez, cuando realmente en cada instante de tiempo sólo están siendo excitados los leds de una columna en particular.

Nótese que las conexiones al puerto de salidas digitales del entrenador, tanto las de las columnas como las de las filas, requieren un buffer 74HC244. Estos buffers tienen por objetivo que no se exija mucha corriente a los puertos del entrenador, sino a los propios 74HC244. Las conexiones entre el buffer y las filas requieren además de un conjunto de resistencias de un valor adecuado (será responsabilidad del alumno determinarlo) que permita que los leds no consuman mucha corriente pero al mismo tiempo se iluminen bien.

4.2. Pulsadores

El subsistema HW necesario se completará con la conexión de dos pulsadores que permitirán controlar la posición de la raqueta desplazándola a la izquierda o a la derecha. Para ello el alumno deberá calcular las resistencias de pull-up/pull-down necesarias.

Cada pulsador estará conectado a un pin diferente del puerto de entrada de la Raspberry Pi, ambos a elegir por el alumno.

4.3. Observaciones adicionales sobre el HW

Es necesario recordar que los puertos de entrada y salida disponibles en la plataforma ENT2004CF están formados por buffers digitales con resistencias de pull-down de 10 KΩ. Si no se tiene en cuenta este hecho, se podrían producir efectos de carga no deseados al conectar el HW a la plataforma.

Igualmente importante desde el punto de vista de la alimentación de los diferentes subsistemas es el hecho de desacoplar las alimentaciones y alimentar en estrella.

5. Subsistema Software

El software estará basado en una máquina de estados que gestionará la entrada/salida, las interrupciones y la temporización.

5.1. Esquema de procesos

Frecuentemente, los sistemas electrónicos digitales precisan realizar diversas acciones de una manera paralela (concurrente en varios procesadores) o aparentemente paralela (todas las acciones se ejecutan entrelazadas, produciéndose la apariencia de paralelismo si la frecuencia de conmutación entre ambas es elevada). Son los denominados procesos. La creación, ejecución, sincronización y destrucción entre procesos suele ser facilitada por el sistema operativo a través de diversas primitivas. Pero incluso en ese caso, el diseñador debe definir qué tareas o rutinas serán concurrentes, elegir los mecanismos de comunicación entre ellas, los tiempos de vida, etc.

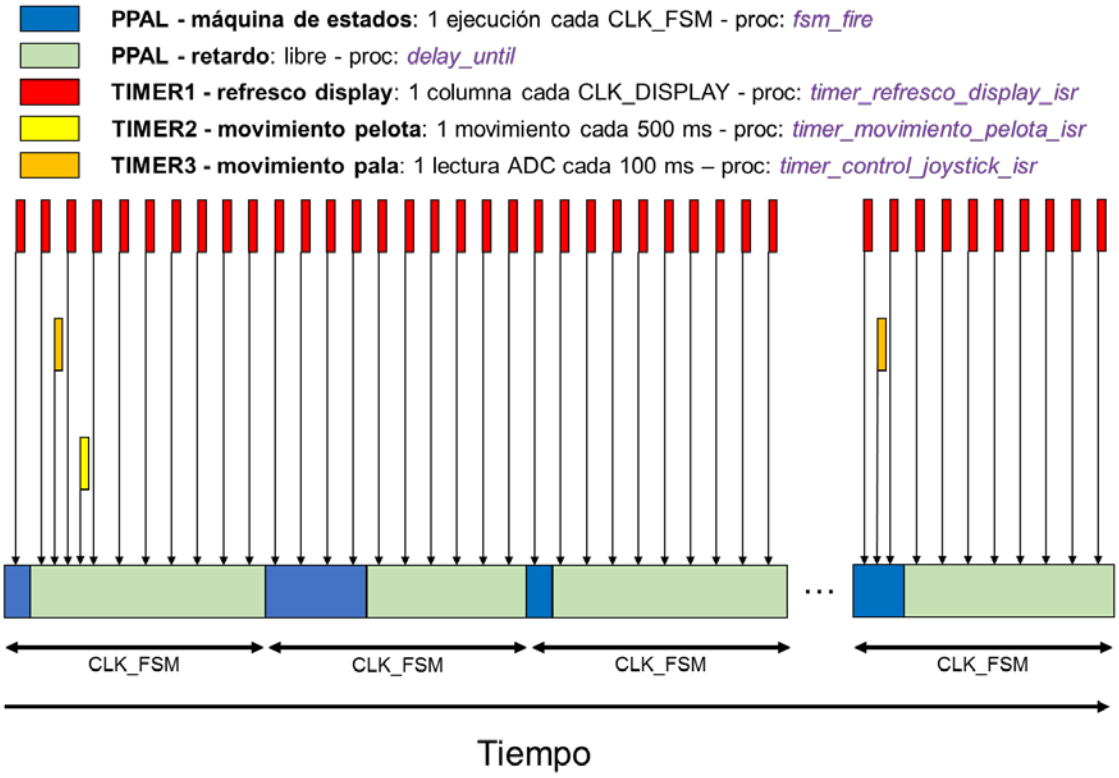

Figura 3. Detalle del esquema de procesos propuesto para el proyecto.

En la Figura 3 se muestra el esquema de procesos propuesto para este proyecto. Dicho esquema comprende 3 procesos (4 en caso de implementar la mejora de control mediante joystick analógico; más detalles en el apartado 5.6):

- **el programa principal**: siempre existe y debe: inicializar los objetos del sistema, inicializar el HW, inicializar y habilitar tanto los procesos de interrupción periódica como los vinculados a las interrupciones externas. Será igualmente el **responsable de inicializar y ejecutar la máquina de estados** con la que controlaremos el sistema. Tal y como puede observarse en la Figura 3, dicha máquina de estados se ejecutará de forma permanente en un **bucle infinito** mediante la **invocación periódica** (i.e. 1 vez cada CLK_FSM milisegundos) de

la función *fsm_fire* que, a su vez, será la encargada de ejecutar las funciones de comprobación (si se ha producido algún evento especial para el sistema, e.g. que haya ocurrido una interrupción) y actualización del estado del sistema (leer [5]). Nótese que el tiempo necesario para hacerlo (fragmentos en color azul en la figura) puede variar dependiendo del estado en el que se encuentre el sistema.

- **dos** (tres en caso de implementar la mejora) **interrupciones periódicas** (i.e. temporizadores):
    - **TIMER1:** interrupción responsable de controlar la parte de **temporización del sistema relacionada con el refresco de la matriz de leds**, refresco del cual depende directamente su correcta visualización en el display HW. Su ejecución es cíclica, discontinua y sucede a intervalos regulares de tiempo (1 vez cada milisegundo); es concurrente con el proceso principal y la otra interrupción periódica. Se encargará de la ejecución de todas las funciones críticas en el tiempo relacionadas con el refresco de los leds. A diferencia del resto de temporizadores, este **estará activo de forma permanente**, desde el mismo momento en que finalice la inicialización del sistema e **independientemente del estado** en el que este se encuentre de manera que, en cualquiera de dichos estados, resulte posible (aunque no obligatoria) la visualización de cualquier tipo de información en el display.
    - **TIMER2:** interrupción responsable de controlar la parte de **temporización del sistema relacionada con el movimiento de la pelota**, movimiento del cual depende directamente el correcto funcionamiento del juego. Su ejecución también es cíclica, discontinua y sucede a intervalos regulares de tiempo (1 vez cada 500 milisegundos); es concurrente con el proceso principal y la otra interrupción periódica pero **sólo estará activa durante el transcurso del juego**.
    - **TIMER3** (sólo en caso de implementar la mejora del control mediante joystick analógico)**:** interrupción responsable de controlar la parte de **temporización del sistema relacionada con el movimiento de la raqueta**, movimiento del cual depende también el correcto funcionamiento del juego. Su ejecución también es cíclica, discontinua y sucede a intervalos regulares de tiempo (1 vez cada 100 milisegundos); es concurrente con el proceso principal y la otra interrupción periódica pero, al igual que sucede con el TIMER2, **sólo estará activa durante el transcurso del juego**.

Nótese que, al margen del programa principal y de aquellos procesos que requieran de algún control del tiempo como los mencionados, **serán igualmente necesarias 2 interrupciones externas ligadas respectivamente a los dos pulsadores** de control del movimiento de la raqueta.

Este esquema de procesos es obligatorio.

Para el desarrollo de software de tiempo real sobre la plataforma disponible en el laboratorio es indispensable la lectura de [4]. En ese documento encontrará la información necesaria para la definición de diferentes procesos por medio de interrupciones periódicas, interrupciones externas y el programa principal, así como para la provisión de los requeridos mecanismos de sincronización entre todos ellos.

5.2. Interrupciones

La máquina de estados hará uso de las interrupciones (a excepción de la ya mencionada interrupción asociada al refresco del display), independientemente de que estas sean periódicas (i.e. vinculada a un temporizador) o externas (i.e. vinculada a una entrada digital), por medio de la misma aproximación presentada en [5]: utilizaremos una **variable global para indicar si se ha producido o no** la interrupción en cuestión (a modo de flag). De este modo, la ocurrencia de cada interrupción provocará la llamada y ejecución de una **función de atención a la interrupción** (procedimientos cuyo nombre termina por _isr conforme a la nomenclatura empleada en [5], correspondiente a Interrupt Service Routine), **que pondrá a 1 la variable activando el correspondiente flag**. Cuando el sistema, a través de la máquina de estados, haya tenido en cuenta el evento reflejado por dicha activación, volverá a poner la variable, y con ello el flag, a 0. Con objeto de simplificar la gestión de las diferentes interrupciones y sus correspondientes flags asociados, en vez de tener una variable para cada interrupción, como solo hace falta un bit, utilizaremos una única variable global *flags* para todas (tenemos para 32 interrupciones). El uso de un conjunto de máscaras con las funciones binarias OR y AND nos permitirán referirnos a cada uno de los flags por separado para su oportuna lectura o escritura (el alumno debe leer [5] para comprender en detalle estos mecanismos).

Como consecuencia inmediata de simplificar según el modo sugerido la atención a las interrupciones, el tiempo necesario para completar ésta será despreciable en comparación con el requerido por *fsm_fire* para actualizar (si procede) el estado del sistema. Por idéntico motivo, este último puede verse razonablemente afectado por la ocurrencia de interrupciones ligadas a TIMER2, TIMER3 o a cualquiera de los pulsadores (y por tanto a la activación de sus correspondientes flags). El alumno **deberá diseñar un valor adecuado para la constante CLK_FSM** que permita una frecuencia de actualización del estado del sistema lo suficientemente alta como para que los tiempos de respuesta sean bajos (algo necesario para una buena experiencia de juego), y sin que por ello se vean comprometidas ninguna de las tareas a realizar en el peor escenario posible (i.e. co-ocurrencia múltiple de eventos entre sucesivas llamadas a *fsm_fire*).

Nótese que las funciones asociadas a un mismo objeto, por ejemplo al display de leds, **no tienen por qué ejecutarse necesariamente en un mismo proceso.** Dicho objeto, entre otras, debe contener la información relativa al estado del área de juego, estado que determina el correspondiente encendido o apagado de los leds que componen el display. Esa información debe ser consultada por la interrupción periódica encargada del refresco del display, pero es el programa principal, a través de la máquina de estados, el responsable de su actualización con motivo de los respectivos movimientos de la pelota y la raqueta.

Igualmente, nótese también que las interrupciones periódicas pueden interrumpir la ejecución del programa principal en instantes de tiempo que resultan imprevisibles en el momento de escribir el código. Por tanto, será **fundamental contar los mecanismos de sincronización oportunos que nos permitan actualizar correctamente todos los objetos compartidos** (conjunto de variables globales) entre los distintos procesos con el fin de intercambiar información. Preste para ello especial atención al capítulo 6 de [4] y en particular a la sección 6.5. Sincronización de threads.

5.3. Proceso principal y máquina de estados

El sistema completo propuesto se basa en una máquina de tres estados (S1-S3), tal y como se muestra en la Figura 4. Como se ha comentado previamente, la interfaz de control para el usuario constará de dos pulsadores con los que el usuario podrá dar comienzo al juego y controlar el movimiento de la raqueta durante el mismo.

Figura 4. Máquina de estados propuesta para el sistema.

5.4. Estado S1 - wait_start

El sistema espera a que cualquiera de los pulsadores de control sea accionado para dar comienzo al juego. En este estado todos los leds del display pueden estar apagados. Alternativamente, se podría presentar un mensaje o figura a modo de bienvenida (e.g. un smiley). La acción por parte del usuario sobre cualquiera de los pulsadores (EVENT_BUTTON_START) dará paso a la inicialización del juego (procedimiento *InicializaJuego*) y al siguiente estado S2 en el que se produce de forma efectiva el comienzo del juego.

5.5. Estado S2 - wait_push

Estado en el que se desarrolla el juego. Durante el mismo, estos serán los eventos fundamentales cuya detección será obligatoria para determinar la correcta evolución del juego:

- EVENT_BUTTON_PALA_IZQUIERDA: evento ligado a uno de los flags de estado del sistema que será indicativo de la acción del usuario sobre el pulsador de control correspondiente al movimiento hacia la izquierda de la raqueta. Su eventual comprobación y detección por parte de la máquina de estados

provocará que el sistema lleve a cabo todas las acciones oportunas (procedimiento *MuevePalaIzquierda*) para desplazar la raqueta en una posición en el display hacia la izquierda (en caso de que sea posible, esto es, siempre y cuando no se excedan los límites del área de juego).

- EVENT_BUTTON_PALA_DERECHA: similar al anterior pero en este caso correspondiente al movimiento hacia la derecha de la raqueta.
- EVENT_TIMEOUT_PELOTA: evento ligado a otro de los flags de estado del sistema que en éste caso será indicativo de la necesidad de actualizar la posición ocupada por la pelota en el display como consecuencia del movimiento de ésta (la activación de dicho flag tendrá lugar en el momento en que el correspondiente temporizador sobrepase un valor de timeout definido previamente). Su eventual comprobación y detección por parte de la máquina de estados provocará que el sistema lleve a cabo todas las acciones oportunas (procedimiento *MovimientoPelota*) para:
    - desplazar la pelota en el display conforme a su actual trayectoria,
    - gestionar su eventual rebote contra algún otro elemento del juego (la raqueta, un ladrillo o una pared) actualizando convenientemente su trayectoria (y en el área de juego, en caso que proceda, tras el rebote con un ladrillo y su consiguiente eliminación),
    - resolver si la nueva posición de la pelota excede el límite inferior del área de juego (situación de final de juego) y, en ese caso, actualizar el estado del sistema haciendo notar dicha situación (activando el flag correspondiente),
    - actualizar (si procede) el número de ladrillos restante y, en caso de haber eliminado todos los ladrillos (situación de final de juego), actualizar el estado del sistema haciendo notar dicha situación (activando el flag correspondiente).
- EVENT_FINAL_JUEGO: evento ligado a otro de los flags de estado del sistema que en este caso será indicativo de la finalización del juego como consecuencia del último movimiento realizado por la pelota (la pelota ha rebotado contra el último ladrillo o ha rebasado el límite inferior del área de juego). Su eventual comprobación y detección por parte de la máquina de estados provocará (procedimiento *FinalJuego*):
    - Que el sistema apague el display (o alternativamente muestre en él un mensaje o figura acorde con el resultado del juego),
    - y muestre un mensaje por la terminal informando al usuario del número de ladrillos eliminados y de que puede hacer uso de cualquiera de los pulsadores para dar comienzo a una nueva partida.
- EVENT_TIMEOUT_JOYSTICK: evento adicional a considerar solo en el caso de implementar la mejora de control de la raqueta mediante un joystick analógico. También ligado a los flags de estado del sistema en este caso será indicativo de la necesidad de actualizar la posición ocupada por la raqueta en el display mediante la lectura e interpretación del valor devuelto por el ADC vía SPI (la activación de dicho flag tendrá lugar en el momento en que el temporizador correspondiente sobrepase el valor de timeout definido previamente). Su

eventual comprobación y detección por parte de la máquina de estados provocará que el sistema lleve a cabo todas las acciones oportunas (procedimiento *ControlJoystick*) para reubicar la raqueta en la posición del display correspondiente al mencionado valor (una vez más de forma coherente con los límites del área de juego).

De este modo, el sistema permanecerá en el estado S2 hasta que se produzca la finalización del mismo (EVENT_FINAL_JUEGO), situación que provocará la transición al estado S3.

5.6. Estado S3 - wait_end

Llegaremos a este estado cuando el juego haya finalizado. En este estado el sistema permanecerá a la espera de que el usuario accione alguno de los pulsadores (evento EVENT_BUTTON_PALA_IZQUIERDA o EVENT_BUTTON_PALA_DERECHA) para, acto seguido, llevar a cabo todas las acciones oportunas para resetear el sistema (e.g. apagar el display, en caso de que este estuviese encendido) y devolverlo al estado inicial S1.

Este proceso se repetirá indefinidamente hasta que se apague o reinicie el sistema.

6. Desarrollo recomendado

Tal y como recoge la guía docente de la asignatura, el curso está organizado en sesiones de laboratorio orientadas a la implementación de un nuevo módulo o una nueva versión del sistema final a implementar.

En particular, cada sesión planteará al alumno la consecución de un hito que corresponderá a un cierto nivel de desarrollo o madurez (funcionalidad) alcanzado por el prototipo. Posteriormente, la consecución de cada nuevo hito, aplicando las herramientas y fundamentos adquiridos, significará la implementación de una nueva versión del prototipo (2.0, 3.0, ...) al que se le irán añadiendo nuevos elementos hardware (pulsadores, displays,...) y software (nuevos eventos y estados, temporización,...) que completarán y mejorarán su funcionalidad.

Esta es una planificación orientativa en relación al posible desarrollo de la práctica. En caso de que los alumnos superen los objetivos planteados para una determinada sesión antes de la finalización de la misma, se recomienda al alumno tratar de abordar los objetivos y tareas planteados para la siguiente sesión.

Igualmente, y en relación con la redacción de la memoria final, **se recomienda que la realización de la memoria se vaya realizando a medida que se van consiguiendo cada uno de los hitos descritos**. Adicionalmente, tras la consecución de cada hito el alumno deberá hacer **entrega vía Moodle de una copia del código desarrollado** para la correspondiente versión del sistema.

Finalmente, con carácter general y con el fin de aprovechar al máximo las sesiones en el Laboratorio, que son un recurso muy limitado, **el alumno debe prestar especial atención al trabajo previo requerido** para cada sesión. En particular, el alumno deberá haber leído las fuentes recomendadas, completado los tutoriales sugeridos, realizado los montajes HW indicados o traerse escrita de casa una primera versión de los programas que se puedan solicitar. En este sentido, y en la medida de lo posible, es fundamental aprovechar las sesiones de Laboratorio para la depuración de errores de ejecución, que no de sintaxis, de los sistemas a desarrollar.

6.1. Sesiones 1, 2 y 3: Versión 1.0 del juego

Durante las tres primeras sesiones se presentarán los conceptos y las herramientas básicas necesarias (familiarización con el laboratorio, su instrumental y el entorno de desarrollo) para el desarrollo del proyecto propuesto. Igualmente, se espera que dichas sesiones sean suficientes como para permitir al alumno conseguir una primera versión simplificada pero completamente funcional del sistema (versión 1.0). Esta primera versión del juego alumno será totalmente jugable y deberá cumplir con las siguientes especificaciones:

- La **visualización de los elementos que componen el juego se realizará a través de la ventana de terminal** de la plataforma de desarrollo. En particular, se empleará la función **printf** para reflejar en cada momento el estado del área de juego. En la Figura 5 puede observarse a modo de ejemplo una captura que muestra la información a escribir por la terminal para hacer posible el juego. Nótese que el estado encendido o apagado de los leds (estado que, por el momento, será virtual hasta que se implemente el HW correspondiente al display) queda reflejado por los valores 1 ó 0 respectivamente.
- Para poder jugar, el usuario empleará **el teclado convencional** del PC. Para ello, el alumno contará, como punto de partida, con un **programa** (ver ANEXOS) **que facilitará la captura, y posterior interpretación en términos del juego, de las diferentes pulsaciones de sus teclas**. En particular, el alumno deberá gestionar la ocurrencia de 3 eventos básicos relacionados con la pulsación de las siguientes teclas:
    - Tecla i: desplazamiento de la raqueta hacia la izquierda,
    - Tecla o: desplazamiento de la raqueta hacia la derecha,
    - Tecla p: actualización de la posición de la pelota.

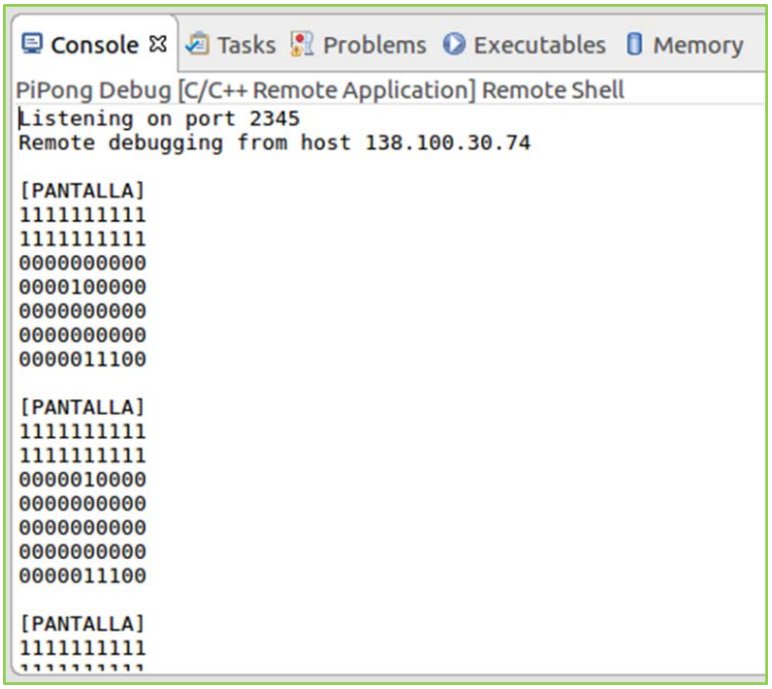

Figura 5. Captura de una ventana de terminal que permite observar el desarrollo de una partida para la versión 1.0 del juego, a implementar en las 3 primeras sesiones.

Nótese que en esta primera versión del juego la pelota no se desplazará de forma autónoma (su implementación está prevista en posteriores sesiones). Por el contrario, el movimiento de la pelota será "manual" al estar este condicionado a la pulsación de la tecla dedicada al efecto (i.e. la pelota sólo actualizará su posición en caso de que el usuario así lo decida).

En cualquier caso, e independientemente de la naturaleza de este (movimiento de la raqueta o movimiento de la pelota), con cada nuevo evento (pulsación) detectado será necesario re-pintar la pantalla.

El alumno dispone de más información acerca del programa de ejemplo con el que comenzar, así como de una descripción más detallada del modelo de implementación a adoptar, en el apartado correspondiente a ANEXOS incluido en este mismo documento.

Sesiones 1, 2 y 3) Versión 1.0 del juego	
Trabajo previo	• Serán de obligada lectura [1] y [2]. • Implementación previa de ciertas subrutinas (ver 7.1)
Objetivos docentes / de aprendizaje	Adquirir los conocimientos suficientes para poder: • Comprender el funcionamiento del entorno: - Perspectivas 'Debug' y 'Remote System Explorer' • Saber crear y configurar un proyecto desde cero - Saber crear y configurar un nuevo proyecto a partir de otro • Saber compilar el proyecto - Y entender el proceso… • Saber depurar: - Breakpoints, ejecución paso a paso, visualización de variables, … • Adquirir conocimientos de C necesarios y suficientes para abordar el proyecto, en particular: - Uso de arrays y estructuras de datos
Recursos adicionales	• Programa ejemplo con rutina de detección de pulsaciones, declaración de todos los objetos básicos y prototipos de las principales funciones a implementar
Resultados para el proyecto	• Familiarización con el entorno y herramientas básicas • Primera versión del juego totalmente jugable desde terminal y teclado del PC

Tabla 2. Resumen del trabajo previo, objetivos, recursos y resultados de las tres primeras sesiones.

6.2. Sesión 4: Versión 2.0 del juego

Sesión de gran importancia de cara al desarrollo del proyecto toda vez que propone como objetivo la **reformulación completa del sistema** conforme a uno de los paradigmas más populares para el desarrollo de programas: una **máquina de estados**. Para ello, el alumno hará uso de la implementación y los ejemplos presentados en [5], prestando especial atención al uso de flags de estado, y seguirá igualmente las directrices indicadas en este mismo documento, en particular, cuanto concierne a los apartados 2.1, 4.1 y 4.3. Nótese que, desde el punto de vista funcional, las dos versiones implicadas en el presente hito, la inicial y la nueva como resultado del mismo, serán indistinguibles (la nueva versión permitirá igualmente jugar al juego en idénticas condiciones a la versión anterior). Aunque algunas de las notables ventajas introducidas

por dicho cambio no serán evidentes hasta alcanzar etapas de desarrollo más maduras (en particular, cuando aparezcan la gestión de las interrupciones y los distintos eventos ligados a las mismas), otras serán notorias desde el mismo momento de su adopción (e.g. una vez completada la modificación el alumno podrá valorar la complejidad, o simplicidad, del bucle por medio del cual se define el comportamiento del programa principal y, en particular, el coste que supondría la inclusión de un nuevo estado, por ejemplo de pausa, en una y otra versión).

Sesión 4) Versión 2.0 del juego	
Trabajo previo	• Será de obligada lectura [5] • Versión 1.0 del sistema
Objetivos docentes / de aprendizaje	• Ser capaces de reformular el sistema a partir de la primera versión como una máquina de estados identificando para ello: - Los distintos estados del sistema: wait_start, wait_push, wait_end - Las posibles transiciones entre ellos - Las funciones de entrada (o de comprobación de eventos, e.g. EVENT_BUTTON), - y las funciones de salida (o de actualización del sistema) que gobiernan dichas transiciones
Recursos adicionales	• Programa ejemplo con fsm y proceso que corre rutina de detección de pulsaciones
Resultados para el proyecto	• Segunda versión del juego igualmente jugable desde la terminal y el teclado del PC pero basada en: - máquinas de estado. - codificación de eventos y flags de estado.

Tabla 3. Resumen del trabajo previo, objetivos, recursos y resultados de la cuarta sesión.

6.3. Sesiones 5 y 6: Versión 3.0 del juego

El objetivo de estas dos sesiones consiste en incorporar al juego el hardware necesario para visualizar el mismo en la matriz de leds.

En este sentido, la primera sesión puede dedicarse a diseñar, implementar y probar el HW de visualización conectándolo a los puertos GPIO de la Raspberry Pi. Para ello, se facilitará al alumno un programa que permitirá realizar una prueba básica de las conexiones. Asumiendo que se emplea una configuración como la del presente enunciado, este programa alternará en intervalos de 1 segundo el encendido y apagado de todos y cada uno de los leds que componen el display mostrando lo que podría considerarse un damero o tablero de ajedrez similar al representado en la Figura 6:

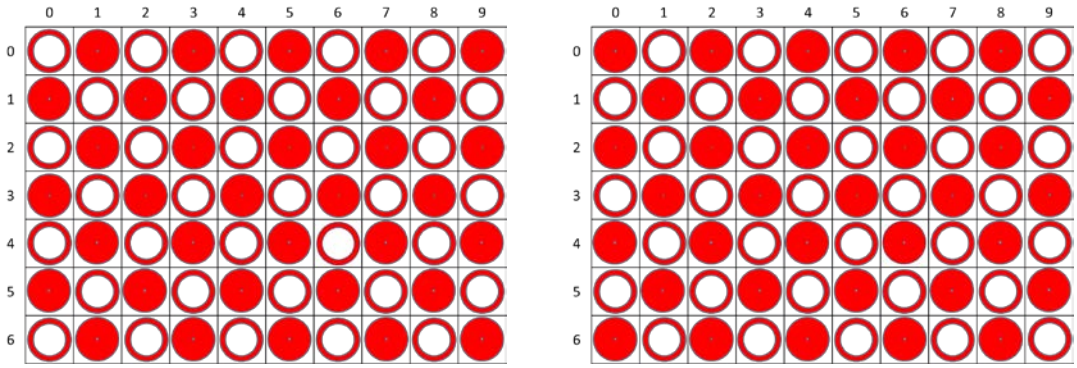

Figura 6. Detalle del programa para la validación del HW de visualización: estados mostrados por pantalla a intervalos de 1 segundo.

Durante la siguiente sesión el alumno deberá añadir a su propio sistema la temporización necesaria para refrescar correctamente la matriz aplicando para ello las excitaciones que resulten oportunas en el puerto de salida.

En particular, será necesario recorrer cíclicamente un autómata en anillo de 10 estados (10 es el número de columnas de leds). Cada columna estará iluminada durante 1 ms y a continuación se pasará a iluminar la siguiente columna. Una vez finalizado el periodo de excitación correspondiente a la última columna daremos comienzo nuevamente a un nuevo periodo de excitación de la primera. Será el contenido de la matriz asociada al área de juego el que determinará el conjunto de leds que es preciso iluminar en cada momento.

El proceso a seguir para cada una de las 10 columnas puede resumirse de la siguiente manera:

- recorreremos la columna de la matriz que estemos procesando (i.e. x), aquella que sea objeto de la excitación, comprobando las posiciones correspondientes a las diferentes filas (i.e. (x, 1), (x, 2),…),
- sólo en caso de que la posición (x, y) contenga un valor 1 significará que debemos iluminar el led asociado a la misma.

En este caso la prueba que nos permitirá verificar el correcto funcionamiento del sistema consistirá en comprobar que todos y cada uno de los leds de las filas y las columnas se encienden correctamente al enviar al puerto de salida los valores apropiados, sin que el teclado deje de funcionar y, por tanto, permitiendo el control del juego desde éste último. En este sentido, será importante comprobar con el osciloscopio la correcta generación de las señales de excitación aplicadas a las diferentes columnas de la matriz de leds, señales necesarias para su oportuno refresco.

Sesiones 5 y 6) Versión 3.0 del juego	
Trabajo previo	• Serán de obligada lectura [3] y [4] • Montaje HW necesario para excitar matriz de leds • Implementación previa de subrutinas de refresco • Versión 2.0 del sistema

Objetivos docentes / de aprendizaje	• Comprender el procedimiento de excitación y refresco empleado • Dimensionar e identificar los recursos (puertos GPIO) necesarios • Conocer los aspectos de configuración requeridos para un correcto funcionamiento • Verificar el correcto funcionamiento del montaje por medio del programa facilitado
Recursos adicionales	• Programa de test del HW implementado
Resultados para el proyecto	• Tercera versión del juego que incluye el HW y la temporización necesaria para el refresco del display de leds, y que por tanto es jugable en el display utilizando el teclado del PC

Tabla 4. Resumen del trabajo previo, objetivos, recursos y resultados de las sesiones 5 y 6.

6.4. Sesión 7: Versión 4.0 del juego

Durante la sesión 7 (correspondiente a la semana 8 del calendario académico) tendrá lugar una revisión intermedia obligatoria por parte de los profesores del estado del sistema en desarrollo. En este punto el alumno debería haber completado con éxito la versión 3.0 del juego.

En cuanto al hito a conseguir durante la sesión, este consistirá en añadir un par de pulsadores que permitan el control de la raqueta, prescindiendo así del teclado convencional del PC. Para ello, el alumno deberá implementar el hardware correspondiente a los pulsadores de control, conectarlo a los pines GPIO de la Raspberry Pi, configurar convenientemente los pines elegidos como entradas, y definir las correspondientes rutinas de atención a la interrupción ligadas respectivamente a cada interrupción externa. El alumno deberá verificar su correcto funcionamiento prestando especial atención a los posibles rebotes mecánicos que puedan producirse al accionar los pulsadores. Estos rebotes deberán ser eliminados por SW de modo que, si se realiza una única pulsación, el sistema interprete una única pulsación.

Sesión 7) Versión 4.0 del juego	
Trabajo previo	• Serán de obligada lectura [3] y [4] • Montaje HW necesario para los pulsadores • Versión 3.0 del sistema
Objetivos docentes / de aprendizaje	• Capacidad para configurar convenientemente las entradas de interrupción dedicadas (R interna de pull-up/pull-down, limitaciones entrenadora,…) • Capacidad para habilitar soluciones SW / HW frente al problema de rebotes presente en las entradas de interrupción
Recursos adicionales	-
Resultados para el proyecto	• Cuarta versión del juego, ya casi completo a falta del movimiento automático de la pelota, jugable desde el display de leds sin artefactos (parpadeo) y pulsadores de control para la raqueta totalmente funcionales

Tabla 5. Resumen del trabajo previo, objetivos, recursos y resultados de la séptima sesión.

6.5. Sesión 8: Versión 5.0 del juego

Finalmente, y para completar el sistema, en esta sesión se propone la implementación del movimiento automático de la pelota según el modelo de trayectorias detallado y conforme al ritmo o velocidad especificada. Para ello, el alumno deberá incorporar un nuevo temporizador que mida el tiempo que la pelota permanece en la misma posición y que provoque su paso a la siguiente posición, la que corresponda según su trayectoria, en caso de que dicho tiempo supere el umbral establecido.

Sesión 8) Versión 5.0 del juego	
Trabajo previo	• Será de obligada lectura [4] • Versión 4.0 del sistema
Objetivos docentes / de aprendizaje	• Capacidad para gestionar correctamente el acceso por parte de las rutinas de atención a la interrupción a los recursos compartidos por estas con el resto de procesos del sistema (exclusión mutua)
Recursos adicionales	-
Resultados para el proyecto	• Quinta versión del juego totalmente funcional que finalmente incorpora el movimiento automático de la pelota

Tabla 6. Resumen del trabajo previo, objetivos, recursos y resultados de la octava sesión.

6.6. Sesiones 9 y 10: Versión 6.0 del juego

La sexta versión del sistema, versión mejorada y por tanto opcional, propone incorporar hardware analógico (i.e. un potenciómetro) para el control de la raqueta. Con ese propósito, y una vez completado el tutorial de uso del ADC / DAC [6], además del mencionado HW será necesario incorporar un nuevo temporizador al sistema configurado convenientemente para realizar la lectura periódica del joystick (1 lectura cada 100 ms debería permitir un control fluido de la raqueta, tal y como se adelantó en el apartado 4.1) con el fin de determinar la posición a ocupar por la raqueta en el área de juego. A este respecto, será responsabilidad del alumno tomar las decisiones de diseño oportunas para discretizar los valores de tensión continuos entregados por el HW y su necesaria correspondencia con el conjunto de posibles posiciones en las que puede estar ubicada la raqueta dentro del área de juego. Una prueba final que realizar, especialmente importante en este caso, será comprobar que la visualización de los leds no se ve en modo alguno afectada por la inclusión del joystick.

Sesiones 9 y 10) Versión 6.0 del juego	
Trabajo previo	• Será de obligada lectura [6] • Montaje HW necesario para el joystick (potenciómetro + resistencia + conexiones pines GPIO) • Configuración RasPi para uso de SPI (sudo raspi-config + gpio load spi) • Versión 5.0 del sistema
Objetivos docentes / de aprendizaje	• Capacidad para controlar los conversores analógico-digital y digital-analógico disponibles en el entrenador ENT2004CF

	mediante un protocolo de comunicación serie (SPI) desde las entradas y salidas digitales del entrenador
Recursos adicionales	• Tutorial básico con ejemplos de manejo de DAC/ADC vía SPI
Resultados para el proyecto	• Nueva versión mejorada del sistema que permite controlar la raqueta mediante un joystick analógico

Tabla 7. Resumen del trabajo previo, objetivos, recursos y resultados de las sesiones 9 y 10.

6.7. Sesiones 11-12: Mejoras y puesta a punto del sistema final

Durante estas últimas sesiones se espera que el alumno ponga a punto la versión final de su sistema incorporando para ello diversas mejoras, de su elección, que le permitan alcanzar la máxima nota (recuerde que la implementación del proyecto propuesto cubriendo sólo las **especificaciones mínimas obligatorias** será valorada **con un máximo de 7 puntos; esta puntuación se podrá incrementar** hasta alcanzar la máxima nota, 10 puntos, **por medio de la realización de dichas mejoras).**

Sesiones 11-12) Mejoras y puesta a punto del sistema final	
Trabajo previo	• Versión 5.0 o 6.0 del sistema
Objetivos docentes / de aprendizaje	-
Recursos adicionales	• Descripción básica de posibles mejoras a implementar (disponible en este mismo documento)
Resultados para el proyecto	• Versión mejorada del sistema con nuevas funcionalidades a libre disposición o criterio del alumno

Tabla 8. Resumen del trabajo previo, objetivos, recursos y resultados de las sesiones 11, 12 y 13.

6.8. Mejoras con puntuación predefinida

Se trata de modificaciones sobre las especificaciones básicas del proyecto que pueden permitir al alumno alcanzar una calificación superior a 7 en la asignatura. En este apartado se incluyen algunas mejoras con una indicación sobre la máxima calificación que se podría obtener en cada una de ellas. Por simplicidad, se omite en este caso deliberadamente toda mención a la mejora de “control analógico de la raqueta”, también con puntuación predefinida (hasta 1 punto) y cuya realización forma parte de la planificación propuesta al alumno (tal y como puede comprobarse en el apartado 5.6).

Pausa de juego	
Descripción	La acción simultánea de ambos pulsadores de dirección pausará / reanudará el juego
Requisitos de implementación	Nuevo estado FSM
Puntuación	0,5 puntos

Efectos de sonido	
Descripción	Cada vez que la pelota golpee la pala, los ladrillos o las paredes se escuchará un efecto de sonido
Requisitos de implementación	Generación señal PWM + timeout duración efecto Al menos 2 frecuencias / duraciones diferentes
Puntuación	1 punto

Música de juego / reproducción de voz vía DAC	
Descripción	Durante el juego se podrá escuchar una melodía o mensajes hablados informando acerca del desarrollo de éste
Requisitos de implementación	Array de muestras PCM (frec. muestreo 8KHz) DAC + Filtro + Etapa potencia
Puntuación	1,5 puntos

Tabla 9. Mejoras con puntuación predefinida.

6.9. Otras mejoras

Las sugerencias recogidas en este apartado tienen un carácter meramente orientativo. La valoración de la dificultad (de diseño y de implementación) que se hace de cada una de ellas es igualmente orientativa. No obstante, y con carácter general, se valorarán especialmente aquellas mejoras que conlleven algún tipo de esfuerzo por parte del alumno a un mayor número de niveles. En este sentido, se tendrán en cuenta los 3 niveles de la arquitectura típica de cualquier sistema embebido: nivel software o de aplicación, nivel hardware, y nivel de software de sistema o nivel dedicado a la integración hardware-software, concretada a través de la figura de "driver" (i.e. controlador) SW para el control de un dispositivo HW.

De este modo, el modelo de valoración adoptado, resumido en la Figura 7, reflejará respectivamente el impacto de la mejora sobre:

- sólo el primero de los niveles: BAJA valoración, por tratarse de una mejora que sólo requiere modificar código;
- sólo los dos primeros niveles: valoración MEDIA, por tratarse de una mejora que, además de modificar el código, requiera del uso adicional de algún nuevo recurso de la Raspberry Pi (como por ejemplo otro temporizador) o de una configuración alternativa de los mismos (ya sean éstos HW o SW);
- todos los niveles anteriormente mencionados: valoración ALTA, por afectar a todos los niveles de diseño del sistema, aspecto para el que será esencial la incorporación de nuevo HW al sistema.

En cualquier caso, todas las mejoras propuestas (y las que puedan surgir por iniciativa del alumno) podrán llevarse a cabo de diversas maneras y con distintos grados de perfección por lo que la calificación final de las mismas también dependerá de factores como la originalidad y la calidad del trabajo.

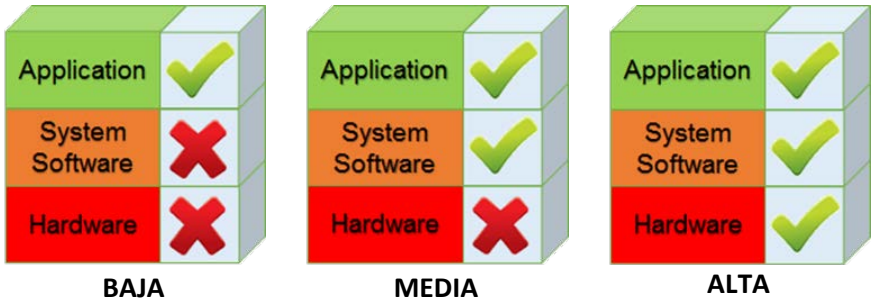

Figura 7. Niveles de valoración de las posibles mejoras a implementar.

Menús de configuración del juego	
Descripción	El objetivo es desarrollar un sistema de menús que permita al usuario seleccionar, entre otros: • Número de vidas • Velocidad de la pelota • Tamaño de la pala*** • Patrones de los ladrillos • Número de impactos mínimo requerido para hacer desaparecer un ladrillo • Histórico de puntuaciones • Perfil de usuario
Requisitos de implementación	-
Puntuación	BAJA

Nuevas características añadidas al juego	
Descripción	• Cada vez que un jugador elimina todos los ladrillos pasa al siguiente nivel en el que la dificultad aumenta: - Aparecen otros patrones de bloques - Aparecen ladrillos que hay que golpear varias veces para que desaparezcan - La velocidad a la que se desplaza la pelota aumenta • Aparición de "cápsulas" de mejora: - Que expanden la pala*** - Que la equipan con un "cañón láser" - Que permiten pasar directamente al siguiente nivel - Que aumentan el número de pelotas / vidas - Que permiten atrapar / ralentizar la pelota • Desplazamiento vertical de la pala por las paredes izquierda y derecha.
Requisitos de implementación	-
Puntuación	BAJA

Cambios sustanciales del juego / Nuevos juegos	
Descripción	• Modelo de rebotes más complejo*** - Tener en cuenta velocidad de desplazamiento de la pala. • Aparecen "naves" enemigas • Parpadeo de los ladrillos / pelota • Inclusión de mensajes en el display con efectos de desplazamiento (scroll) vertical u horizontal • Otros juegos (con el mismo HW): - Pong - Snake - Otros…
Requisitos de implementación	-
Puntuación	MEDIA

Nuevos elementos HW incorporados al sistema	
Descripción	• LCD (uso como marcador o similares) • Empleo de otros displays • Empleo de otros dispositivos de control - Mediante un acelerómetro • Esquemas circuitales alternativos • Otros juegos (con NUEVO HW): - Pong 2 jugadores - Otros…
Requisitos de implementación	-
Puntuación	ALTA

Tabla 10. Otras mejoras.

Para conocer detalles adicionales sobre la implementación o la valoración de éstas u otras mejoras, no dude en ponerse en contacto con el coordinador o con cualquiera de los profesores de la asignatura.

6.10. Fechas importantes

A continuación se facilita un calendario en el que se han destacado algunas **fechas relevantes** para el transcurso de la asignatura. Revíselas concienzudamente pues algunas de ellas afectan específicamente a los procedimientos de evaluación de la misma.

En la figura se ha añadido a la izquierda del calendario la planificación por sesiones recomendada y su correspondencia con las diferentes semanas del curso. Nótese que, debido a la acumulación de festivos a lo largo del curso, algunos grupos pueden alcanzar ciertas sesiones 1 semana antes o después que el resto.

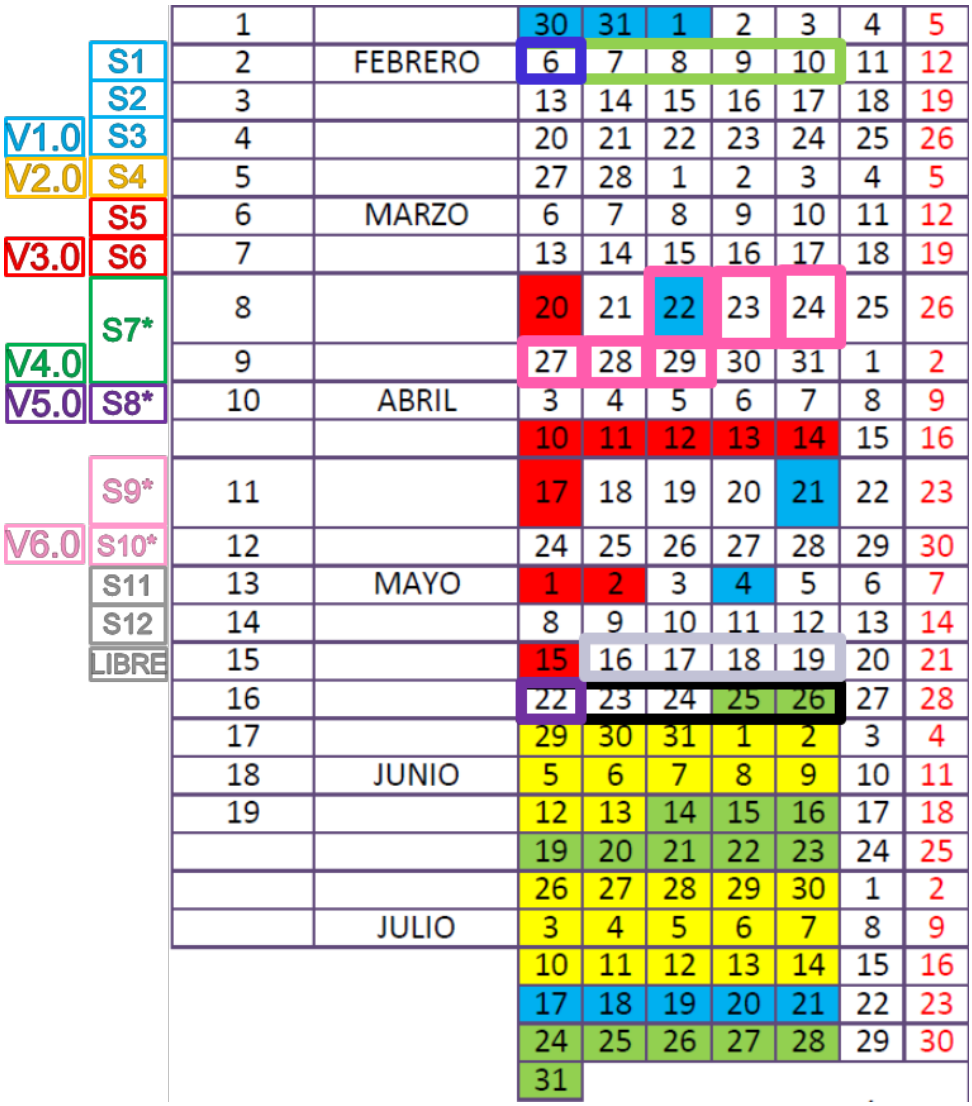

Figura 8. Calendario de fechas importantes para la asignatura.

Resumen de fechas importantes:

- **Semana 2: Comienzo sesiones**
  - **6 FEB: Presentación (12h, Salón de Actos Edificio A)**
- **Semana 8/9: Hito intermedio**
  - **Sobre entrega V3.0**
  - **22 MAR: Test TODOS (12h)**
  - **22 MAR: LIBRE (16-19h)**
  - **23 MAR: Oral JT**
  - **24 MAR: Oral VC**
  - **27 MAR: Oral LT**
  - **28 MAR: Oral MT**
  - **29 MAR: Oral XT**
- **21 ABR: S9 para LT**
- **4 MAY: S11 para LT**
- **Semana 15: LIBRE (16-19h)**
- **Semana 16: FINAL**
  - **22 MAY: Test TODOS (12h)**
  - **22-26 MAY: LT-VC**

Para cualquier duda en relación a los procedimientos y fechas indicadas, no dude en ponerse en contacto con el coordinador docente o administrativo de la asignatura.

6.11. Entregas electrónicas del código previstas

Las diferentes versiones a implementar del sistema exigirán la entrega vía Moodle del SW desarrollado. Concretamente, los alumnos deberán realizar una entrega del mismo al final de la última sesión dedicada a cada versión (e.g. primera entrega al finalizar la sesión 3).

En el calendario reflejado en la Figura 8 se ha añadido una columna adicional a la izquierda (e.g. “V1.0”) que hace referencia a la versión del sistema prevista a la finalización de cada semana y que indica, por tanto, la obligatoriedad de realizar una entrega del código desarrollado hasta ese momento independientemente de que cumpla o no los requisitos previstos.

En resumen, los alumnos **deberán realizar un total de 6 entregas**: 5 correspondientes a las diferentes versiones 1.0-5.0 del sistema, conforme a las especificaciones básicas y obligatorias, y una última entrega final con las posibles mejoras que se hayan podido añadir al mismo. Nótese que la V6.0, a pesar de aparecer indicada en el calendario, es opcional por lo que no será obligatoria su entrega.

7. Material complementario

[1] Introducción al entorno de desarrollo en C para Raspberry Pi
[2] Prácticas de lenguaje C para Raspberry Pi
[3] Iniciación al Manejo de las Entradas/Salidas del BCM 2835
[4] Manejo de temporizadores, interrupciones y procesos con la Raspberry Pi
[5] Introducción a las máquinas de estados en C para Raspberry Pi
[6] Manejo de DAC/ADC vía SPI

8. ANEXOS

8.1. Proyecto inicial: arkanoPi_1

Con objeto de facilitar el desarrollo del sistema se facilitará al alumno un proyecto y sus correspondientes fuentes que servirán de punto de partida para la implementación de la primera versión del mismo. Este proyecto proporcionará, entre otras, **la definición de los objetos SW básicos** (i.e. `typedef struct{…} tipo_objeto;`) necesarios para su implementación, incluyendo:

- la **raqueta**, la **pelota** y los **ladrillos**, objetos que contendrán la información característica de dichos elementos (información reflejada en la Figura 8),
- la **pantalla** (o área de juego), objeto que permitirá mostrar la raqueta, la pelota y los ladrillos en la matriz de leds (ver Figura 8); para ello contendrá información acerca de las dimensiones de la matriz y de la ocupación de las distintas posiciones que la conforman (correspondiente al encendido/apagado de cada uno de los leds);
- el **estado,** objeto que reflejará el estado en que se encuentra en todo momento el sistema,
- y finalmente el propio **juego,** objeto que define en todo momento las características y los elementos del juego y que garantiza la integridad del mismo conforme al conjunto de reglas o restricciones básicas definidas para éste (hace contendrá a todos los anteriores).

Igualmente, se proporcionan los **prototipos** (i.e. declaración de una función, disponible en el correspondiente fichero de cabecera) **de todas las funciones básicas** necesarias para la implementación de la primera versión del sistema, incluyendo:

- **InicializaArkanoPi**: método encargado de la inicialización de toda variable o estructura de datos específicamente ligada al desarrollo del juego y su visualización a través de la ventana de terminal.
- **InicializaJuego**: función encargada de llevar a cabo la oportuna inicialización del sistema incluyendo, en particular, la inicialización de aquellos elementos y recursos HW que resulten necesarios para el correcto desarrollo del juego.
- **MueveRaquetaIzquierda**: función encargada de ejecutar el movimiento hacia la izquierda contemplado para la raqueta. Debe garantizar la viabilidad del mismo mediante la comprobación de que la nueva posición correspondiente a la raqueta no suponga que esta rebase o exceda los límites definidos para el área de juego (i.e. al menos uno de los leds que componen la raqueta debe permanecer visible durante todo el transcurso de la partida).
- **MueveRaquetaDerecha**: función similar a la anterior encargada del movimiento hacia la derecha.
- **MovimientoPelota**: función encargada de actualizar la posición de la pelota conforme a la trayectoria definida para ésta. Para ello deberá identificar los posibles rebotes de la pelota para, en ese caso, modificar su correspondiente trayectoria (los rebotes detectados contra alguno de los ladrillos implicarán adicionalmente la eliminación del ladrillo). Del mismo modo, deberá también

identificar las situaciones en las que se dé por finalizada la partida: bien porque el jugador no consiga devolver la pelota, y por tanto esta rebase el límite inferior del área de juego, bien porque se agoten los ladrillos visibles en el área de juego.

- **CalculaLadrillosRestantes**: función encargada de evaluar el estado de ocupación del área de juego por los ladrillos y devolver el número de estos.
- **FinalJuego**: función encargada de mostrar en la ventana de terminal los mensajes necesarios para informar acerca del resultado del juego.
- **ReseteaJuego**: función encargada de llevar a cabo la reinicialización de cuantas variables o estructuras resulten necesarias para dar comienzo a una nueva partida.

Será tarea del alumno completar la definición de dichas funciones. Nótese que será igualmente responsabilidad del alumno completar la definición de aquellas otras funciones destinadas a actualizar el área de juego y visualizar el estado de ésta a través de la ventana de terminal o consola de la plataforma, en particular de:

- **ActualizaPantalla**: método cuya ejecución estará ligada a cualquiera de los movimientos de la raqueta o de la pelota y que será el encargado de actualizar convenientemente la estructura de datos en memoria que representa el área de juego y su correspondiente estado.
- **PintaMensajeInicialPantalla**: método encargado de aprovechar el display (o la terminal hasta que éste esté disponible) para presentar un mensaje de bienvenida al usuario.
- **PintaPantallaPorTerminal:** método encargado de mostrar el contenido o la ocupación de la matriz de leds en la ventana de terminal o consola. Este método será fundamental para facilitar la labor de depuración de errores (por ejemplo, en la programación de los diferentes movimientos tanto de la raqueta como de la pelota)**.**

Finalmente, se facilita también **una definición parcial del bucle infinito** mediante el cual el programa principal comprueba periódicamente si hay que cambiar de estado. Para ello es necesario llamar periódicamente la función ***kbhit*** para detectar si se ha producido la pulsación de alguna de las teclas del teclado, en cuyo caso se llama a la función ***kbread*** para recuperar la tecla pulsada (ambas funciones se facilitan íntegramente al alumno). Posteriormente se lleva a cabo la interpretación de la pulsación detectada en el contexto específico del estado en el que se encuentra el sistema. El alumno deberá completar el bucle añadiendo el código requerido y sustituyendo las expresiones señaladas.

Para poder completar con éxito la implementación del proyecto en el siguiente apartado se facilitan adicionalmente algunas **consideraciones de diseño básicas** acerca del modelo a seguir.

8.2. Aspectos básicos de diseño

Para gestionar la información relativa a la posición y movimiento de los diferentes elementos de juego, y dado el modelo matricial propuesto para el área de juego (construida como una matriz de leds), lo más sencillo es emplear un sistema de coordenadas cartesianas (x, y) que permita hacer referencia a cualquiera de las posiciones del mismo.

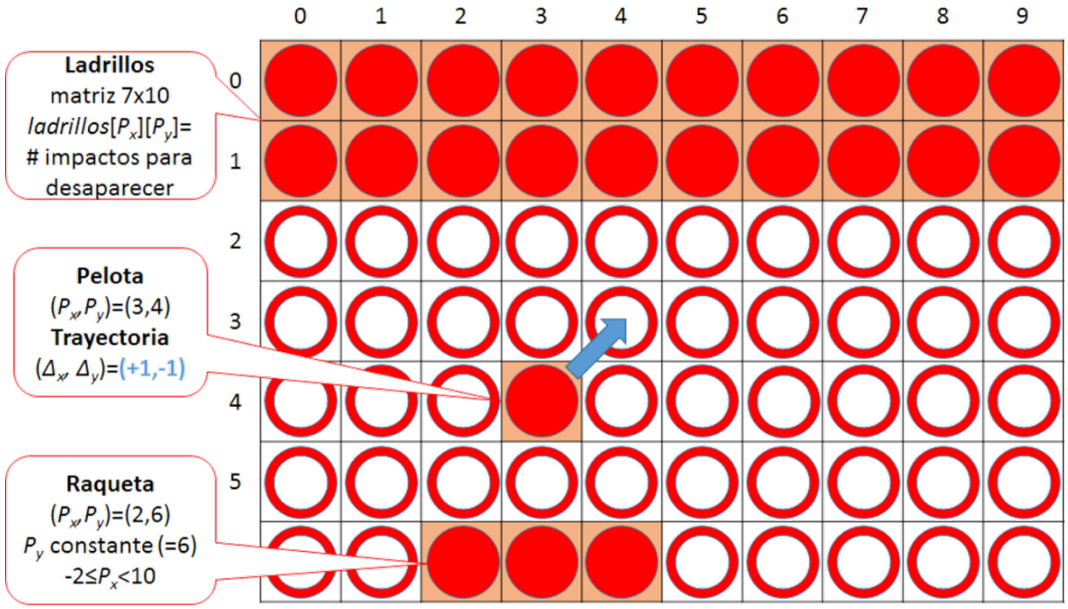

Figura 9. Detalle del objeto pantalla y relación con los objetos raqueta, pelota y ladrillos.

La Figura 8 representa, a modo de ejemplo, el área de juego con un posible estado de ocupación en el que hemos incluido la raqueta, la pelota y los ladrillos y en el que, por simplicidad, hemos hecho coincidir nuestro origen de coordenadas con la esquina superior izquierda del área de juego.

Nótese que la anchura y la altura que emplearemos para representar una raqueta serán respectivamente 1 y 3. Nótese igualmente que, dado que sólo se permiten desplazamientos horizontales de la raqueta, la coordenada de posición correspondiente al eje de abcisas (i.e. x) deberá permanecer constante durante todo el desarrollo del juego (i.e. Py = 6).

En relación al objeto ladrillos, cabe mencionar que, teniendo en cuenta que un ladrillo puede potencialmente aparecer en cualquier posición dentro de la pantalla, se ha optado por la opción más simple de modelar dicho objeto de igual modo que la propia pantalla (i.e. matriz de idénticas dimensiones en cuyas posiciones el valor 1 o 0 representa respectivamente la ocupación o no de dicha posición por un ladrillo).

Finalmente, respecto a la pelota, además de su posición, es preciso disponer información relativa a la trayectoria seguida por esta, información que permitirá actualizar su posición cuando corresponda. El conjunto de posibles trayectorias que podrá seguir la pelota durante el juego está definido en la Figura 9.

Tal y como puede observarse, dicho conjunto es bastante simple, aspecto este condicionado en parte por la 'limitada' resolución de nuestro sistema de visualización (por simplicidad, hemos prohibido los desplazamientos horizontales). En todo momento, la trayectoria seguida por la pelota nos dará la información necesaria para resolver cómo actualizar las coordenadas de posición de la misma.

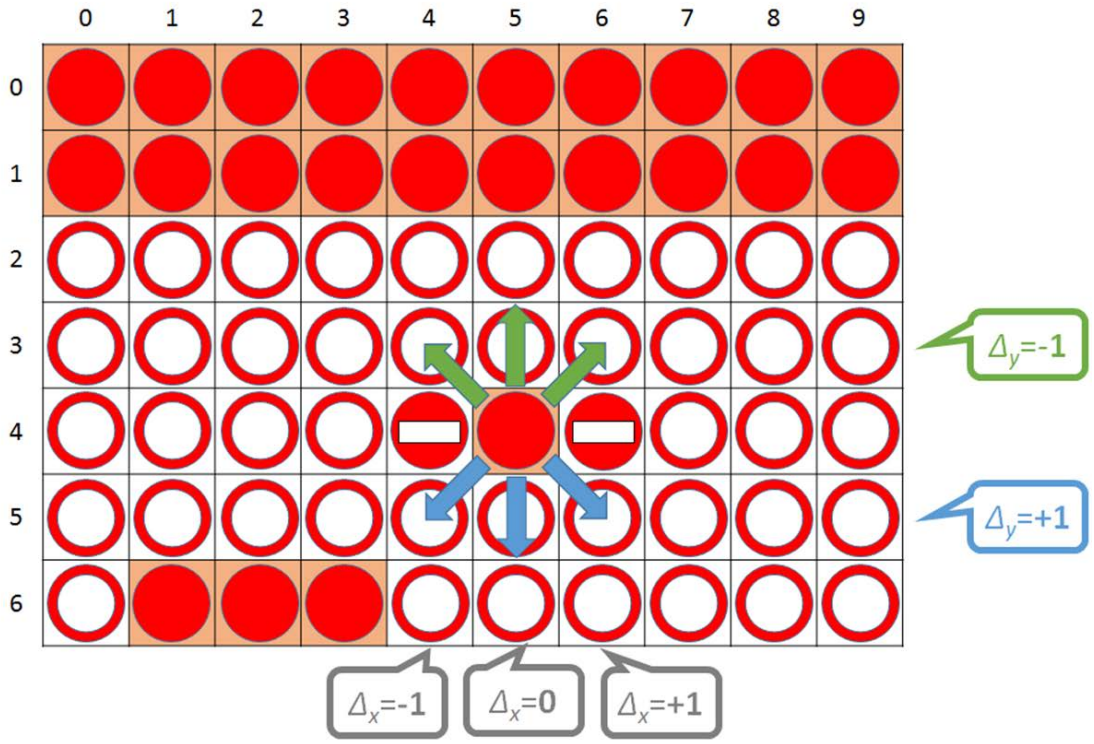

Figura 10. Modelo propuesto de posibles trayectorias que podrá seguir la pelota durante el juego.

Para la adecuada gestión de cualquier movimiento que pueda experimentar la pelota, resulta indispensable introducir el concepto de "rebote". Asumiremos que se produce un rebote toda vez que, como consecuencia del propio movimiento de la pelota:

- la pelota adquiera una posición que exceda o rebase los límites definidos para el área de juego (excepto cuando suceda por el lado que ocupa la raqueta, situación de final de juego).
- la pelota adquiera una posición que ya estuviera anteriormente ocupada por un ladrillo.
- o que la pelota adquiera una posición que ya estuviera anteriormente ocupada por la raqueta.

Siempre que se produzca un rebote será necesario modificar la trayectoria seguida por la pelota. Por simplicidad, asumiremos que para actualizar la trayectoria, definida por $(\Delta x, -\Delta y)$, tras cualquier rebote con una pared o ladrillo, simplemente será necesario cambiar el signo de Δx o Δy. Recogemos la casuística completa en la siguiente Tabla asumiendo que la trayectoria de la pelota previa al rebote es $(\Delta x_o, \Delta y_o)$:

Rebote con…	Nueva Trayectoria
Pared SUPERIOR / LADRILLO	$(\Delta x_f, \Delta y_f) = (\Delta x_o, -\Delta y_o)$
Pared DERECHA / IZQUIERDA	$(\Delta x_f, \Delta y_f) = (-\Delta x_o, \Delta y_o)$

Tabla 11. Actualización necesaria de la trayectoria de la pelota tras rebote con alguna pared.

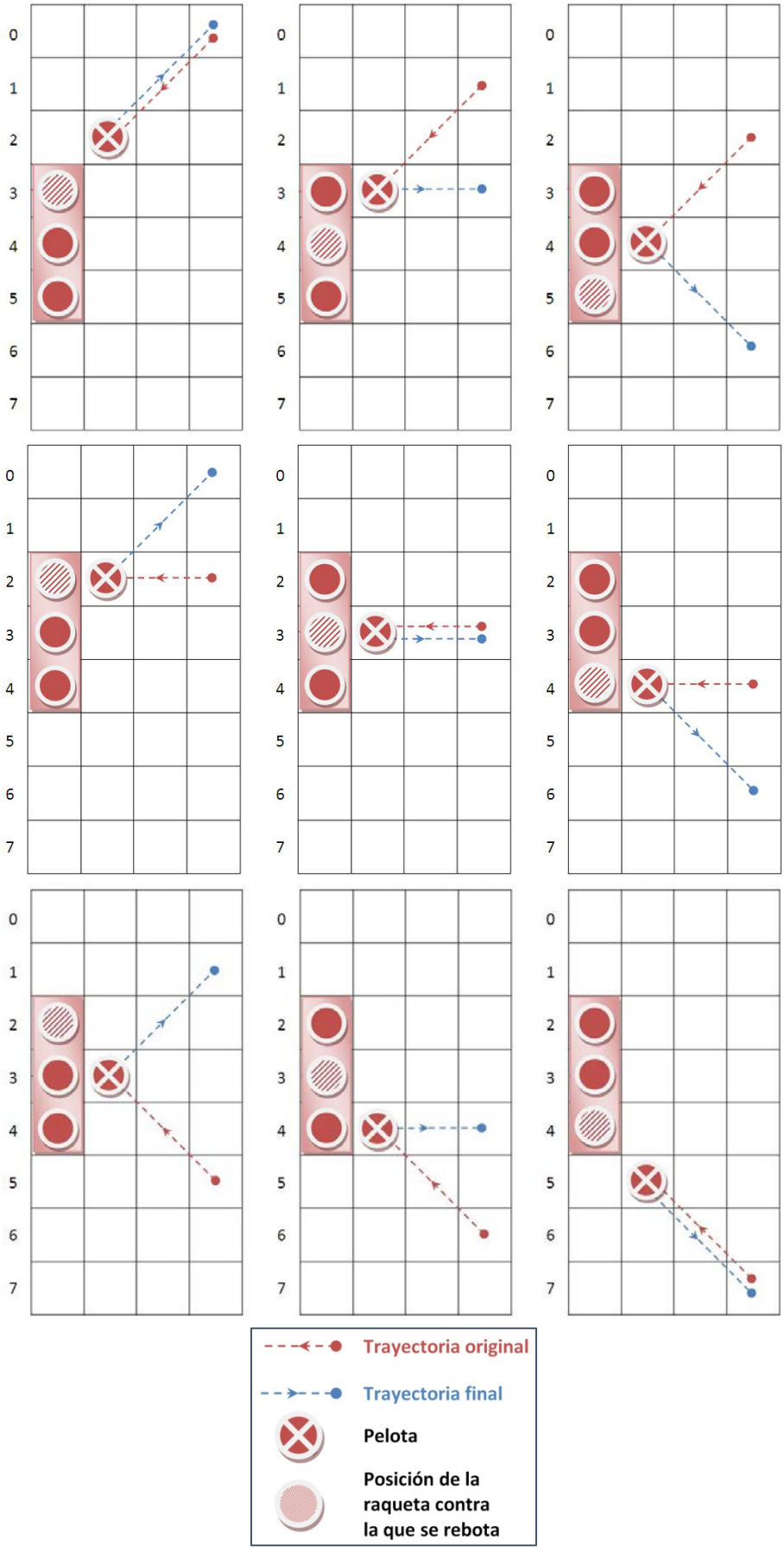

Figura 11. Modelo tentativo de rebotes contra la raqueta.

Por el contrario, los rebotes que se produzcan contra la raqueta van a precisar de un tratamiento especial. En particular, y con objeto de garantizar una cierta variabilidad en las posiciones a ocupar por la pelota dentro del área de juego (i.e. evitar una excesiva repetición de caminos seguidos por la pelota haciendo el juego más entretenido), obligaremos a que la nueva trayectoria de la pelota tras rebotar con la raqueta dependa de la posición de la misma contra la que se produzca específicamente el rebote. Para ello, se propone el modelo presentado en la Figura 10 en el que hemos recogido los 9 posibles casos correspondientes a las 3 posibles trayectorias previas al rebote y a las 3 posibles posiciones contra las que dicho rebote puede producirse (nótese que, para una representación más compacta, las situaciones reflejadas en la figura están giradas 90º con respecto a la orientación real con la que se observan en el display).

Las definiciones de los objetos mencionados anteriormente están incluidas en los ficheros de cabecera, arkanoPi_1.h y arkanoPiLib.h entregados como parte del proyecto inicial, y están acompañadas de la declaración y definición de las respectivas funciones que nos permiten inicializar convenientemente cada estructura de datos en memoria.

8.3. arkanoPi_1.c

```
#include "arkanoPi_1.h"

static volatile tipo_juego juego;

//------------------------------------------------------
// FUNCIONES DE ACCION
//------------------------------------------------------

// void InicializaJuego (void): funcion encargada de llevar a cabo
// la oportuna inicialización de toda variable o estructura de datos
// que resulte necesaria para el desarrollo del juego.
void InicializaJuego (void) {
	// A completar por el alumno...

}

// void MueveRaquetaIzquierda (void): funcion encargada de ejecutar
// el movimiento hacia la izquierda contemplado para la raqueta.
// Debe garantizar la viabilidad del mismo mediante la comprobación
// de que la nueva posición correspondiente a la raqueta no suponga
// que ésta rebase o exceda los límites definidos para el área de
// juego (i.e. al menos uno de los leds que componen la raqueta debe
// permanecer visible durante todo el transcurso de la partida).
void MueveRaquetaIzquierda (void) {
	// A completar por el alumno...

}

// void MueveRaquetaDerecha (void): función similar a la anterior
// encargada del movimiento hacia la derecha.
void MueveRaquetaDerecha (void) {
	// A completar por el alumno...

}

// void MovimientoPelota (void): función encargada de actualizar la
// posición de la pelota conforme a la trayectoria definida para ésta.
// Para ello deberá identificar los posibles rebotes de la pelota
// para, en ese caso, modificar su correspondiente trayectoria (los
// rebotes detectados contra alguno de los ladrillos implicarán
// adicionalmente la eliminación del ladrillo). Del mismo modo, deberá
// también identificar las situaciones en las que se dé por finalizada
// la partida: bien porque el jugador no consiga devolver la pelota, y
// por tanto ésta rebase el límite inferior del área de juego, bien
// porque se agoten los ladrillos visibles en el área de juego.
void MovimientoPelota (void) {
	// A completar por el alumno...

}

// void FinalJuego (void): función encargada de mostrar en la ventana
// de terminal los mensajes necesarios para informar acerca del
// resultado del juego.
void FinalJuego (void) {
	// A completar por el alumno...

}

//void ReseteaJuego (void): función encargada de llevar a cabo la
```

```
// reinicialización de cuantas variables o estructuras resulten
// necesarias para dar comienzo a una nueva partida.
void ReseteaJuego (void) {
	// A completar por el alumno...

}

//-----------------------------------------------------
// FUNCIONES DE INICIALIZACION
//-----------------------------------------------------

// int systemSetup (void): procedimiento de configuracion del sistema.
// Realizará, entra otras, todas las operaciones necesarias para:
// configurar el uso de posibles librerías (e.g. Wiring Pi),
// configurar las interrupciones externas asociadas a los pines GPIO,
// configurar las interrupciones periódicas y sus correspondientes
// temporizadores, crear, si fuese necesario, los threads adicionales
// que pueda requerir el sistema
int systemSetup (void) {
	// A completar por el alumno...

}

int main ()
{
	// Configuracion e inicializacion del sistema
	// A completar por el alumno...

	while (1) {
		if(kbhit()) { // Funcion que detecta si se ha producido
pulsacion de tecla alguna
			juego.teclaPulsada = kbread(); // Funcion que
devuelve la tecla pulsada

			// Interpretacion de las pulsaciones para cada
			// posible estado del sistema
			if( juego.estado == WAIT_START ) { // Cualquier
pulsacion da comienzo al juego...
				// Descomente ambas lineas y sustituya cada
				// etiqueta XXXXXXX por lo que corresponda en
				// cada caso...
				// XXXXXXX();
				// juego.estado = XXXXXXX;
			}
			else if( juego.estado == WAIT_END ) { // Cualquier
nos devuelve al estado inicial...
				// Descomente ambas lineas y sustituya cada
				// etiqueta XXXXXXX por lo que corresponda en
				// cada caso...
				// XXXXXXX();
				// juego.estado = XXXXXXX;
			}
			else { // Si estamos jugando...
				switch(juego.teclaPulsada) {
					case 'i':
						// A completar por el alumno...

						break;

					case 'o':
						// A completar por el alumno...
```

```
                                    break;

                              case 'p':
                                    // A completar por el alumno...

                                    break;

                              case 'q':
                                    exit(0);
                                    break;

                              default:
                                    printf("INVALID KEY!!!\n");
                                    break;
                        }
                  }
            }
      }
}
```

8.4. arkanoPi_1.h

```
#ifndef _ARKANOPI_H_
#define _ARKANOPI_H_

#include <time.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/time.h>
#include <wiringPi.h>

#include "kbhit.h" // para poder detectar teclas pulsadas sin bloqueo
y leer las teclas pulsadas

#include "arkanoPiLib.h"

typedef enum {
      WAIT_START,
      WAIT_PUSH,
      WAIT_END} tipo_estados_juego;

typedef struct {
      tipo_arkanoPi arkanoPi;
      tipo_estados_juego estado;
      char teclaPulsada;
} tipo_juego;

//-------------------------------------------------------
// FUNCIONES DE ACCION
//-------------------------------------------------------

void InicializaJuego (void);
void MueveRaquetaIzquierda (void);
void MueveRaquetaDerecha (void);
void MovimientoPelota (void);
void FinalJuego (void);
void ReseteaJuego (void);

//-------------------------------------------------------
// FUNCIONES DE INICIALIZACION
//-------------------------------------------------------
int systemSetup (void);

#endif /* ARKANOPI_H_ */
```

8.5. arkanoPiLib.c

```
#include "arkanoPiLib.h"

int ladrillos_basico[MATRIZ_ANCHO][MATRIZ_ALTO] = {
            {1,1,0,0,0,0,0},
            {1,1,0,0,0,0,0},
            {1,1,0,0,0,0,0},
            {1,1,0,0,0,0,0},
            {1,1,0,0,0,0,0},
            {1,1,0,0,0,0,0},
            {1,1,0,0,0,0,0},
            {1,1,0,0,0,0,0},
            {1,1,0,0,0,0,0},
            {1,1,0,0,0,0,0},
};

//------------------------------------------------------
// FUNCIONES DE INICIALIZACION / RESET
//------------------------------------------------------

void ReseteaMatriz(tipo_pantalla *p_pantalla) {
      int i, j = 0;

      for(i=0;i<MATRIZ_ANCHO;i++) {
            for(j=0;j<MATRIZ_ALTO;j++) {
                  p_pantalla->matriz[i][j] = 0;
            }
      }
}

void ReseteaLadrillos(tipo_pantalla *p_ladrillos) {
      int i, j = 0;

      for(i=0;i<MATRIZ_ANCHO;i++) {
            for(j=0;j<MATRIZ_ALTO;j++) {
                  p_ladrillos->matriz[i][j] = ladrillos_basico[i][j];
            }
      }
}

void ReseteaPelota(tipo_pelota *p_pelota) {
      // Pelota inicialmente en el centro de la pantalla
      p_pelota->x = MATRIZ_ANCHO/2 - 1;
      p_pelota->y = MATRIZ_ALTO/2;

      // Trayectoria inicial
      p_pelota->y = MATRIZ_ALTO/2-1;
      p_pelota->xv = -1;
      p_pelota->yv = 1;
      p_pelota->xv = 0;
}

void ReseteaRaqueta(tipo_raqueta *p_raqueta) {
      // Raqueta inicialmente en el centro de la pantalla
      p_raqueta->x = MATRIZ_ANCHO/2 - p_raqueta->ancho/2;
      p_raqueta->y = MATRIZ_ALTO - 1;
      p_raqueta->ancho = RAQUETA_ANCHO;
      p_raqueta->alto = RAQUETA_ALTO;
}
```

```
//--------------------------------------------------------
// FUNCIONES DE VISUALIZACION (ACTUALIZACION DEL OBJETO PANTALLA QUE
LUEGO USARA EL DISPLAY)
//--------------------------------------------------------

// void PintaMensajeInicialPantalla (...): metodo encargado de
// aprovechar el display para presentar un mensaje de bienvenida al
// usuario
void PintaMensajeInicialPantalla (tipo_pantalla *p_pantalla,
tipo_pantalla *p_pantalla_inicial) {
      // A completar por el alumno...

}

// void PintaPantallaPorTerminal (...): metodo encargado de mostrar
// el contenido o la ocupación de la matriz de leds en la ventana de
// terminal o consola. Este método sera fundamental para facilitar
// la labor de depuración de errores (por ejemplo, en la programación
// de los diferentes movimientos tanto de la raqueta como de la
// pelota).
void PintaPantallaPorTerminal  (tipo_pantalla *p_pantalla) {
      // A completar por el alumno...

}

// void PintaLadrillos(...): funcion encargada de “pintar” los
// ladrillos en sus correspondientes posiciones dentro del área de
// juego
void PintaLadrillos(tipo_pantalla *p_ladrillos, tipo_pantalla
*p_pantalla) {
      int i, j = 0;

      for(i=0;i<MATRIZ_ANCHO;i++) {
            for(j=0;j<MATRIZ_ALTO;j++) {
                  p_pantalla->matriz[i][j] = p_ladrillos->matriz[i][j];
            }
    }
}

// void PintaRaqueta(...): funcion encargada de “pintar” la raqueta
// en su posicion correspondiente dentro del área de juego
void PintaRaqueta(tipo_raqueta *p_raqueta, tipo_pantalla *p_pantalla)
{
      int i, j = 0;

      for(i=0;i<RAQUETA_ANCHO;i++) {
            for(j=0;j<RAQUETA_ALTO;j++) {
                  if (( (p_raqueta->x+i >= 0) && (p_raqueta->x+i <
MATRIZ_ANCHO) ) &&
                              ( (p_raqueta->y+j >= 0) && (p_raqueta-
>y+j < MATRIZ_ALTO) ))
                        p_pantalla->matriz[p_raqueta->x+i][p_raqueta-
>y+j] = 1;
            }
      }
}

// void PintaPelota(...): funcion encargada de “pintar” la pelota
// en su posicion correspondiente dentro del área de juego
void PintaPelota(tipo_pelota *p_pelota, tipo_pantalla *p_pantalla) {
      if( (p_pelota->x >= 0) && (p_pelota->x < MATRIZ_ANCHO) ) {
```

```
            if( (p_pelota->y >= 0) && (p_pelota->y < MATRIZ_ALTO) ) {
                  p_pantalla->matriz[p_pelota->x][p_pelota->y] = 1;
            }
            else {
                  printf("\n\nPROBLEMAS!!!! posicion y=%d de la pelota
INVALIDA!!!\n\n", p_pelota->y);
                  fflush(stdout);
            }
      }
      else {
            printf("\n\nPROBLEMAS!!!! posicion x=%d de la pelota
INVALIDA!!!\n\n", p_pelota->x);
            fflush(stdout);
      }
}

// void ActualizaPantalla(...): metodo cuya ejecucion estara ligada a
// cualquiera de los movimientos de la raqueta o de la pelota y que
// sera el encargado de actualizar convenientemente la estructura de
// datos en memoria que representa el área de juego y su
// correspondiente estado.
void ActualizaPantalla(tipo_arkanoPi* p_arkanoPi) {
    // Borro toda la pantalla
      ReseteaMatriz((tipo_pantalla*)(&(p_arkanoPi->pantalla)));

      // A completar por el alumno...

}

// void InicializaArkanoPi(...): metodo encargado de la inicialización
// de toda variable o estructura de datos especificamente ligada al
// desarrollo del juego y su visualizacion.
void InicializaArkanoPi(tipo_arkanoPi *p_arkanoPi) {

      ReseteaMatriz((tipo_pantalla*)(&(p_arkanoPi->pantalla)));

      ReseteaLadrillos((tipo_pantalla*)(&(p_arkanoPi->ladrillos)));
      ReseteaPelota((tipo_pelota*)(&(p_arkanoPi->pelota)));
      ReseteaRaqueta((tipo_raqueta*)(&(p_arkanoPi->raqueta)));

      // A completar por el alumno...

}

// int CalculaLadrillosRestantes(...): función encargada de evaluar
// el estado de ocupacion del area de juego por los ladrillos y
// devolver el numero de estos
int CalculaLadrillosRestantes(tipo_pantalla *p_ladrillos) {
      int num_ladrillos_restantes = 0;

      // A completar por el alumno...

      return num_ladrillos_restantes;
}
```

8.6. arkanoPiLib.h

```
#ifndef _ARKANOPILIB_H_
#define _ARKANOPILIB_H_

#include <stdio.h>

// CONSTANTES DEL JUEGO
#define MATRIZ_ANCHO    10
#define MATRIZ_ALTO     7
#define LADRILLOS_ANCHO 10
#define LADRILLOS_ALTO  2
#define RAQUETA_ANCHO         3
#define RAQUETA_ALTO          1

typedef struct {
      // Posicion
      int x;
      int y;
      // Forma
      int ancho;
      int alto;
} tipo_raqueta;

typedef struct {
      // Posicion
      int x;
      int y;
      // Trayectoria
      int xv;
      int yv;
} tipo_pelota;

typedef struct {
      // Matriz de ocupación de las distintas posiciones que conforman
el display
      // (correspondiente al estado encendido/apagado de cada uno de
los leds)
      int matriz[MATRIZ_ANCHO][MATRIZ_ALTO];
} tipo_pantalla;

typedef struct {
  tipo_pantalla ladrillos; // Notese que, por simplicidad, los
ladrillos comparten tipo con la pantalla
  tipo_pantalla pantalla;
  tipo_raqueta raqueta;
  tipo_pelota pelota;
} tipo_arkanoPi;

extern tipo_pantalla pantalla_inicial;

//-------------------------------------------------------
// FUNCIONES DE INICIALIZACION / RESET
//-------------------------------------------------------
void ReseteaMatriz(tipo_pantalla *p_pantalla);
void ReseteaLadrillos(tipo_pantalla *p_ladrillos);
void ReseteaPelota(tipo_pelota *p_pelota);
void ReseteaRaqueta(tipo_raqueta *p_raqueta);

//-------------------------------------------------------
```

```
// FUNCIONES DE VISUALIZACION (ACTUALIZACION DEL OBJETO PANTALLA QUE
LUEGO USARA EL DISPLAY)
//------------------------------------------------------
void PintaMensajeInicialPantalla (tipo_pantalla *p_pantalla,
tipo_pantalla *p_pantalla_inicial);
void PintaPantallaPorTerminal (tipo_pantalla *p_pantalla);
void PintaLadrillos(tipo_pantalla *p_ladrillos, tipo_pantalla
*p_pantalla);
void PintaRaqueta(tipo_raqueta *p_raqueta, tipo_pantalla *p_pantalla);
void PintaPelota(tipo_pelota *p_pelota, tipo_pantalla *p_pantalla);
void ActualizaPantalla(tipo_arkanoPi* p_arkanoPi);

void InicializaArkanoPi(tipo_arkanoPi *p_arkanoPi);
int CalculaLadrillosRestantes(tipo_pantalla *p_ladrillos);

#endif /* _ARKANOPILIB_H_ */
```