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

Memoria de las

mejoras realizadas

Curso 2020/2021

"Arcade Pi"

Autores (orden alfabético): Andrés Álvarez de Cienfuegos Cremades

Cristina Conforto López

Código de la pareja: JT-02

ÍNDICE GENERAL

1 Introducción

Hemos querido aprovechar la oportunidad que ofrecía la asignatura para poder realizar todo el proyecto en una Raspberry Pi en lugar de en una máquina virtual. El modelo empleado ha sido una Rapsberry Pi 3 B. Todo el proyecto ha sido desarrollado empleando la IDE VS Code, haciendo uso del debugger que ofrece el mismo y usando la extensión LiveShare para poder permitirnos a ambos componentes del grupo trabajar sobre el proyecto simultáneamente.

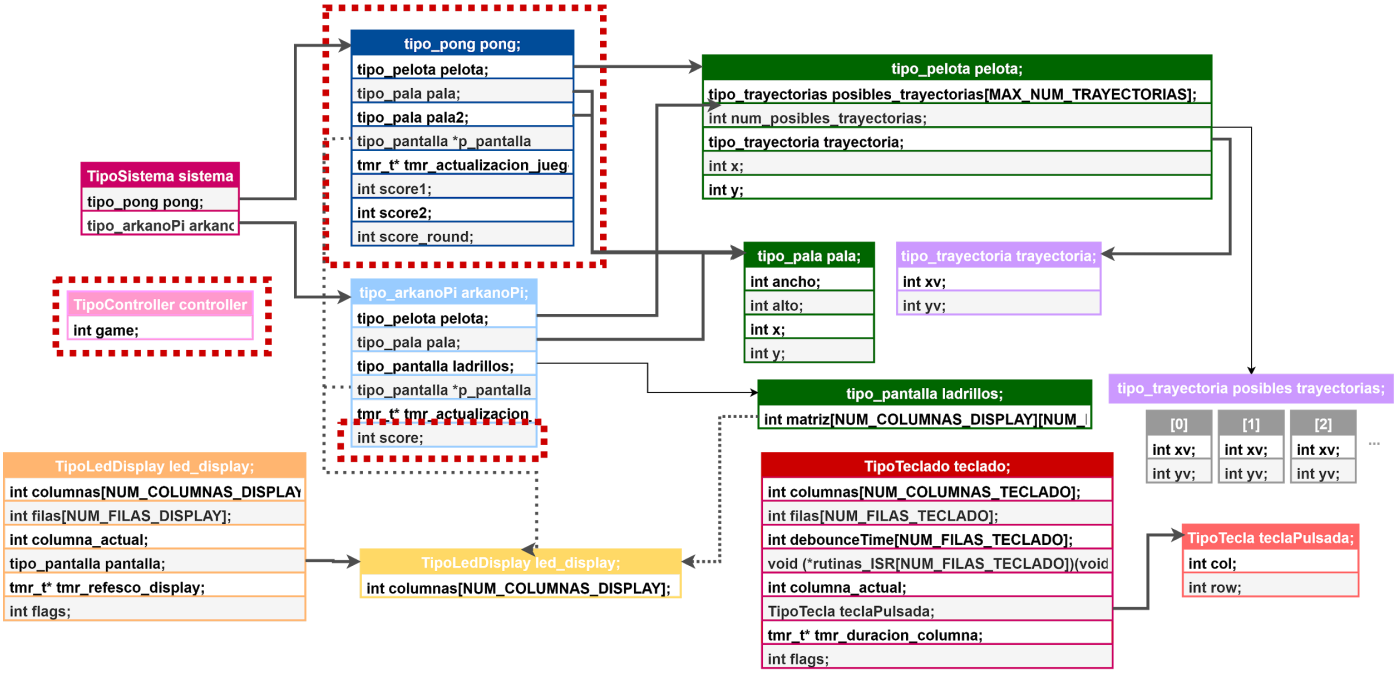
Partiendo del juego ArkanoPi, hemos querido evolucionar alrededor de éste ofreciendo nuevas funcionalidades. En cuanto al Hardware, el proyecto consiste en un teclado para jugar, un display donde se proyecta el juego y adicionalmente un display auxiliar LED 8x8, conectado empleando el protocolo de comunicación SPI.

Con respecto al Software, hemos decidido ampliar los juegos. Ahora cuando se quiera comenzar a jugar, tendremos una pantalla inicial en ambos display que nos permitirá seleccionar, a través del teclado, entre dos juegos: ArkanoPi y PingPong para dos jugadores. Una vez se seleccione uno de los dos, se puede comenzar a jugar, pudiendo ver en el display auxiliar los puntos que se van obteniendo. Independientemente del juego en el que se esté, se puede pausar, continuar y, adicionalmente, salir del juego y volver a la pantalla inicial. También se ha implementado un sistema de puntuación.

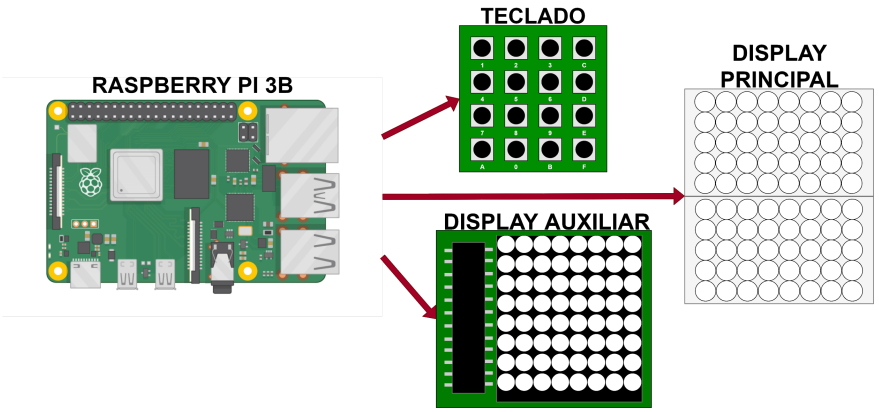
A lo largo de los distintos puntos de esta memoria se explicará con mayor profundidad cómo se han implementado todas las mejoras. En el siguiente enlace, puede encontrar un video demostrativo del juego: https://drive.google.com/file/d/19p6xUoSwdOIaQ9_WGFB6K7ryNIEvk8ee/view

El código del proyecto se puede encontrar en el siguiente repositorio de GitHub: https://github.com/aacienfuegos/arcadePi

2 Diagrama de subsistemas ampliado



En la imagen superior, se puede ver el diagrama del subsistema software con los objetos nuevos creados marcados en rojo. En cuanto a la mejoras hardware, se puede ver en la imagen inferior un pequeño esquemático de los componentes empleados.



3 Mejora 1: Implementación HW Teclado y Display

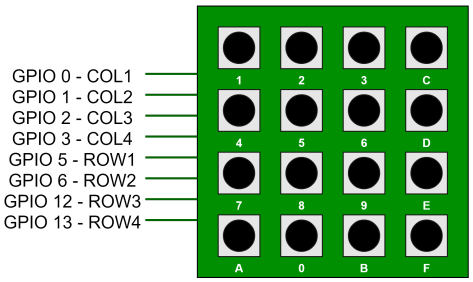
3.1 Objetivos de la mejora

El objetivo de esta mejora es implementar el teclado y el display cuyo código forma parte de la parte obligatoria para poder jugar únicamente empleando el microcontrolador. Se hará una breve explicación únicamente del hardware correspondiente a estos elementos.

3.2 Descripción del subsistema Hardware

3.2.1 Descripción del módulo Teclado

En la imagen lateral se puede ver la conexión realizada del teclado a la Raspberry Pi. Dado que nuestro modelo de teclado no cuenta con conexiones de tensión, no ha sido necesario emplear reguladores. Dado que la conexión era sencilla, no hemos encontrado ningún problema en su implementación ni ha sido necesaria el uso de ningún componente adicional.

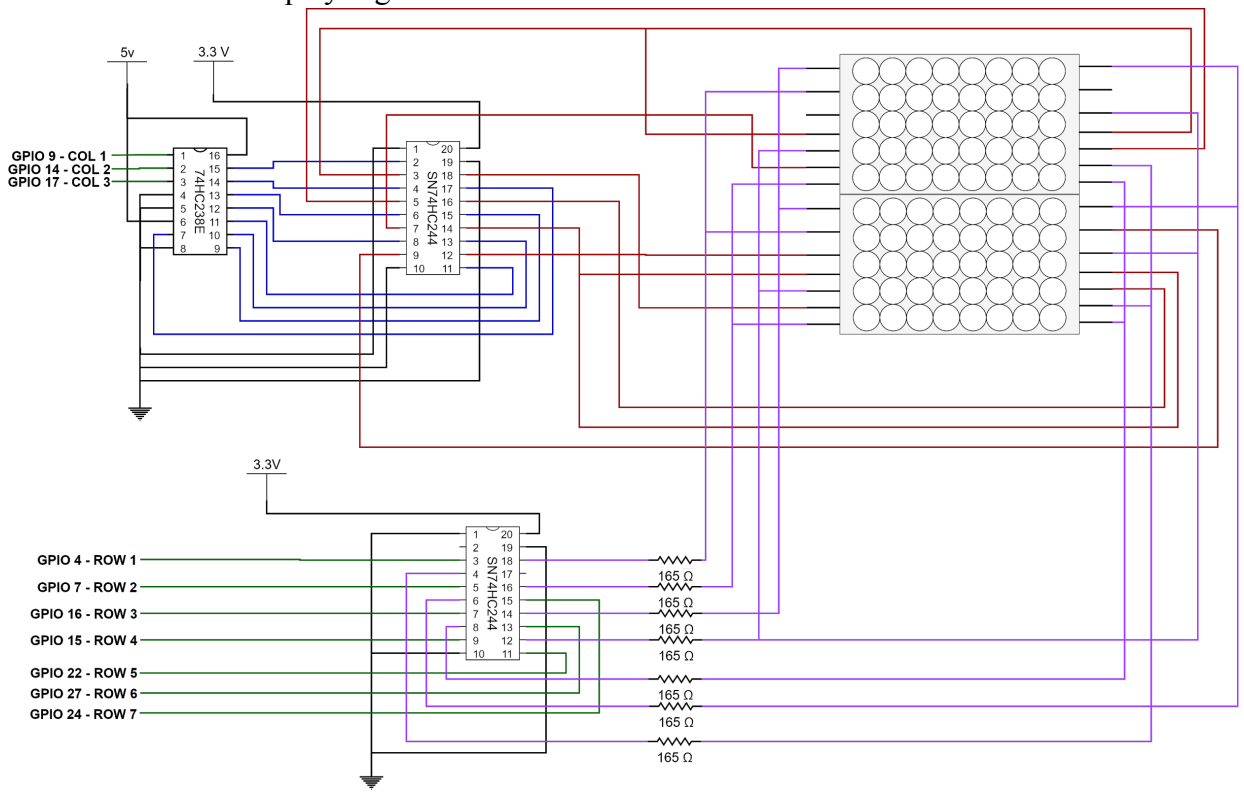


3.2.2 Descripción del módulo Display

En la imagen inferior, se pueden ver las conexiones realizadas para la conexión del display donde se proyecta el juego a la Raspberry Pi.

Empezaremos hablando sobre la conexión de las columnas. Para no tener que emplear 8 pines de la Raspberry, se ha empleado un decodificador de 3 a 8 bits, componente 74HC238E. Éste debía ser alimentado a 5V en el puerto de Vcc para su correcto funcionamiento según la hoja de especificaciones. Por otra parte, si nos fijamos en la tabla lógica de la hoja de especificaciones, podemos ver que para poder obtener una correcta decodificación a la salida de cualquier número del 0 al 7, los puertos E1 y E2 se deben encontrar a low, por lo que están conectados a tierra, mientras que el E3 se debe encontrar a high, por lo que está conectado a 5V. Adicionalmente, se conecta a ground el puerto GND. Las salidas de este componente se pueden encontrar en azul en el esquema. Posteriormente, se conectan todas las salidas a un buffer, componente SN74HC244, por seguridad. Para que obtengamos a la salida lo mismo que a la entrada, es necesario conectar los puertos 1OE y 2OE a tierra, junto con el puerto GND. La alimentación Vcc se debe conectar a 3.3V, siguiendo lo indicado en la hoja de especificaciones. Por último, se conectan las salidas del buffer al display,

poniendo las primeras 5 salidas en las 5 columnas del display primero y las 3 salidas restantes a las 3 primeras columnas del display segundo.



Con respecto a la conexión a las filas, al igual que antes se emplea un buffer, componente SN74HC244, por seguridad. Para limitar la tensión que llega al display y evitar que éste se deteriore, hemos empleado una serie de resistencias. Éstas han sido calculadas teniendo en cuenta la tensión de entrada, 3.3 V y la máxima corriente que para por los diodos que forman el display, 20mA sabiendo que estos están posteriormente conectados a tierra. Como se ve en la fórmula inferior, obtenemos un valor de 165 Ω. Las resistencias empleadas son de $R = 160 \pm 5\%\ \Omega$.

$$R = \frac{V_{cc} - 0V}{I_{diodo}}\ ;\ R = \frac{3.3V - 0V}{20\times10^{-3}A} = 165\ \Omega$$

Conectamos por último, las salidas por orden a cada una de las filas de cada display, para conseguir que ambos displays parezcan ser uno solo.

Para comprobar que las conexiones eran correctas, fuimos probando excitando led por led del display (la correspondiente fila y columna) y ajustando en caso de que no se iluminara el led esperado.

4 Mejora 2: Botón de Pausa/Continuar

4.1 Objetivos de la mejora

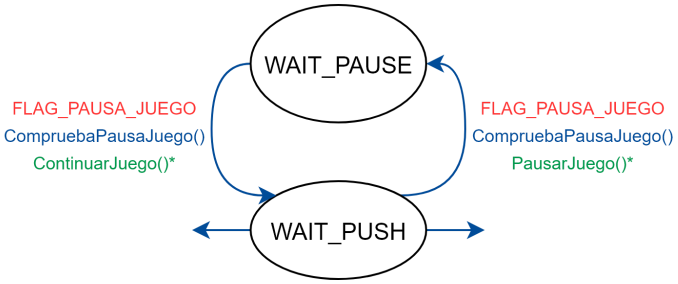
Desarrollamos un botón de pausa con el fin de poder parar el juego durante cierto tiempo. Ésto será de gran utilidad si en medio de un juego el jugador necesita detener la partida sin necesidad de tener que perder esa ronda.

4.2 Descripción del subsistema Software

La implementación de este botón se realiza mediante la adición de un estado extra al juego (*WAIT_PAUSE*). El juego entrará en este estado una vez pulsada la tecla de pausa y se mantendrá en él hasta que se vuelva a pulsar esta tecla. Esta implementación nos permite hacer un uso mucho más efectivo de la memoria que soluciones alternativas como el uso de una variable booleana. Además el uso de máquinas de estado proporciona una solución más simple (no hay necesidad de escribir *if-statements* para cada función) y sencilla de entender (con un vistazo a la máquina de estados observamos la presencia de este estado de pausa).

4.2.1 Flujo de ejecución del programa principal

Una vez el usuario pulsa la tecla de pausa, el teclado activa la flag *FLAG_PAUSA_JUEGO*, que será la responsable de comunicar al juego que se ha pulsado esta tecla.



La prueba más realizada ha sido la de usar puntos de debug que nos permitan conocer a qué puntos de la ejecución no se está llegando. De esta forma, pudimos comprobar la necesidad de llamar a la función de actualización del juego cuando queremos reanudar el juego (se explica con detalle en la función de salida *ContinuarJuego()*).

4.2.1.1 *Función de entrada o comprobación*

Se trata de una misma función para ambas direcciones en la que se comprueba el estado de las flags del sistema (con key *SYSTEM_FLAGS_KEY*) para ver si se ha activado la *FLAG_PAUSA_JUEGO*. De ser así, la función devuelve el valor de 1 y activa la función correspondiente de salida de la máquina de estados.

4.2.1.2 *Funciones de salida o actuación*

Pese a que estas funciones son específicas para cada uno de los juegos, la implementación es muy similar para ambos. Por ello se explicarán las funciones desde un punto de vista genérico:

- ***PausarJuego(fsm_t* this):*** Primero reseteamos la flag de pausa (*FLAG_PAUSA_JUEGO*) y, luego, cargamos en pantalla (*p_pantalla*) la pantalla de pausa (*pantalla_pausa*). Para poder asignar la pantalla de pausa a la pantalla del juego correcto, debemos tener funciones *PausarJuego()* diferentes para cada juego.
- ***ContinuarJuego(fsm_t* this):*** Realizamos una cuenta atrás desde 3 llamando a la función del display auxiliar *display_countdown()* pasando como parámetros 3 (número en el que comienza la cuenta atrás) y 1000 (delay entre números). Pasados los 3 segundo, reseteamos la flag de pausa a 0 y llamamos a la función *ActualizaJuego()* para continuar el juego de forma inmediata (también se podría activar el timer y que, pasado el tiempo de actualización, la máquina de estados actualizará la partida automáticamente).

5 Mejora 3: Sistema de Puntuación

5.1 Objetivos de la mejora

Primero, la creación de un sistema de puntuación permitiría al usuario conocer cómo está jugando, sabiendo de una forma sencilla cuántos ladrillos ha destruido o cuántos puntos le ha ganado a su contrincante.

5.2 Descripción del subsistema Software

La implementación de la puntuación se realiza usando una o varias variables de números enteros, como atributo de cada juego, donde se almacena la puntuación del jugador o jugadores. Al usar variables como atributos de cada juego, permitimos el acceso a la puntuación del juego de una forma sencilla desde cualquier parte del código. De esta forma, cambiar el valor de la puntuación del jugador, se transforma en algo trivial, simplemente accediendo al atributo *score* ,o *scoreX* si se cuenta con más de un jugador.

5.2.1 Flujo de ejecución del programa principal

Se separará en dos la explicación, por una parte cómo se ha implementado dentro del juego ArkanoPi y por otra cómo se ha implementado dentro de Pong.

5.2.1.1 Puntuación en ArkanoPi

La puntuación es un atributo de tipo entero dentro de *tipo_arkanoPi*. Se inicializa a 0 dentro de la función InicializaJuego. Dentro de ActualizaJuego, si la pelota ha golpeado contra un ladrillo, la puntuación se aumenta en 1 unidad. En caso de volver a jugar una vez se pierda o se gane, se pone a 0 de nuevo la puntuación dentro de la función StartJuego.

5.2.1.2 Puntuación en Pong

Dado que hay dos jugadores, la puntuación de cada uno es un atributo de tipo entero dentro de *tipo_pong*: score1 para el jugador 1 y *score2* para el jugador 2. Adicionalmente, tenemos un atributo dentro de *tipo_pong* de tipo entero llamado *score_round*, para ver a cuántos puntos estamos jugando. Esto es necesario ya que gana el jugador que llegue primero a 3 puntos, por lo que una vez esto ocurre es necesario reiniciar las puntuaciones de ambos jugadores. Esto se podría modificar y ajustar el valor de *score_round* según el número de rondas que queramos que tenga.

En la función InicializaJuegoPong, se inicializa *score1* y *score2* a 0, mientras que *score_round* se inicializa a 3. En la función *compruebaPunto()*, se mira qué jugador ha conseguido un punto, y se suma 1 unidad a su puntuación. Ésto lo sabemos mirando si la pelota ha pasado los límites de la pantalla, si ha sido por encima, el jugador 1 ha conseguido un punto y si es por abajo quiere decir que el jugador 2 ha conseguido un punto.

En la función ActualizaJuego, llamamos a compruebaPunto. Una vez se actualice el valor de la puntuación de cada jugador correspondientemente, vemos si alguno de los dos ha alcanzado el valor de score_round, llamamos a la función VictoriaPong pasándole como parámentro el jugador que ha ganado. Se mostraría en el display auxiliar que jugador ha ganado y se resetean score1 y score2 a 0 por si se desea jugar otra partida.

6 Mejora 4: Controlador de Juegos

6.1 Objetivos de la mejora

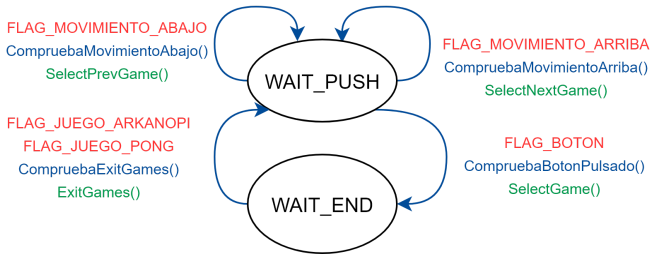
Diseñamos un controlador encargado de la selección del juego, así como de la actualización de displays correspondiente a la pantalla inicial. Este controlador facilita la implementación de nuevos juegos y extrae la pantalla inicial común a los juegos en sí.

6.2 Descripción del subsistema Software

La arquitectura software se basa en una máquina de estados que, en primera instancia, se encarga de cambiar el juego seleccionado y comunicar a éste que ha sido seleccionado. La elección de esta implementación permite desligar completamente el tratamiento de la pantalla de inicio de

los propios juegos. Además añadir nuevos juegos se convierte en algo trivial, tal y como veremos más adelante.

6.2.1 Flujo de ejecución del programa principal



En primera instancia se entra en un estado de *WAIT_PUSH* en el que se comprueba si se ha pulsado la tecla de elección de uno de los juegos para seleccionar ese juego y cargar su icono en el display auxiliar. Al mismo tiempo, se comprueba la pulsación del botón de selección que confirma la elección anterior y activa la flag correspondiente al juego seleccionado. Una vez se ha seleccionado, se pasa al estado *WAIT_END* donde se espera que se desactive la flag del juego correspondiente y volver al estado inicial.

El uso de flags permite una comunicación sencilla y rápida entre procesos, permitiendo que podamos comunicar procesos diferentes realizando tan solo una operación lógica sencilla (activaciones de flags).

6.2.1.1 *Función de entrada o comprobación*

- **Funciones de comprobación de botones (arriba, abajo y start):** De la misma manera que en la versión original del proyecto, se realiza la comprobación de la flag correspondiente mediante una operación de AND lógica y devolviendo el resultado de la misma.
- ***CompruebaExitGames(fsm_t* this):*** Esta función pretende comprobar que ambas flags de juego (*FLAG_JUEGO_ARKANOPI* y *FLAG_JUEGO_PONG*) están desactivadas, lo que significa que se ha salido del juego al que se estaba jugando y se debe volver a la pantalla de inicio. Para esto debemos primero intercambiar los valores de ambas flags de juego mediante una operación XOR lógica para luego comprobar mediante una operación AND lógica.

6.2.1.2 *Función de salida o actuación*

- ***SelectNextGame(fsm_t* this)***

Primero reseteamos la flag de la tecla de arriba (*FLAG_MOV_ARRIBA*). Aumentamos en uno el juego elegido, devolviendolo a 0 en caso de haber llegado al valor del número de juegos (lo que significa que hemos llegado al último juego), y actualizamos el icono del display auxiliar llamando a la función *display_icon()* pasándole el valor del juego elegido.

- ***SelectPrevGame(fsm_t* this)***

Primero reseteamos la flag de la tecla de abajo (*FLAG_MOV_ABAJO*). Disminuimos en uno el juego elegido, devolviéndolo al valor del último juego en caso de ser menos que cero (lo que significa que hemos llegado al primer juego), y actualizamos el icono del display auxiliar.

- ***SelectGame(fsm_t* this)***

Primero reseteamos la flag del botón de start (*FLAG_BOTON*). Mediante un switch-case activamos la flag del juego seleccionado.

- ***ExitGames(fsm_t* this)***

En esta función simplemente devolveremos el display auxiliar al icono inicial.

6.2.1.3 *Modificaciones de los juegos*

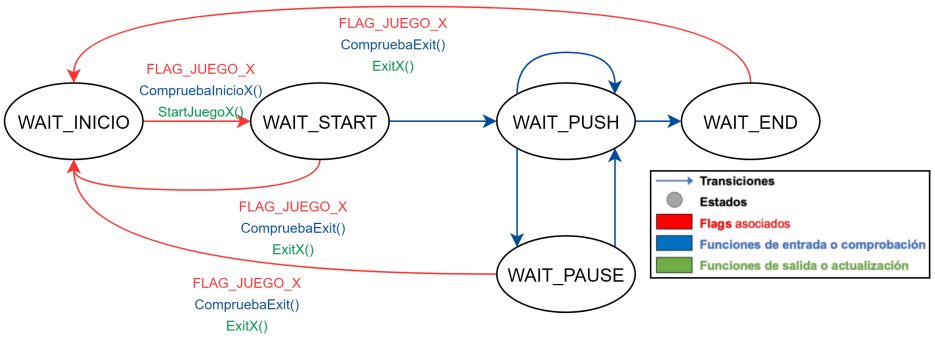
- **Estado de inicio en la máquina de estados**

Añadimos un estado inicial *WAIT_INICIO* que estará comprobando el estado de la flag del juego correspondiente (*FLAG_JUEGO_ARKANOPI* o *FLAG_JUEGO_PONG*) y cuando esté

activada pasará al estado *WAIT_START* donde esperará que se pulse el botón de start, iniciando el proceso del juego.

Además, modificamos la máquina de estados para que una vez se resetea la partida se llame a la función *StartJuego* en vez de *InicializaJuego*, para poder distinguir entre la inicialización del juego y el inicio de la partida.

- **Salida del juego**
  En los estados de WAIT_START, *WAIT_PAUSE* y *WAIT_END* se comprueba la pulsación del botón de Exit y si se pulsa se llama a la función *ExitArkano()* o *ExitPong()* y se vuelve al estado *WAIT_INICIO*. Estas funciones de salida resetearán las flags del botón de exit (*FLAG_EXIT*) y del juego (*FLAG_JUEGO_ARKANOPI* o *FLAG_JUEGO_PONG*), ésta última comunicará al controlador que se ha acabado el juego, tal y como se especificó en la función *CompruebaExitGames()*.



El diagrama completo del juego queda de la siguiente manera (solo se especifican los cambios realizados que pueden verse en las líneas de color rojo).

7 Mejora 5: Juego Ping Pong

7.1 Objetivos de la mejora

Con el fin de ampliar las funcionalidades del sistema, se diseña un juego de Ping Pong.

7.2 Descripción del subsistema Software

La implementación del juego se realiza de la misma forma que el Arkano, mediante una máquina de estados. Al tener máquinas de estados independientes para cada juego, permitimos que los juegos sean completamente independientes y con funcionalidades completamente diferentes, simplemente se requiere un punto de entrada (comprobando la activación de la flag del juego *FLAG_JUEGO_PONG*) y un punto de salida que desactive esta entrada.

Puesto que el juego Pong tiene grandes similitudes con el juego Arkano, se crea una librería de funciones comunes para no tener que doblar funciones idénticas (*commonLib.c*)

7.2.1 Flujo de ejecución del programa principal

Tal y como en Diagrama de subsistemas ampliado el objeto pong se define con:

- Una pantalla *tipo_pantalla**: *p_pantalla*
- 2 palas *tipo_pala*: *pala* y *pala2*
- Una pelota *tipo_pelota*: *pelota*
- Un timer para la actualización del juego *trm_t**: *tmr_actualizacion_juego_isr*
- 2 variables *int* de puntuación: *score1* y *score2*

- Una variable *int* con la puntuación necesaria para ganar la ronda: *score_round*

A continuación, se detallan las funciones específicas del juego Pong:

7.2.1.1 Funciones de Inicialización y Reset

- ***InicializaPala2(tipo_pala *p_pala):*** Esta función recibe un objeto pala para iniciarla con la función común *InicializaPala()* y luego cambiar la posición de ésta a la primera línea (coordenada 'y' a 0).
- ***InicializaPong(tipo_pong *p_pong):*** Llama a la función de *ResetPong()* y actualiza pantalla del juego con *ActualizaPantallaPong()*.
- ***ResetPong(tipo_pong *p_pong):*** Llama a las funciones comunes *ReseteaPantalla(), InicializaPelota()* y *InicializaPala()* y a la función antes explicada *InicializaPala2()*.

7.2.1.2 Procedimientos para la Gestión del Juego

- ***CompruebaPunto(tipo_pong *p_pong):*** Comprueba si la posición de la pelota saldría de la pantalla (por arriba o por abajo) en la siguiente iteración, es decir, si la posición de la pelota más la trayectoria es mayor (o igual) que el número de filas del display (igual que en el juego ArkanoPi) o menor que cero (la pelota sale por arriba de la pantalla).

7.2.1.3 Procedimientos para la Visualización del Juego

- ***ActualizaPantallaPong(tipo_pong *p_pong):*** Llama a las funciones comunes *ReseteaPantalla(), PintaPala()* pasando primero la *pala1* y luego la *pala2*, *PintaPelota()*.
- ***ActualizaPantallaScorePong(tipo_pong *p_pong):*** Actualiza la pantalla de puntos lllamando a la función del display auxiliar *display_score()*, pasando como parámetro un array formado por la puntuación de ambos jugadores.

7.2.1.4 Funciones de Comprobación de la Máquina de Estados

- ***CompruebaInicioPong(fsm_t* this):*** La función comprueba el estado de la flag de juego *FLAG_JUEGO_PONG* tal y como se explicó en Mejora 4: Controlador de Juegos.
- ***CompruebaMovimientoIzquierda2(fsm_t* this) y CompruebaMovimientoDerecha2(fsm_t* this):*** Comprueba el estado de las flags de movimiento del Jugador 2 (*FLAG_MOVIMIENTO_IZQUIERDA2* y *FLAG_MOVIMIENTO_DERECHA2*) de la misma manera que se hace en el proyecto original.

7.2.1.5 Funciones de Acción de la Máquina de Estados

Estas funciones son muy similares a las desarrolladas en el ArkanoPi. Un *InicializaJuegoPong()* encargado de inicializar el sistema, un *StartJuegoPong()* para comenzar la ronda (tal y como se explica en Mejora 4: Controlador de Juegos), un *ActualizarJuegoPong()*, un *FinalJuegoPong()* para los finales de ronda y un *ReseteaJuegoPong()* encargada de resetear el juego. Adicionalmente se desarrolla una función *VictoriaPong()* para tratar los finales de partida tal y como se explica en Mejora : Sistema de Puntuación.

7.2.2 Procesos de las interrupciones

El juego Pong usa, al igual que el ArkanoPi, una interrupción causada por un timer para actualizar el juego de forma automática. Con el fin de ahorrar código, se ha utilizado la misma función de atención a la interrupción *tmr_actualizacion_juego_isr()*.

8 Mejora 6: Display Auxiliar

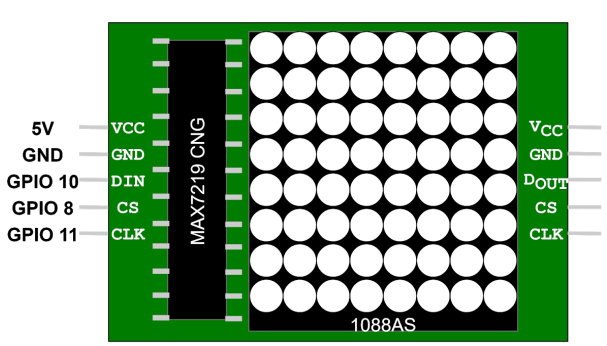
8.1 Objetivos de la mejora

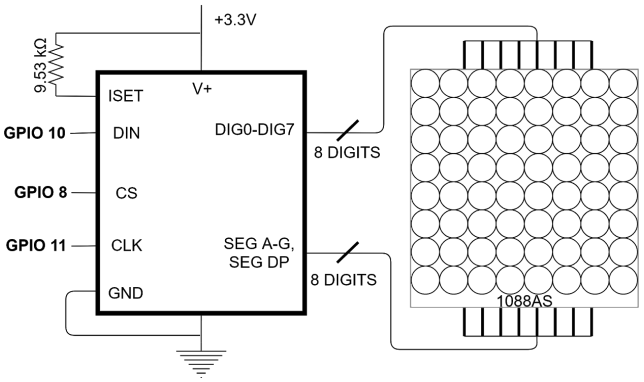
El objetivo de añadir este segundo display es conseguir otro apoyo al display donde se imprime el juego para incluir las funcionalidades anteriormente descritas como es poder ver el score mientras se juega, o poder saber qué juego se ha seleccionado en cada instante.

8.2 Descripción del subsistema Hardware.

8.2.1 Descripción del módulo Display Auxiliar

Para incluir un display auxiliar, hemos empleado el componente 64er LED Matrix Display de AZ-Delivery. La razón de ser las conexiones del componente a la Raspberry Pi que se pueden ver en la imagen inferior es porque para la transmisión de la información de la Raspberry Pi a la matriz de LEDs se emplea el protocolo SPI. SPI es un protocolo serie que nos permite comunicarnos con los componentes conectados a la Raspberry Pi empleando únicamente 3 pines, uno de alimentación y uno de tierra. Si nos fijamos en cómo está hecho el componente, podemos encontrar dos módulos principales. En primer lugar encontramos una unidad LED de visualización para displays, MAX7219CNG de Maxim Integrated, que es el que nos permite emplear el protocolo SPI, ya que como se puede ver en el esquema inferior el que traduce los datos de entrada que recibe desde la Raspberry Pi a 16 pines de salida que son los que se conectan a la matriz de LED. En segundo lugar tenemos la matriz LED 8x8, 1088AS, que es donde se refleja la imagen que queramos obtener.





Dado que el componente utilizado ya contaba con las conexiones y elementos necesarios para conectarlo al microcontrolador, no ha sido necesario añadir ninguno adicional. Además dado que las conexiones eran muy sencillas, no hemos encontrado ningún problema en implementarlo.

8.3 Descripción del subsistema Software

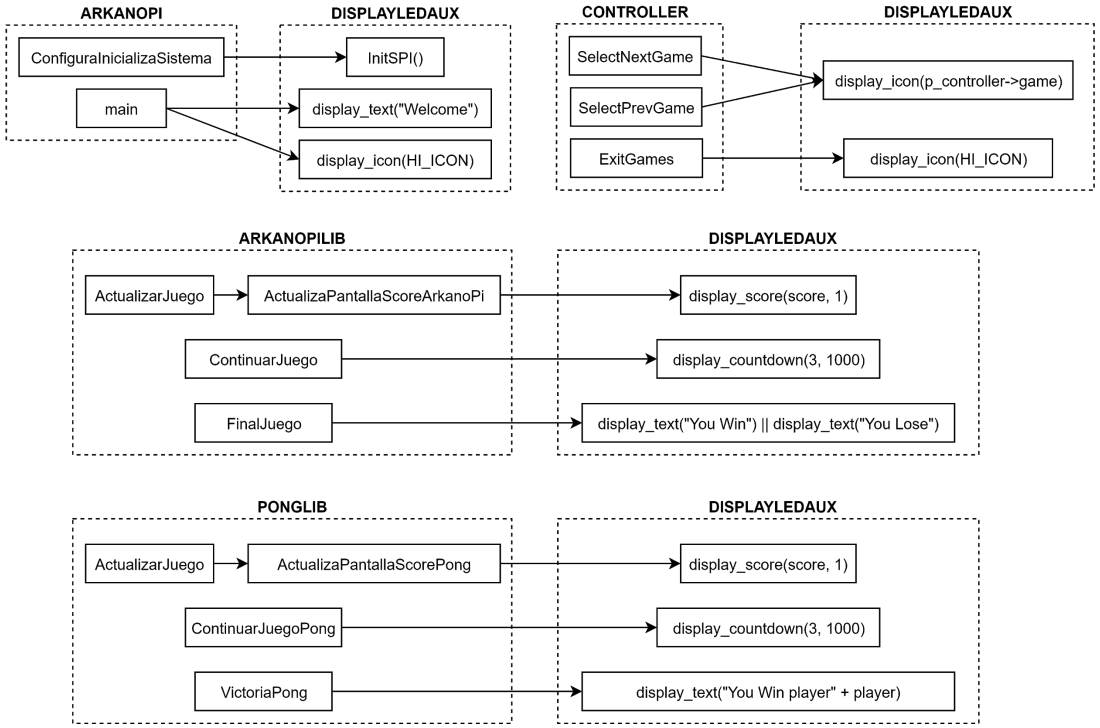
8.3.1 Flujo de ejecución del programa principal

El código empleado para el uso del display auxiliar se puede dividir en dos partes: las funciones internas necesarias para la correcta impresión sobre el display, y las funciones empleadas para imprimir sobre el display lo que queramos. Para su implementación, se ha hecho uso de los headers encontrados en WiringPiSPI.h de la librería WiringPi.

En el diagrama inferior podemos ver las interacciones externas con el display auxiliar. Como se puede ver, se van llamando según se actualizan los valores que se muestran sobre el display

auxiliar. Hemos optado hacerlo de esta manera ya que así únicamente cambiamos la imagen bajo necesidad y no es necesario ir comprobando periódicamente como sucede con un timer.

La estructura seguida por el diagrama pretende facilitar la comprensión de cada función dentro de *DisplayLedAux()* y su utilidad, por lo que recomendamos ir mirándolo mientras se vaya leyendo la explicación de cada función. A continuación se explicarán dichas funciones.



8.3.1.1 *Write_SPI(unsigned char reg, unsigned char data)*

Esta función es la encargada de llamar a la función *wiringPiSPIDataRW()* perteneciente al header WiringPiSPI.h para comunicar los datos necesarios al display. El parámetro reg identifica la fila del display donde va dirigida la información, yendo desde 1 para la primera fila hasta 8 para la última, y el parámetro data la información que se refleja en ella, pasando un hexadecimal de 8 bits correspondiendo cada bit del mismo con cada uno de los leds de la fila.

La función *wiringPiSPIDataRW()* exige como parámetros un canal, que en este caso empleamos el canal 0, los datos a transmitir, que en este caso es el array formado por reg y data, y la longitud del array los datos pasados que en ese caso es 2.

8.3.1.2 *InitSPI()*

Es la encargada de inicializar el display. Esto se realiza llamando a la función *Write_SPI()*, pasando como primer parámetro la dirección de la variable a inicializar y como segundo parámetro cómo queremos que se inicialice. Como podemos ver en la datasheet correspondiente al componente MAX7219CNG, la dirección *0x09* corresponde con modo del decodificador (configurado a *0x00* - no decode mode), la dirección *0x0A* con la intensidad del display (configurado a *0xFF* - maximum value), la dirección *0x0B* al número de dígitos escaneados (configurado a *0x07* - digits 01234567), la dirección *0x0C* con el apagado del display (configurado a *0x01* - normal shutdown) y la dirección *0x0F* con el display test (configurado a *0x00* - normal operation).

8.3.1.3 *Push(char row) y ActualizaDisplayAux()*

La función *Push()* se emplea para ir metiendo en cada una de las posiciones del array display la información que se desea pasar a cada fila del array, siendo dicha información el

parámetro *row* que se le pasa. Una vez tengamos en el array display los datos que queramos, llamamos a *ActualizaDisplayAux()* que llama a la función *Write_SPI()* para trasladar a cada fila la información necesaria. Estas dos funciones se encuentran separadas para permitir funcionalidades que se verán en los siguientes apartados.

8.3.1.4 *Display_clear()*

Es la encargada de apagar todos los LEDs del display. Para ello, llama a la función push pasando como parámetro 0 tantas veces como filas hay en el display. Una vez el array display está completo, se llama a *ActualizaDisplayAux()* para que se reflejen los cambios en el display auxiliar.

8.3.1.5 *Display_score(int score[], int nPlayers)*

Es la encargada de ir mostrando la puntuación del juego en el display auxiliar. Es necesario pasarle como parámetros un array con las puntuaciones que haya actualmente en el juego y el número de jugadores, por ejemplo, para el juego Pong el array de puntuaciones contendrá dos puntuaciones y el número de jugadores será 2. Los valores necesarios para representar cada fila de cada número que se quiera imprimir sobre el display se pueden encontrar en el array *nums*, definido por nosotros que se ubica en el archivo *font-8x8.inc*. En este cada número se encuentra definido de tal forma que ocupe 4 columnas del display para permitir realizar las combinaciones necesarias para representar desde el valor 00 hasta el valor 99. Para realizar dichas combinaciones, únicamente debemos multiplicar el número que se encuentre en las decenas por 16 (hacemos un shift para la izquierda de 4 posiciones) y sumarle el número que se encuentre en las unidades.

Miramos inicialmente el número de jugadores. En caso de que sea 1, dividimos la puntuación en unidades y decenas empleando una función auxiliar llamada *split()*. Si hay más esto no es necesario puesto que las puntuaciones sólo alcanzan el valor de las unidades. A continuación, formamos el número que queramos, realizando la multiplicación correspondiente y llamando al array nums con el número que queramos imprimir en cada lugar. Una vez tengamos cada fila obtenida, llamamos a la función push pasándole el valor de los números que queramos imprimir. Una vez la función push complete todas las posiciones del array display, llamamos a la función *ActualizaDisplayAux()* para obtener el número final el el display auxiliar.

8.3.1.6 *Display_countdown(int start, int delay)*

Es la función encargada de representar la cuenta atrás cuando queremos pasar del estado de pausa a continuar. Como parámetro se le pasa el número desde el que queremos comenzar la cuenta atrás (típicamente 3) y el delay que debe haber entre la impresión de cada número. Seguimos un procedimiento similar al empleado en la función *display_score()*. Miramos en el array font que se encuentra en el archivo *font-8x8.inc* a partir de la posición 48, ya que es donde comienza la representación de los números, y vamos pasando cada fila como parámetro llamando a la función push. Una vez esté el array display con los valores necesarios, llamamos a *ActualizaDisplayAux()* para que se imprima en la pantalla.

El array font ha sido tomado del código abierto que se puede encontrar en el siguiente repositorio de GitHub.

8.3.1.7 *Display_icon(int icon)*

Su función es imprimir sobre el display auxiliar los distintos iconos que tenemos que se encuentran definidos en el array icons (definido por nosotros) que se encuentra en el archivo *font-8x8.inc*: pantalla inicial (Hi!), selección juego ArkanoPi y selección juego Pong. Funciona de la misma forma que las funciones anteriores, miramos en el array *icons* la posición que se nos pasa como parámetro y haciendo uso de la función push rellenamos el array display con los valores correspondientes. Finalmente llamamos a ActualizaDisplayAux para mostrar el icono en el display.

8.3.1.8 *Display_text(char* text)*

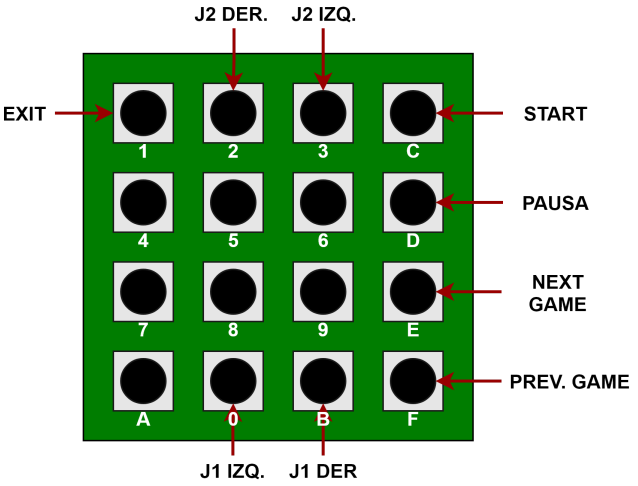
Su función es representar un texto sobre el display auxiliar, mostrandolo letra por letra en forma de scroll. Así como en la función *display_countdown()*, hacemos uso del array fonts, comenzando a mirar desde la posición 0. Para obtener el texto en formato scroll, hacemos igual que en las funciones anteriores, una vez tengamos los datos necesarios para una fila, llamamos a la función push, pero en este caso en vez de esperar a tener el array display completo, vamos llamando a *ActualizaDisplayAux()* fila por fila. Adicionalmente, cada vez que se termina una letra dejamos una fila en blanco para que no se solapen.

9 Principales problemas encontrados

En cuanto al hardware, hemos encontrado problemas con respecto a las conexiones necesarias para el display principal, ya que al emplear tantos componentes intermedios entre el microcontrolador y el display, por lo que tuvimos que ir comprobando paso a paso que cada una de las conexiones era correcta.

En cuanto al software, en primer lugar encontramos problemas con el debounce time del teclado dado que se producían pulsaciones innecesarias, por lo que aumentando el mismo lo conseguimos solucionar. Por otra parte, a la hora de implementar el código correspondiente al teclado auxiliar, tuvimos problemas inicialmente en entender adecuadamente el funcionamiento del protocolo SPI. Además, una vez optamos por emplear el header *WiringPiSPI.h*, nos dimos cuenta de que el proyecto se había dejado de desarrollar, por lo que encontramos problemas para encontrar información.

10 Manual de usuario



En la imagen lateral, se puede ver el teclado que se encuentra conectado a la Raspberry Pi. Para comenzar a jugar, primero se debe escoger juego. Si se pulsa la tecla NEXT GAME, se seleccionará el juego siguiente al que salga actualmente y si se pulsa la tecla PREV. GAME seleccionará el juego anterior. Para confirmar la selección, se debe pulsar la tecla START. Podrá ver en el display auxiliar un icono del juego seleccionado antes de confirmar su elección.

En caso de haber seleccionado el juego Arkano Pi, se verá en el display y se podrá proceder a pulsar la tecla de start para comenzar la ronda. Para mover la pala, se moverá hacia izquierda si se pulsa la tecla J1 IZQ. y a la derecha en caso de pulsar la tecla J1 DER. En caso de querer volver a empezar ya que se haya perdido o se haya ganado, basta con volver a pulsar la tecla START.

Si se ha seleccionado el jugo Ping Pong, el jugador 1 se moverá a la izquierda pulsando la tecla J1 IZQ. y a la derecha pulsando la tecla J1 DER. El jugador 2 se moverá a la izquierda pulsando la tecla J2 IZQ. y la derecha pulsando la tecla J2 DER. Para iniciar cada ronda dentro de una misma partida tanto como para empezar una partida nueva, habrá que pulsar el botón START y

los puntos se reflejarán acordemente, reiniciándose tras acabar la partida cuando unos de los 2 jugadores llega a 3 puntos.

En cualquiera de los dos juegos, se puede pausar pulsando la tecla PAUSA. Para continuar con el juego, se tendrá que pulsar nuevamente la tecla PAUSA y tras una cuenta atrás en la pantalla auxiliar de 3 segundos continuará el juego por el estado en el que se pausó.

En caso de estar jugando a uno de los dos juegos y querer cambiar al otro, basta con pulsar la tecla EXIT y se volverá a la pantalla inicial para seleccionar juego como se ha explicado anteriormente.

11 Bibliografía

Datasheet componente MAX7219CNG:
https://www.mouser.es/datasheet/2/256/max7219-max7221-1178406.pdf

Datasheet matrix de LED 1088AS:
https://2653vv2fz8mg3wr99k44dygo-wpengine.netdna-ssl.com/wp-content/uploads/2017/08/DATASHEET-1088AS.pdf

Blog consultado para la compresión del protocolo de comunicación PSI y sus posibles librerías:
https://franciscomoya.gitbooks.io/taller-de-raspberry-pi/content/es/c/spi.html

Código abierto empleado para la obtención del array font empleado en LedDisplayAux:
https://github.com/raspberry-alpha-omega/spi/

Código abierto empleado para consultar el funcionamiento de un display de LEDs con protocolo SPI con la librería BCM:
https://github.com/leon-anavi/raspberrypi-matrix-led-max7219

Comunicado oficial de la discontinuidad del desarrollo de la librería WiringPi:
http://wiringpi.com/wiringpi-deprecated/

12 ANEXO I: Código del programa del proyecto final

1) ARKANOPI.C

```
#include "arkanoPi.h"
#include <stdio.h>
#include <string.h>

// Inicializamos las flags
int flags = 0;

// Declaramos el sistema
TipoSistema sistema;

// Declaracion del objeto teclado
TipoTeclado teclado = {
	.columnas = {
		GPIO_KEYBOARD_COL_1,
		GPIO_KEYBOARD_COL_2,
		GPIO_KEYBOARD_COL_3,
		GPIO_KEYBOARD_COL_4

	},
	.filas = {
		GPIO_KEYBOARD_ROW_1,
		GPIO_KEYBOARD_ROW_2,
		GPIO_KEYBOARD_ROW_3,
		GPIO_KEYBOARD_ROW_4
	},
	/* Rutinas de atención a las
	interrupciones de las filas */
	.rutinas_ISR = {
		teclado_fila_1_isr,
		teclado_fila_2_isr,
		teclado_fila_3_isr,
		teclado_fila_4_isr
	},

	.columna_actual = COLUMNA_1,

	.teclaPulsada.col = -1,
```

```
	.teclaPulsada.row = -1,
};

// Declaracion del objeto display
TipoLedDisplay led_display = {
	.pines_control_columnas = {
		GPIO_LED_DISPLAY_COL_1,
		GPIO_LED_DISPLAY_COL_2,
		GPIO_LED_DISPLAY_COL_3
		/* GPIO_LED_DISPLAY_COL_4 */

	},
	.filas = {
		GPIO_LED_DISPLAY_ROW_1,
		GPIO_LED_DISPLAY_ROW_2,
		GPIO_LED_DISPLAY_ROW_3,
		GPIO_LED_DISPLAY_ROW_4,
		GPIO_LED_DISPLAY_ROW_5,
		GPIO_LED_DISPLAY_ROW_6,
		GPIO_LED_DISPLAY_ROW_7

	},

	.p_columna = 0,
};

// Declaracion del objeto controller
TipoController controller = {
	.game = 0,
};

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

// int ConfiguracionSistema (TipoSistema *p_sistema): procedimiento de
configuracion
// e inicializacion 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,
// la inicializacion de los diferentes elementos de los que consta
nuestro sistema,
// crear, si fuese necesario, los threads adicionales que pueda
requerir el sistema
// como el thread de exploración del teclado del PC
int ConfiguraInicializaSistema (TipoSistema *p_sistema) {
	int result = 0;

	// Inicializamos el wiringPi
	wiringPiSetupGpio();

	// Inicializamos los componentes
	InicializaTeclado(&teclado);
	InicializaLedDisplay(&led_display);
	InitSPI();

	// Lanzamos thread para exploracion del teclado convencional del
PC
	/* result = piThreadCreate (thread_explora_teclado_PC); */

	if (result != 0) {
		printf ("Thread didn't start!!!\n");
		return -1;
	}

	return result;
}

//------------------------------------------------------
// FUNCIONES LIGADAS A THREADS ADICIONALES
//------------------------------------------------------

PI_THREAD (thread_explora_teclado_PC) {
	int teclaPulsada;

	while(1) {
		/* Wiring Pi function: pauses program */
		/* execution for at least 10 ms */
		delay(10);

		piLock (STD_IO_BUFFER_KEY);
```

```
        if(kbhit()) {
            teclaPulsada = kbread();

            switch(teclaPulsada) {
                case 'a':
                    piLock(SYSTEM_FLAGS_KEY);
                    flags |= FLAG_MOV_IZQUIERDA;
                    piUnlock(SYSTEM_FLAGS_KEY);
                    break;

                case 'd':
                    piLock(SYSTEM_FLAGS_KEY);
                    flags |= FLAG_MOV_DERECHA;
                    piUnlock(SYSTEM_FLAGS_KEY);
                    break;

                case 'c':
                    piLock(SYSTEM_FLAGS_KEY);
                    flags |= FLAG_TIMER_JUEGO;
                    piUnlock(SYSTEM_FLAGS_KEY);
                    break;

                case 's':
                    piLock(SYSTEM_FLAGS_KEY);
                    flags |= FLAG_BOTON;
                    piUnlock(SYSTEM_FLAGS_KEY);
                    break;

                case 'p':
                    piLock(SYSTEM_FLAGS_KEY);
                    flags |= FLAG_PAUSA_JUEGO;
                    piUnlock(SYSTEM_FLAGS_KEY);
                    break;

                case '1':
                    piLock(SYSTEM_FLAGS_KEY);
                    flags |= FLAG_MOV_ABAJO;
                    piUnlock(SYSTEM_FLAGS_KEY);
                    break;

                case '0':
```

```
                              piLock(SYSTEM_FLAGS_KEY);
                              flags |= FLAG_MOV_ARRIBA;
                              piUnlock(SYSTEM_FLAGS_KEY);
                              break;

                        case 'j':
                              piLock(SYSTEM_FLAGS_KEY);
                              flags |= FLAG_MOV_IZQUIERDA2;
                              piUnlock(SYSTEM_FLAGS_KEY);
                              break;

                        case 'l':
                              piLock(SYSTEM_FLAGS_KEY);
                              flags |= FLAG_MOV_DERECHA2;
                              piUnlock(SYSTEM_FLAGS_KEY);
                              break;

                        case 'x':
                              piLock(SYSTEM_FLAGS_KEY);
                              flags |= FLAG_EXIT;
                              /* flags |= FLAG_EXIT; */
                              piUnlock(SYSTEM_FLAGS_KEY);
                              break;

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

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

            piUnlock (STD_IO_BUFFER_KEY);
      }
}

// wait until next_activation (absolute time)
void delay_until (unsigned int next) {
      unsigned int now = millis();
```

```
	if (next > now) {
		delay (next - now);
	}
}

int main () {
	unsigned int next;

	// Maquina de estados: lista de transiciones
	// {EstadoOrigen, CondicionDeDisparo, EstadoFinal,
AccionesSiTransicion }

	fsm_trans_t arkanoPi[] = {
		{ WAIT_INICIO, CompruebaInicioArkanoPi, WAIT_START,
InicializaJuego },
		{ WAIT_START, CompruebaBotonPulsado, WAIT_PUSH, StartJuego
},

		{ WAIT_PUSH, CompruebaTimeoutActualizacionJuego, WAIT_PUSH,
ActualizarJuego },
		{ WAIT_PUSH, CompruebaMovimientoIzquierda, WAIT_PUSH,
MuevePalaIzquierda },
		{ WAIT_PUSH, CompruebaMovimientoDerecha, WAIT_PUSH,
MuevePalaDerecha },

		{ WAIT_PUSH, CompruebaPausaJuego, WAIT_PAUSE, PausarJuego },
		{ WAIT_PAUSE, CompruebaPausaJuego, WAIT_PUSH, ContinuarJuego
},

		{ WAIT_PUSH, CompruebaFinalJuego, WAIT_END, FinalJuego },
		{ WAIT_END,  CompruebaBotonPulsado, WAIT_START, ReseteaJuego
},

		{ WAIT_START,  CompruebaExit, WAIT_INICIO, ExitArkano },
		{ WAIT_PAUSE,  CompruebaExit, WAIT_INICIO, ExitArkano },
		{ WAIT_END,  CompruebaExit, WAIT_INICIO, ExitArkano },
		{-1, NULL, -1, NULL },
	};
	fsm_trans_t pong[] = {
		{ WAIT_INICIO, CompruebaInicioPong, WAIT_START,
InicializaJuegoPong },
		{ WAIT_START, CompruebaBotonPulsado, WAIT_PUSH,
```

```
StartJuegoPong },

            { WAIT_PUSH, CompruebaTimeoutActualizacionJuego, WAIT_PUSH,
ActualizarJuegoPong },
            { WAIT_PUSH, CompruebaMovimientoIzquierda, WAIT_PUSH,
MuevePalaIzquierdaPong },
            { WAIT_PUSH, CompruebaMovimientoDerecha, WAIT_PUSH,
MuevePalaDerechaPong },
            { WAIT_PUSH, CompruebaMovimientoIzquierda2, WAIT_PUSH,
MuevePalaIzquierdaPong2 },
            { WAIT_PUSH, CompruebaMovimientoDerecha2, WAIT_PUSH,
MuevePalaDerechaPong2 },

            { WAIT_PUSH, CompruebaPausaJuego, WAIT_PAUSE,
PausarJuegoPong },
            { WAIT_PAUSE, CompruebaPausaJuego, WAIT_PUSH,
ContinuarJuegoPong },

            { WAIT_PUSH, CompruebaFinalJuego, WAIT_END, FinalJuegoPong
},
            { WAIT_END,  CompruebaBotonPulsado, WAIT_START,
ReseteaJuegoPong },

            { WAIT_START,  CompruebaExit, WAIT_INICIO, ExitPong },
            { WAIT_PAUSE,  CompruebaExit, WAIT_INICIO, ExitPong },
            { WAIT_END,  CompruebaExit, WAIT_INICIO, ExitPong },
            {-1, NULL, -1, NULL },
      };

      // Configuracion e incializacion del sistema
      ConfiguraInicializaSistema (&sistema);
      sistema.arkanoPi.p_pantalla = &(led_display.pantalla);
      sistema.pong.p_pantalla = &(led_display.pantalla);

      // Juegos
      fsm_t* arkanoPi_fsm = fsm_new (WAIT_INICIO, arkanoPi, &sistema);
      fsm_t* pong_fsm = fsm_new (WAIT_INICIO, pong, &sistema);

      // Controller
      fsm_t* selector_fsm = fsm_new (WAIT_PUSH, fsm_trans_selector,
&(controller));
```

```
	// Teclado
	fsm_t* teclado_fsm = fsm_new ( TECLADO_ESPERA_COLUMNA,
fsm_trans_excitacion_columnas, &(teclado));
	fsm_t* tecla_fsm = fsm_new (TECLADO_ESPERA_TECLA,
fsm_trans_deteccion_pulsaciones, &(teclado));
	teclado.tmr_duracion_columna = tmr_new
(timer_duracion_columna_isr);

	// Display
	fsm_t* display_fsm = fsm_new (DISPLAY_ESPERA_COLUMNA,
fsm_trans_excitacion_display, &(led_display));
	led_display.tmr_refresco_display =  tmr_new
(timer_refresco_display_isr);

	// Start screen
	display_text("Welcome");
	display_icon(HI_ICON);

	next = millis();
	while (1) {
		// Arrancamos fsm del sw
		fsm_fire (selector_fsm);
		fsm_fire (arkanoPi_fsm);
		fsm_fire (pong_fsm);

		// Arrancamos fsm del hw
		fsm_fire (teclado_fsm);
		fsm_fire (tecla_fsm);
		fsm_fire (display_fsm);

		next += CLK_MS;
		delay_until (next);

	}

	// Eliminamos las maquinas de estado
	tmr_destroy((tmr_t*)(tmr_actualizacion_juego_isr));
	tmr_destroy((tmr_t*)(timer_duracion_columna_isr));

	fsm_destroy (selector_fsm);
	fsm_destroy (arkanoPi_fsm);
	fsm_destroy (pong_fsm);
```

```
	fsm_destroy (teclado_fsm);
	fsm_destroy (tecla_fsm);
	fsm_fire (display_fsm);
}
```

2) ArkanoPiLib.c

```
#include "arkanoPiLib.h"

int ladrillos_basico[NUM_FILAS_DISPLAY][NUM_COLUMNAS_DISPLAY] = {
	{1,1,1,1,1,1,1,1}, // 0xFF
	{1,1,1,1,1,1,1,1}, // 0xFF
	{0,0,0,0,0,0,0,0}, // 0x00
	{0,0,0,0,0,0,0,0}, // 0x00
	{0,0,0,0,0,0,0,0}, // 0x00
	{0,0,0,0,0,0,0,0}, // 0x00
	{0,0,0,0,0,0,0,0}, // 0x00

};

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

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

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

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

	for(i=0;i<NUM_FILAS_DISPLAY;i++) {
		for(j=0;j<NUM_COLUMNAS_DISPLAY;j++) {
			p_pantalla->matriz[i][j] = p_ladrillos->matriz[i][j];
```

```
            }
        }
}

void ActualizaPantalla(tipo_arkanoPi* p_arkanoPi) {

    // Borro toda la pantalla
      ReseteaPantalla((tipo_pantalla*)(p_arkanoPi->p_pantalla));

    // Pinta los ladrillos
      PintaLadrillos(
            (tipo_pantalla*)(&(p_arkanoPi->ladrillos)),
            (tipo_pantalla*)(p_arkanoPi->p_pantalla));

    // Pinta la pala
      PintaPala(
            (tipo_pala*)(&(p_arkanoPi->pala)),
            (tipo_pantalla*)(p_arkanoPi->p_pantalla));

      // Pinta la pelota
      PintaPelota(
            (tipo_pelota*)(&(p_arkanoPi->pelota)),
            (tipo_pantalla*)(p_arkanoPi->p_pantalla));
}

/* Funcion encargada de actualizar */
/* el  en la pantall auxiliar */
void ActualizaPantallaScoreArkanoPi(tipo_arkanoPi* p_arkanoPi) {
      int s_score[1];
      s_score[0] = p_arkanoPi->score;
      display_score(s_score, 1);
}

void InicializaArkanoPi(tipo_arkanoPi *p_arkanoPi) {
      ResetArkanoPi(p_arkanoPi);
      ActualizaPantalla(p_arkanoPi);
}

void ResetArkanoPi(tipo_arkanoPi *p_arkanoPi) {
      ReseteaPantalla((tipo_pantalla*)(p_arkanoPi->p_pantalla));
      InicializaLadrillos((tipo_pantalla*)(&(p_arkanoPi->ladrillos)));
      InicializaPelota((tipo_pelota*)(&(p_arkanoPi->pelota)));
```

```
	InicializaPala((tipo_pala*)(&(p_arkanoPi->pala)));
}

int CompruebaReboteLadrillo (tipo_arkanoPi *p_arkanoPi) {
	int p_posible_ladrillo_x = 0;
	int p_posible_ladrillo_y = 0;

	p_posible_ladrillo_x = p_arkanoPi->pelota.x +
p_arkanoPi->pelota.trayectoria.xv;

	if ( ( p_posible_ladrillo_x < 0) || ( p_posible_ladrillo_x >=
NUM_COLUMNAS_DISPLAY ) ) {
		printf("\n\nERROR GRAVE!!! p_posible_ladrillo_x =
%d!!!\n\n", p_posible_ladrillo_x);
		fflush(stdout);
		exit(-1);
	}

	p_posible_ladrillo_y = p_arkanoPi->pelota.y +
p_arkanoPi->pelota.trayectoria.yv;

	if ( ( p_posible_ladrillo_y < 0) || ( p_posible_ladrillo_y >=
NUM_FILAS_DISPLAY ) ) {
		printf("\n\nERROR GRAVE!!! p_posible_ladrillo_y =
%d!!!\n\n", p_posible_ladrillo_y);
		fflush(stdout);
	}


if(p_arkanoPi->ladrillos.matriz[p_posible_ladrillo_y][p_posible_ladrill
o_x] > 0 ) {
		// La pelota ha entrado en el area de ladrillos
		// y descontamos el numero de golpes que resta para destruir
el ladrillo

p_arkanoPi->ladrillos.matriz[p_posible_ladrillo_y][p_posible_ladrillo_x
] =
p_arkanoPi->ladrillos.matriz[p_posible_ladrillo_y][p_posible_ladrillo_x
] - 1;

		return 1;
	}
```

```
	return 0;
}

int CompruebaFallo (tipo_arkanoPi arkanoPi) {
	// Comprobamos si no hemos conseguido devolver la pelota
	if(arkanoPi.pelota.y + arkanoPi.pelota.trayectoria.yv >=
NUM_FILAS_DISPLAY) {
		// Hemos fallado
		return 1;
	}
	return 0;
}

int CalculaLadrillosRestantes(tipo_pantalla *p_ladrillos) {
	int i=0, j=0;
	int numLadrillosRestantes;

	numLadrillosRestantes = 0;

	for(i=0;i<NUM_FILAS_DISPLAY;i++) {
		for(j=0;j<NUM_COLUMNAS_DISPLAY;j++) {
			if(p_ladrillos->matriz[i][j] != 0) {
				numLadrillosRestantes++;
			}
		}
	}

	return numLadrillosRestantes;
}

//------------------------------------------------------
// FUNCIONES DE ACCION DE LA MAQUINA DE ESTADOS
//------------------------------------------------------

/* void ExitArkano (void): funcion engarda */
/* de resetear las flags para volver a la pantalla */
void ExitArkano (fsm_t* this) {
	tipo_arkanoPi* p_arkanoPi;
	p_arkanoPi = (tipo_arkanoPi*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags = 0; // Reseteamos todas las flags
```

```
	piUnlock(SYSTEM_FLAGS_KEY);
}

// void MuevePalaIzquierda (void): funcion encargada de ejecutar
// el movimiento hacia la izquierda contemplado para la pala.
// Debe garantizar la viabilidad del mismo mediante la comprobación
// de que la nueva posición correspondiente a la pala 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 MuevePalaIzquierda (fsm_t* this) {
	tipo_arkanoPi* p_arkanoPi;
	p_arkanoPi = (tipo_arkanoPi*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_MOV_IZQUIERDA);
	piUnlock(SYSTEM_FLAGS_KEY);

	ActualizaPosicionPala(&(p_arkanoPi->pala), IZQUIERDA);
	fflush(stdout);

	piLock(MATRIX_KEY); // CLAVE PANTALLA
	ActualizaPantalla(p_arkanoPi);
	piUnlock(MATRIX_KEY); // CLAVE PANTALLA

	piLock(STD_IO_BUFFER_KEY); // CLAVE E/S STD
	PintaPantallaPorTerminal(p_arkanoPi->p_pantalla);
	piUnlock(STD_IO_BUFFER_KEY); // CLAVE E/S STD

}

// void MuevePalaDerecha (void): función similar a la anterior
// encargada del movimiento hacia la derecha.

void MuevePalaDerecha (fsm_t* this) {
	tipo_arkanoPi* p_arkanoPi;
	p_arkanoPi = (tipo_arkanoPi*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_MOV_DERECHA);
```

```
	piUnlock(SYSTEM_FLAGS_KEY);

	ActualizaPosicionPala(&(p_arkanoPi->pala), DERECHA);

	piLock(MATRIX_KEY); // CLAVE PANTALLA
	ActualizaPantalla(p_arkanoPi);
	piUnlock(MATRIX_KEY); // CLAVE PANTALLA

	piLock(STD_IO_BUFFER_KEY); // CLAVE E/S STD
	PintaPantallaPorTerminal(p_arkanoPi->p_pantalla);
	piUnlock(STD_IO_BUFFER_KEY); // CLAVE E/S STD
}

// void PausaJuego (void): función encargada de pausar el juego
void PausarJuego (fsm_t* this) {
	tipo_arkanoPi *p_arkanoPi;
	p_arkanoPi = (tipo_arkanoPi*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_PAUSA_JUEGO);
	piUnlock(SYSTEM_FLAGS_KEY);

	int i,j;
	for(i=0;i<NUM_FILAS_DISPLAY;i++) {
		for(j=0;j<NUM_COLUMNAS_DISPLAY;j++) {
			p_arkanoPi->p_pantalla->matriz[i][j] =
pantalla_pausa.matriz[i][j];
		}
	}
}

/* void ContinuaJuego (void): función encargada */
/* de continuar el juego tras una pausa */
void ContinuarJuego (fsm_t* this) {
	tipo_arkanoPi *p_arkanoPi;
	p_arkanoPi = (tipo_arkanoPi*)(this->user_data);

	// Hacemos una cuenta atrás de 3 segundos
	display_countdown(3, 1000);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_PAUSA_JUEGO);
```

```
	piUnlock(SYSTEM_FLAGS_KEY);

	// Actualizamos el juego al momento
	ActualizarJuego(this);
}

//-------------------------------------------------------
// FUNCIONES DE TRANSICION DE LA MAQUINA DE ESTADOS
//-------------------------------------------------------
int CompruebaInicioArkanoPi(fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
	result = (flags & FLAG_JUEGO_ARKANOPI);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

//-------------------------------------------------------
// FUNCIONES DE ACCION DE LA MAQUINA DE ESTADOS
//-------------------------------------------------------

// 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(fsm_t* this) {
	tipo_arkanoPi *p_arkanoPi;
	p_arkanoPi = (tipo_arkanoPi*)(this->user_data);

	p_arkanoPi->score = 0;

	p_arkanoPi->tmr_actualizacion_juego_isr =
tmr_new(tmr_actualizacion_juego_isr);

	piLock (STD_IO_BUFFER_KEY);
	InicializaArkanoPi(p_arkanoPi);

	PintaMensajeInicialPantalla(p_arkanoPi->p_pantalla,
p_arkanoPi->p_pantalla);
	PintaPantallaPorTerminal(p_arkanoPi->p_pantalla);
```

```
	piUnlock (STD_IO_BUFFER_KEY);

	/* pseudoWiringPiEnableDisplay(1); */
}

// void StartJuego (void): funcion encargada de llevar a cabo
// la oportuna inicialización de toda variable o estructura de datos
// que resulte necesaria para el comienzo del juego.

void StartJuego(fsm_t* this) {
	printf("Starting ArkanoPi\n");

	tipo_arkanoPi *p_arkanoPi;
	p_arkanoPi = (tipo_arkanoPi*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_BOTON);
	piUnlock(SYSTEM_FLAGS_KEY);

	// Reiniciamos la puntuación al final
	// de cada ronda
	p_arkanoPi->score = 0;

	piLock (STD_IO_BUFFER_KEY);
	InicializaArkanoPi(p_arkanoPi);

	tmr_startms(p_arkanoPi->tmr_actualizacion_juego_isr,
TIMEOUT_ACTUALIZA_JUEGO);

	PintaMensajeInicialPantalla(p_arkanoPi->p_pantalla,
p_arkanoPi->p_pantalla);
	PintaPantallaPorTerminal(p_arkanoPi->p_pantalla);

	piUnlock (STD_IO_BUFFER_KEY);


	/* pseudoWiringPiEnableDisplay(1); */
}

// void ActualizarJuego (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 ActualizarJuego (fsm_t* this) {
	tipo_arkanoPi* p_arkanoPi;
	p_arkanoPi = (tipo_arkanoPi*)(this->user_data);

	piLock (SYSTEM_FLAGS_KEY);
	flags &= (~ FLAG_TIMER_JUEGO);
	piUnlock (SYSTEM_FLAGS_KEY);

	if(CompruebaReboteParedesVerticales(p_arkanoPi->pelota)) {
		p_arkanoPi->pelota.trayectoria.xv =
-p_arkanoPi->pelota.trayectoria.xv;
	}

	if(CompruebaReboteTecho(p_arkanoPi->pelota)) {
		p_arkanoPi->pelota.trayectoria.yv =
-p_arkanoPi->pelota.trayectoria.yv;
	}

	if(CompruebaFallo (*p_arkanoPi)) {
		piLock (SYSTEM_FLAGS_KEY);
		flags |= FLAG_FIN_JUEGO;
		piUnlock (SYSTEM_FLAGS_KEY);
		return;

	} else if (CompruebaRebotePala (p_arkanoPi->pelota,
p_arkanoPi->pala)) {
		switch(p_arkanoPi->pelota.x +
p_arkanoPi->pelota.trayectoria.xv - p_arkanoPi->pala.x) {

		case 0:

CambiarDireccionPelota(&(p_arkanoPi->pelota),ARRIBA_IZQUIERDA);
```

```
            break;

            case 1:
            CambiarDireccionPelota(&(p_arkanoPi->pelota), ARRIBA);
            break;

            case 2:
            CambiarDireccionPelota(&(p_arkanoPi->pelota),
ARRIBA_DERECHA);
            break;

            }
      }
      if (CompruebaReboteLadrillo(p_arkanoPi)) {
            p_arkanoPi->pelota.trayectoria.yv =
-p_arkanoPi->pelota.trayectoria.yv;
            p_arkanoPi->score ++;
            if(CalculaLadrillosRestantes(&(p_arkanoPi->ladrillos))<= 0)
{
                  piLock (SYSTEM_FLAGS_KEY);
                  flags |= FLAG_FIN_JUEGO;
                  piUnlock (SYSTEM_FLAGS_KEY);
                  return;
      }
      }

      ActualizaPosicionPelota (&(p_arkanoPi->pelota));

      ActualizaPantallaScoreArkanoPi(p_arkanoPi);

      piLock(MATRIX_KEY);
      ActualizaPantalla (p_arkanoPi);
      piUnlock(MATRIX_KEY);

      piLock (STD_IO_BUFFER_KEY);
      PintaPantallaPorTerminal(p_arkanoPi->p_pantalla);
      piUnlock (STD_IO_BUFFER_KEY);

      tmr_startms(p_arkanoPi->tmr_actualizacion_juego_isr,
TIMEOUT_ACTUALIZA_JUEGO);

}
```

```
// void FinalJuego (void): función encargada de mostrar en la ventana
de
// terminal los mensajes necesarios para informar acerca del resultado
del juego.
void FinalJuego (fsm_t* this) {
	tipo_arkanoPi *p_arkanoPi;
	p_arkanoPi = (tipo_arkanoPi*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_FIN_JUEGO);
	piUnlock(SYSTEM_FLAGS_KEY);

	if(CalculaLadrillosRestantes(&(p_arkanoPi->ladrillos))==0){
		piLock(STD_IO_BUFFER_KEY);
		printf("FIN DEL JUEGO: HAS GANADO\n");
		piUnlock(STD_IO_BUFFER_KEY);
		display_text("You Win");
		ActualizaPantallaScoreArkanoPi(p_arkanoPi);

		int i,j;
		for(i=0;i<NUM_FILAS_DISPLAY;i++) {
			for(j=0;j<NUM_COLUMNAS_DISPLAY;j++) {
				p_arkanoPi->p_pantalla->matriz[i][j] =
pantalla_inicial.matriz[i][j];
			}
		}
	} else {
		piLock(STD_IO_BUFFER_KEY);
		printf("FIN DEL JUEGO: HAS PERDIDO\n");
		piUnlock(STD_IO_BUFFER_KEY);
		display_text("You Lose");
		ActualizaPantallaScoreArkanoPi(p_arkanoPi);

		int i,j;
		for(i=0;i<NUM_FILAS_DISPLAY;i++) {
			for(j=0;j<NUM_COLUMNAS_DISPLAY;j++) {
				p_arkanoPi->p_pantalla->matriz[i][j] =
pantalla_final.matriz[i][j];
			}
		}
	}
```

```
	/* pseudoWiringPiEnableDisplay(0); */
}

//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 (fsm_t* this) {
	tipo_arkanoPi *p_arkanoPi;
	p_arkanoPi = (tipo_arkanoPi*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_BOTON);
	piUnlock(SYSTEM_FLAGS_KEY);

	piLock (STD_IO_BUFFER_KEY);
	InicializaArkanoPi(p_arkanoPi);
	piUnlock (STD_IO_BUFFER_KEY);
}
```

3) CommonLib.c

```
#include "commonLib.h"
#include "ledDisplay.h"

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

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

	for(i=0;i<NUM_FILAS_DISPLAY;i++) {
		for(j=0;j<NUM_COLUMNAS_DISPLAY;j++) {
			p_pantalla->matriz[i][j] =
p_pantalla_inicial->matriz[i][j];
		}
	}

	return;
}
```

```
void PintaPantallaPorTerminal (tipo_pantalla *p_pantalla) {
#ifdef __SIN_wiringPi__
	int i=0, j=0;

	printf("\n[PANTALLA]\n");
	fflush(stdout);
	for(i=0;i<NUM_FILAS_DISPLAY;i++) {
		for(j=0;j<NUM_COLUMNAS_DISPLAY;j++) {
			printf("%d", p_pantalla->matriz[i][j]);
			fflush(stdout);
		}
		printf("\n");
		fflush(stdout);
	}
	fflush(stdout);
#endif
}

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

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

//------------------------------------------------------
// FUNCIONES DE INICIALIZACION / RESET
//------------------------------------------------------
void InicializaPelota(tipo_pelota *p_pelota) {
	// Aleatorizamos la posicion inicial de la pelota
	/*p_pelota->x = rand() % NUM_COLUMNAS_DISPLAY;*/
	/*p_pelota->y = 2 + rand() % (NUM_FILAS_DISPLAY-2); // 2 evita
que aparezca encima de ladrillos y para que no empiece demasiado pegada
al suelo de la pantalla*/

	// Pelota inicialmente en el centro de la pantalla
	p_pelota->x = NUM_COLUMNAS_DISPLAY/2 - 1;
	p_pelota->y = NUM_FILAS_DISPLAY/2 -1 ;
```

```
	InicializaPosiblesTrayectorias(p_pelota);

	// Trayectoria inicial
	//p_pelota->trayectoria.xv = 0;
	//p_pelota->trayectoria.yv = 1;
	/*CambiarDireccionPelota(p_pelota, rand() %
p_pelota->num_posibles_trayectorias);*/
	CambiarDireccionPelota(p_pelota, ABAJO);
}

void InicializaPala(tipo_pala *p_pala) {
	// Pala inicialmente en el centro de la pantalla
	p_pala->x = NUM_COLUMNAS_DISPLAY/2 - p_pala->ancho/2;
	p_pala->y = NUM_FILAS_DISPLAY - 1;
	p_pala->ancho = NUM_COLUMNAS_PALA;
	p_pala->alto = NUM_FILAS_PALA;
}

void InicializaPosiblesTrayectorias(tipo_pelota *p_pelota) {
	p_pelota->num_posibles_trayectorias = 0;
	p_pelota->posibles_trayectorias[ARRIBA_IZQUIERDA].xv = -1;
	p_pelota->posibles_trayectorias[ARRIBA_IZQUIERDA].yv = -1;
	p_pelota->num_posibles_trayectorias++;
	p_pelota->posibles_trayectorias[ARRIBA].xv = 0;
	p_pelota->posibles_trayectorias[ARRIBA].yv = -1;
	p_pelota->num_posibles_trayectorias++;
	p_pelota->posibles_trayectorias[ARRIBA_DERECHA].xv = 1;
	p_pelota->posibles_trayectorias[ARRIBA_DERECHA].yv = -1;
	p_pelota->num_posibles_trayectorias++;

	p_pelota->posibles_trayectorias[ABAJO_DERECHA].xv = 1;
	p_pelota->posibles_trayectorias[ABAJO_DERECHA].yv = 1;
	p_pelota->num_posibles_trayectorias++;
	p_pelota->posibles_trayectorias[ABAJO].xv = 0;
	p_pelota->posibles_trayectorias[ABAJO].yv = 1;
	p_pelota->num_posibles_trayectorias++;
	p_pelota->posibles_trayectorias[ABAJO_IZQUIERDA].xv = -1;
	p_pelota->posibles_trayectorias[ABAJO_IZQUIERDA].yv = 1;
	p_pelota->num_posibles_trayectorias++;

	//p_pelota->posibles_trayectorias[IZQUIERDA].xv = -1;
```

```
	//p_pelota->posibles_trayectorias[IZQUIERDA].yv = 0;
	//p_pelota->num_posibles_trayectorias++;
	//p_pelota->posibles_trayectorias[DERECHA].xv = 1;
	//p_pelota->posibles_trayectorias[DERECHA].yv = 0;
	//p_pelota->num_posibles_trayectorias++;
}

void PintaPala(tipo_pala *p_pala, tipo_pantalla *p_pantalla) {
	int i=0, j=0;

	for(i=0;i<NUM_FILAS_PALA;i++) {
		for(j=0;j<NUM_COLUMNAS_PALA;j++) {
			if (( (p_pala->y+i >= 0) && (p_pala->y+i <
NUM_FILAS_DISPLAY) ) &&
				( (p_pala->x+j >= 0) && (p_pala->x+j <
NUM_COLUMNAS_DISPLAY) ))
				p_pantalla->matriz[p_pala->y+i][p_pala->x+j] = 1;
		}
	}
}

void PintaPelota(tipo_pelota *p_pelota, tipo_pantalla *p_pantalla) {
	if( (p_pelota->x >= 0) && (p_pelota->x < NUM_COLUMNAS_DISPLAY) )
{
		if( (p_pelota->y >= 0) && (p_pelota->y < NUM_FILAS_DISPLAY)
) {
			p_pantalla->matriz[p_pelota->y][p_pelota->x] = 7;
		}
		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);
	}
}

int CompruebaReboteParedesVerticales (tipo_pelota pelota) {
```

```
	// Comprobamos si la nueva posicion de la pelota excede los
limites de la pantalla
	if((pelota.x + pelota.trayectoria.xv >= NUM_COLUMNAS_DISPLAY) ||
		(pelota.x + pelota.trayectoria.xv < 0) ) {
		// La pelota rebota contra la pared derecha o izquierda
		return 1;
	}
	return 0;
}

int CompruebaReboteTecho (tipo_pelota pelota) {
	// Comprobamos si la nueva posicion de la pelota excede los
limites de la pantalla
	if(pelota.y + pelota.trayectoria.yv < 0) {
		// La pelota rebota contra la pared derecha o izquierda
		return 1;
	}
	return 0;
}

int CompruebaRebotePala (tipo_pelota pelota, tipo_pala pala) {
		if ((pelota.x + pelota.trayectoria.xv >= pala.x ) &&
			(pelota.x + pelota.trayectoria.xv < pala.x +
NUM_COLUMNAS_PALA)) {

			if(pelota.trayectoria.yv > 0) { // Pelota hacia abajo
				// Rebote con pala inferior
				if ((pelota.y + pelota.trayectoria.yv >= pala.y)
&&
					(pelota.y + pelota.trayectoria.yv < pala.y
+ NUM_FILAS_PALA)) {
					return 1;
				}
			} else if(pelota.trayectoria.yv < 0) { // Pelota hacia
arriba
				// Rebote con pala superior
				if ((pelota.y + pelota.trayectoria.yv <= pala.y)
					&& (pelota.y + pelota.trayectoria.yv >
pala.y - NUM_FILAS_PALA)){
					return 1;
				}
			}
```

```
            }
      return 0;
}

void CambiarDireccionPelota(tipo_pelota *p_pelota, enum t_direccion
direccion) {
      if((direccion < 0)||(direccion >
p_pelota->num_posibles_trayectorias)) {
            printf("[ERROR!!!!][direccion NO VALIDA!!!!][%d]",
direccion);
            return;
      }
      else {
            p_pelota->trayectoria.xv =
p_pelota->posibles_trayectorias[direccion].xv;
            p_pelota->trayectoria.yv =
p_pelota->posibles_trayectorias[direccion].yv;
      }
}

void ActualizaPosicionPala(tipo_pala *p_pala, enum t_direccion
direccion) {
      switch (direccion) {
            case DERECHA:
                  // Dejamos que la pala rebase parcialmente el límite
del area de juego
                  if( p_pala->x + 1 + p_pala->ancho <=
NUM_COLUMNAS_DISPLAY + 2 )
                        p_pala->x = p_pala->x + 1;
                  break;
            case IZQUIERDA:
                  // Dejamos que la pala rebase parcialmente el límite
del area de juego
                  if( p_pala->x - 1 >= -2)
                              p_pala->x = p_pala->x - 1;
                  break;
            default:
                  printf("[ERROR!!!!][direccion NO VALIDA!!!!][%d]",
direccion);
                  break;
      }
```

```
}

void ActualizaPosicionPelota (tipo_pelota *p_pelota) {
	p_pelota->x += p_pelota->trayectoria.xv;
	p_pelota->y += p_pelota->trayectoria.yv;
}

//------------------------------------------------------
// FUNCIONES DE TRANSICION DE LA MAQUINA DE ESTADOS
//------------------------------------------------------
int CompruebaBotonPulsado(fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
	result = (flags & FLAG_BOTON);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

int CompruebaExit(fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
	result = (flags & FLAG_EXIT);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

int CompruebaMovimientoArriba(fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
	result = (flags & FLAG_MOV_ARRIBA);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

int CompruebaMovimientoAbajo(fsm_t* this) {
	int result = 0;
```

```
	piLock(SYSTEM_FLAGS_KEY);
	result = (flags & FLAG_MOV_ABAJO);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

int CompruebaMovimientoIzquierda(fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
	result = (flags & FLAG_MOV_IZQUIERDA);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

int CompruebaMovimientoDerecha(fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
	result = (flags & FLAG_MOV_DERECHA);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

int CompruebaTimeoutActualizacionJuego (fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
	result = (flags & FLAG_TIMER_JUEGO);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

int CompruebaPausaJuego(fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
```

```
	result = (flags & FLAG_PAUSA_JUEGO);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

int CompruebaFinalJuego(fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
	result = (flags & FLAG_FIN_JUEGO);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}


//------------------------------------------------------
// SUBRUTINAS DE ATENCION A LAS INTERRUPCIONES
//------------------------------------------------------

void tmr_actualizacion_juego_isr(union sigval value) {
	piLock(SYSTEM_FLAGS_KEY);
	flags |= FLAG_TIMER_JUEGO;
	piUnlock(SYSTEM_FLAGS_KEY);
}
```

4) Controller.c

```
# include "controller.h"

fsm_trans_t fsm_trans_selector[] = {
    { WAIT_PUSH, CompruebaMovimientoAbajo, WAIT_PUSH, SelectPrevGame
},
    { WAIT_PUSH, CompruebaMovimientoArriba, WAIT_PUSH, SelectNextGame
},
    { WAIT_PUSH, CompruebaBotonPulsado, WAIT_END, SelectGame },
    { WAIT_END, CompruebaExitGames, WAIT_PUSH, ExitGames },
    {-1, NULL, -1, NULL },
};

/* CompruebaExitGames(fsm_t* this) se encarga de comprobar que ninguno
```

```
*/
/* de los juegos se está corriendo */
int CompruebaExitGames(fsm_t* this) {
    TipoController *p_controller;
    p_controller = (TipoController*)(this->user_data);

     int result = 0;

     // Invertimos las flags de los juegos
     int flags_inversed = flags ^ (FLAG_JUEGO_ARKANOPI |
FLAG_JUEGO_PONG);
     // De esta forma observamos una flag HIGH cuando NO estamos
jugando

     piLock(SYSTEM_FLAGS_KEY);
     // Comprobamos si esa flag de NO jugar está activada
     result = (flags_inversed & FLAG_JUEGO_ARKANOPI) &&
(flags_inversed & FLAG_JUEGO_PONG);
     piUnlock(SYSTEM_FLAGS_KEY);

     return result;
}

void SelectNextGame(fsm_t* this){
    TipoController *p_controller;
    p_controller = (TipoController*)(this->user_data);

     piLock(SYSTEM_FLAGS_KEY);
     flags &= (~FLAG_MOV_ARRIBA);
     piUnlock(SYSTEM_FLAGS_KEY);

     // seleccionar el juego siguiente
     p_controller->game++;
     if(p_controller->game >= NUM_JUEGOS) p_controller->game = 0;

     // Mostrar el icono del juego seleccionado
     display_icon(p_controller->game);
}

void SelectPrevGame(fsm_t *this){
    TipoController *p_controller;
    p_controller = (TipoController*)(this->user_data);
```

```
      piLock(SYSTEM_FLAGS_KEY);
      flags &= (~FLAG_MOV_ABAJO);
      piUnlock(SYSTEM_FLAGS_KEY);

      // seleccionar el juego anterior
      p_controller->game--;
      if(p_controller->game < 0) p_controller->game = NUM_JUEGOS-1;

      // Mostrar el icono del juego seleccionado
      display_icon(p_controller->game);
      printf("Game %d chosen\n", p_controller->game);
}

// Función para confirmar la elección del juego
void SelectGame(fsm_t *this){
    TipoController *p_controller;
    p_controller = (TipoController*)(this->user_data);

      piLock(SYSTEM_FLAGS_KEY);
      flags &= (~FLAG_BOTON);
      piUnlock(SYSTEM_FLAGS_KEY);

      // Activamos la flag del juego correspondiente
      switch(p_controller->game){
            case ARKANOPI:
                  piLock(SYSTEM_FLAGS_KEY);
                  flags |= FLAG_JUEGO_ARKANOPI;
                  piUnlock(SYSTEM_FLAGS_KEY);
                  break;
            case PONG:
                  piLock(SYSTEM_FLAGS_KEY);
                  flags |= FLAG_JUEGO_PONG;
                  piUnlock(SYSTEM_FLAGS_KEY);
                  break;
      }
}

void ExitGames(fsm_t* this){
    TipoController *p_controller;
    p_controller = (TipoController*)(this->user_data);
```

```
	// volvemos a mostrar el icono inicial
	display_icon(HI_ICON);
}
```

5) LedDisplayAux.c

```
#include "ledDisplayAux.h"


#define NUM_COLUMN_DISPLAYAUX 8

uchar *data;
uchar display[8];

/* Load font */
#include "font-8x8.inc"

// void delay_xms: funcion para realizar un delay de x milisegundos
void Delay_xms(uint x)
{
	delay(x);
}

// int power: funcion auxiliar de calculo de potencia
int power(int x, int y){
	int res = 1;
	int i;
	for(i=0; i<y; i++) {
		res *=x;
	}
	return res;
}

// funcion auxiliar de separacion en unidades y decenas
int* split_num(int x){
	static int arr[2];

	arr[0]= x%10;
	arr[1]= (x/10);

	return arr;
}
```

```
// void Write_SPI: funcion de transmision de informacion
//con SPI al display
void Write_SPI(uchar reg, uchar data)
{
	uint8_t buf[2];
	buf[0] = reg;
	buf[1] = data;

	wiringPiSPIDataRW(SPI_CHANNEL, buf, 2);
}

// void ActualizaDisplayAux(): funcion para actualizar
// la imagen del display con los valores del array display
void ActualizaDisplayAux(){
	int i;
	for(i = 0; i < NUM_COLUMN_DISPLAYAUX; i++) {
		Write_SPI(i+1, display[i]);
	}
}

// void InitSPI(): Inicializacion del display
void InitSPI()
{
	if (wiringPiSPISetup(SPI_CHANNEL, CLOCK_SPEED) < 0) {
	    fprintf (stderr, "Unable to setup wiringPiSpi: %s\n",
strerror (errno)) ;
	    return;
	}

	Write_SPI(0x09,0x00); // decoder mode - no decode mode
	Write_SPI(0x0A,0xFF); // intensity - maximum value
	Write_SPI(0x0B,0x07); // limit scan - digits 01234567
	Write_SPI(0x0C,0x01); // shutdown - normal shutdown
	Write_SPI(0x0F,0x00); // display test - normal operation

	display_clear();
}

// void push(): funcion encargada de actualizar el array display
void push(char row){
	int i;
```

```
	for(i = 0; i < NUM_COLUMN_DISPLAYAUX - 1; i++) {
		display[i] = display[i+1];
	}
	display[NUM_COLUMN_DISPLAYAUX-1] = row;
}

//void display_clar: funcion encargada de apagar el display
void display_clear(){
	int i;
	for(i = 0; i < NUM_COLUMN_DISPLAYAUX; i++) {
		push(0);
	}
	ActualizaDisplayAux();
}

// void display_score: funcion encargada de mostrar la putuacion
void display_score(int score[], int nPlayers){
	if(nPlayers > 2) {
		printf("Too many characters to display\n");
		return;
	}

	// if(nPlayers == 1 && score[0]>=10) // do not show zero on the
left
	if(nPlayers == 1 ) {
		score = split_num(score[0]);
		nPlayers = 2;
	}

	int i;
	for (i = 0; i< NUM_COLUMN_DISPLAYAUX; i++){
		data = nums[score[0]][i];

		int player;
		for(player = 1; player < nPlayers; player++){
			data += nums[score[player]][i]*power(16, player) ;
		}
		push(data);
	}
	ActualizaDisplayAux();
}
```

```
//void display_coundown: funcion encargada de mostrar la cuenta atras
void display_countdown(int start, int delay){
	while(start >= 1){
		int i;
		const uint8_t* bits = &font[(start+48)*8];
		for (i = 0; i< NUM_COLUMN_DISPLAYAUX; i++){
			push(bits[i]);
		}
		ActualizaDisplayAux();
		Delay_xms(delay);
		start--;
	}
}

//void display_icon: funcion encargada de mostrar los iconos
void display_icon(int icon){
	printf("%d\n", icon);
	int i;
	for (i = 0; i< NUM_COLUMN_DISPLAYAUX; i++){
		data = icons[icon][i];
		push(data);
	}
	ActualizaDisplayAux();

}

// void display_text: funcion encargada de imprimir texto sobre la
pantalla
void display_text(char* text){
	int i = 0;
	for(i=0; i < strlen(text); i++) {
		char letter = text[i];

		int row;
		const uint8_t* bits = &font[letter*8];
		for(row=0; row<NUM_COLUMN_DISPLAYAUX; row++){
			push(bits[row]);
			ActualizaDisplayAux();
			Delay_xms(80);
		}
		push(0);
		ActualizaDisplayAux();
```

```
            Delay_xms(80);

      }

}
```

6) PongLib.c

```
void ExitPong (fsm_t* this) {
      tipo_pong* p_pong;
      p_pong = (tipo_pong*)(this->user_data);

      piLock(SYSTEM_FLAGS_KEY);
      flags = 0;
      piUnlock(SYSTEM_FLAGS_KEY);
}

// void MuevePalaIzquierda (void): funcion encargada de ejecutar
// el movimiento hacia la izquierda contemplado para la pala.
// Debe garantizar la viabilidad del mismo mediante la comprobación
// de que la nueva posición correspondiente a la pala 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 MuevePalaIzquierdaPong (fsm_t* this) {
      tipo_pong* p_pong;
      p_pong = (tipo_pong*)(this->user_data);

      piLock(SYSTEM_FLAGS_KEY);
      flags &= (~FLAG_MOV_IZQUIERDA);
      piUnlock(SYSTEM_FLAGS_KEY);

      ActualizaPosicionPala(&(p_pong->pala), IZQUIERDA);
      fflush(stdout);

      piLock(MATRIX_KEY); // CLAVE PANTALLA
      ActualizaPantallaPong(p_pong);
      piUnlock(MATRIX_KEY); // CLAVE PANTALLA

      piLock(STD_IO_BUFFER_KEY); // CLAVE E/S STD
```

```
	PintaPantallaPorTerminal(p_pong->p_pantalla);
	piUnlock(STD_IO_BUFFER_KEY); // CLAVE E/S STD

}

void MuevePalaIzquierdaPong2 (fsm_t* this) {
	tipo_pong* p_pong;
	p_pong = (tipo_pong*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_MOV_IZQUIERDA2);
	piUnlock(SYSTEM_FLAGS_KEY);

	ActualizaPosicionPala(&(p_pong->pala2), IZQUIERDA);
	fflush(stdout);

	piLock(MATRIX_KEY); // CLAVE PANTALLA
	ActualizaPantallaPong(p_pong);
	piUnlock(MATRIX_KEY); // CLAVE PANTALLA

	piLock(STD_IO_BUFFER_KEY); // CLAVE E/S STD
	PintaPantallaPorTerminal(p_pong->p_pantalla);
	piUnlock(STD_IO_BUFFER_KEY); // CLAVE E/S STD

}

// void MuevePalaDerecha (void): función similar a la anterior
// encargada del movimiento hacia la derecha.

void MuevePalaDerechaPong (fsm_t* this) {
	tipo_pong* p_pong;
	p_pong = (tipo_pong*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_MOV_DERECHA);
	piUnlock(SYSTEM_FLAGS_KEY);

	ActualizaPosicionPala(&(p_pong->pala), DERECHA);

	piLock(MATRIX_KEY); // CLAVE PANTALLA
	ActualizaPantallaPong(p_pong);
	piUnlock(MATRIX_KEY); // CLAVE PANTALLA
```

```
	piLock(STD_IO_BUFFER_KEY); // CLAVE E/S STD
	PintaPantallaPorTerminal(p_pong->p_pantalla);
	piUnlock(STD_IO_BUFFER_KEY); // CLAVE E/S STD
}
void MuevePalaDerechaPong2 (fsm_t* this) {
	tipo_pong* p_pong;
	p_pong = (tipo_pong*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_MOV_DERECHA2);
	piUnlock(SYSTEM_FLAGS_KEY);

	ActualizaPosicionPala(&(p_pong->pala2), DERECHA);

	piLock(MATRIX_KEY); // CLAVE PANTALLA
	ActualizaPantallaPong(p_pong);
	piUnlock(MATRIX_KEY); // CLAVE PANTALLA

	piLock(STD_IO_BUFFER_KEY); // CLAVE E/S STD
	PintaPantallaPorTerminal(p_pong->p_pantalla);
	piUnlock(STD_IO_BUFFER_KEY); // CLAVE E/S STD
}

// void PausaJuego (void): función encargada de pausar el juego
void PausarJuegoPong (fsm_t* this) {
	tipo_pong *p_pong;
	p_pong = (tipo_pong*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_PAUSA_JUEGO);
	piUnlock(SYSTEM_FLAGS_KEY);

	int i,j;
	for(i=0;i<NUM_FILAS_DISPLAY;i++) {
		for(j=0;j<NUM_COLUMNAS_DISPLAY;j++) {
			p_pong->p_pantalla->matriz[i][j] =
pantalla_pausa.matriz[i][j];
		}
	}
}
```

```
// void ContinuaJuego (void): función encargada de continuar el juego
tras una pausa
void ContinuarJuegoPong (fsm_t* this) {
	tipo_pong *p_pong;
	p_pong = (tipo_pong*)(this->user_data);

	display_countdown(3, 1000);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_PAUSA_JUEGO);
	piUnlock(SYSTEM_FLAGS_KEY);

	ActualizarJuegoPong(this);
}

//------------------------------------------------------
// FUNCIONES DE TRANSICION DE LA MAQUINA DE ESTADOS
//------------------------------------------------------
int CompruebaInicioPong (fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
	result = (flags & FLAG_JUEGO_PONG);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

int CompruebaMovimientoIzquierda2(fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
	result = (flags & FLAG_MOV_IZQUIERDA2);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

int CompruebaMovimientoDerecha2(fsm_t* this) {
	int result = 0;

	piLock(SYSTEM_FLAGS_KEY);
```

```
	result = (flags & FLAG_MOV_DERECHA2);
	piUnlock(SYSTEM_FLAGS_KEY);

	return result;
}

//-------------------------------------------------------
// FUNCIONES DE ACCION DE LA MAQUINA DE ESTADOS
//-------------------------------------------------------

// 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 InicializaJuegoPong(fsm_t* this) {
	tipo_pong *p_pong;
	p_pong = (tipo_pong*)(this->user_data);

	p_pong->tmr_actualizacion_juego_isr =
tmr_new(tmr_actualizacion_juego_isr);

	p_pong->score1 = 0;
	p_pong->score2 = 0;
	p_pong->score_round = 3;
	ActualizaPantallaScorePong(p_pong);

	piLock (STD_IO_BUFFER_KEY);

	InicializaPong(p_pong);

	tmr_startms(p_pong->tmr_actualizacion_juego_isr,
TIMEOUT_ACTUALIZA_JUEGO);

	PintaMensajeInicialPantalla(p_pong->p_pantalla,
p_pong->p_pantalla);
	PintaPantallaPorTerminal(p_pong->p_pantalla);

	piUnlock (STD_IO_BUFFER_KEY);


	/* pseudoWiringPiEnableDisplay(1); */
}
```

```
void StartJuegoPong(fsm_t* this) {
	printf("Starting Pong\n");

	tipo_pong *p_pong;
	p_pong = (tipo_pong*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_BOTON);
	piUnlock(SYSTEM_FLAGS_KEY);

	piLock (STD_IO_BUFFER_KEY);
	InicializaPong(p_pong);

	tmr_startms(p_pong->tmr_actualizacion_juego_isr,
TIMEOUT_ACTUALIZA_JUEGO);

	PintaMensajeInicialPantalla(p_pong->p_pantalla,
p_pong->p_pantalla);
	PintaPantallaPorTerminal(p_pong->p_pantalla);

	piUnlock (STD_IO_BUFFER_KEY);

	ActualizaPantallaScorePong(p_pong);


	/* pseudoWiringPiEnableDisplay(1); */
}

// void ActualizarJuego (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 ActualizarJuegoPong (fsm_t* this) {
	tipo_pong* p_pong;
	p_pong = (tipo_pong*)(this->user_data);

	piLock (SYSTEM_FLAGS_KEY);
	flags &= (~ FLAG_TIMER_JUEGO);
	piUnlock (SYSTEM_FLAGS_KEY);

	if(CompruebaReboteParedesVerticales(p_pong->pelota)) {
		p_pong->pelota.trayectoria.xv =
-p_pong->pelota.trayectoria.xv;
	}

	if(CompruebaPunto (p_pong)) {
		ActualizaPantallaScorePong(p_pong);
		if(p_pong->score1 >= p_pong->score_round){
			VictoriaPong(p_pong, 1);
		} else if(p_pong->score2 >= p_pong->score_round){
			VictoriaPong(p_pong, 2);
		} else {
			piLock(SYSTEM_FLAGS_KEY);
			flags |= FLAG_FIN_JUEGO;
			piUnlock(SYSTEM_FLAGS_KEY);
		}
		return;
	}
	else if (CompruebaRebotePala(p_pong->pelota, p_pong->pala)) {
		switch(p_pong->pelota.x + p_pong->pelota.trayectoria.xv -
p_pong->pala.x) {

		case 0:
		CambiarDireccionPelota(&(p_pong->pelota),ARRIBA_IZQUIERDA);
		break;

		case 1:
		CambiarDireccionPelota(&(p_pong->pelota), ARRIBA);
		break;

		case 2:
		CambiarDireccionPelota(&(p_pong->pelota), ARRIBA_DERECHA);
		break;
```

```
            }
      }
      else if (CompruebaRebotePala(p_pong->pelota, p_pong->pala2)) {
            switch(p_pong->pelota.x + p_pong->pelota.trayectoria.xv -
p_pong->pala2.x) {

            case 0:
            CambiarDireccionPelota(&(p_pong->pelota),ABAJO_IZQUIERDA);
            break;

            case 1:
            CambiarDireccionPelota(&(p_pong->pelota), ABAJO);
            break;

            case 2:
            CambiarDireccionPelota(&(p_pong->pelota), ABAJO_DERECHA);
            break;

            }
      }

      ActualizaPosicionPelota (&(p_pong->pelota));

      piLock(MATRIX_KEY);
      ActualizaPantallaPong (p_pong);
      piUnlock(MATRIX_KEY);

      piLock (STD_IO_BUFFER_KEY);
      PintaPantallaPorTerminal(p_pong->p_pantalla);
      piUnlock (STD_IO_BUFFER_KEY);

      tmr_startms(p_pong->tmr_actualizacion_juego_isr,
TIMEOUT_ACTUALIZA_JUEGO);

}

// void FinalJuego (void): función encargada de mostrar en la ventana
de
// terminal los mensajes necesarios para informar acerca del resultado
del juego.
void FinalJuegoPong (fsm_t* this) {
      tipo_pong *p_pong;
```

```
	p_pong = (tipo_pong*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_FIN_JUEGO);
	piUnlock(SYSTEM_FLAGS_KEY);

	/* pseudoWiringPiEnableDisplay(0); */
}

//void VictoriaPong (void): función encargada de tratar con una
victoria
void VictoriaPong(tipo_pong* p_pong, int player){
		char victory_text[15];
		sprintf(victory_text, "YOU WIN PLAYER%d", player);
		printf("%s\n", victory_text);
		display_text(victory_text);

		p_pong->score1 = 0;
		p_pong->score2 = 0;

		piLock(SYSTEM_FLAGS_KEY);
		flags |= FLAG_FIN_JUEGO;
		piUnlock(SYSTEM_FLAGS_KEY);
}

//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 ReseteaJuegoPong (fsm_t* this) {
	tipo_pong *p_pong;
	p_pong = (tipo_pong*)(this->user_data);

	piLock(SYSTEM_FLAGS_KEY);
	flags &= (~FLAG_BOTON);
	piUnlock(SYSTEM_FLAGS_KEY);

	piLock (STD_IO_BUFFER_KEY);
	InicializaPong(p_pong);
	piUnlock (STD_IO_BUFFER_KEY);
}
```

7) TECLADO_TL04.C

```
#include "teclado_TL04.h"

char tecladoTL04[4][4] = {
     {'1', '2', '3', 'C'},
     {'4', '5', '6', 'D'},
     {'7', '8', '9', 'E'},
     {'A', '0', 'B', 'F'}
};

// Maquina de estados: lista de transiciones
// {EstadoOrigen, CondicionDeDisparo, EstadoFinal, AccionesSiTransicion
}
fsm_trans_t fsm_trans_excitacion_columnas[] = {
     { TECLADO_ESPERA_COLUMNA, CompruebaTimeoutColumna,
TECLADO_ESPERA_COLUMNA, TecladoExcitaColumna },
     {-1, NULL, -1, NULL },
};

fsm_trans_t fsm_trans_deteccion_pulsaciones[] = {
     { TECLADO_ESPERA_TECLA, CompruebaTeclaPulsada,
TECLADO_ESPERA_TECLA, ProcesaTeclaPulsada},
     {-1, NULL, -1, NULL },
};


/*static TipoTeclado teclado;*/
int debounceTime[NUM_FILAS_TECLADO] = {0,0,0,0};
int rowPulsada;

//-------------------------------------------------------
// PROCEDIMIENTOS DE INICIALIZACION DE LOS OBJETOS ESPECIFICOS
//-------------------------------------------------------

void InicializaTeclado(TipoTeclado *p_teclado) {
     if (wiringPiSetupGpio() < 0) {
          fprintf (stderr, "Unable to setup wiringPi: %s\n", strerror
(errno)) ;
          return;
     }

     pinMode (GPIO_KEYBOARD_ROW_1, INPUT);
```

```
      pullUpDnControl(GPIO_KEYBOARD_ROW_1, PUD_DOWN);
      wiringPiISR (GPIO_KEYBOARD_ROW_1, INT_EDGE_RISING,
p_teclado->rutinas_ISR[0]);

      pinMode (GPIO_KEYBOARD_ROW_2, INPUT);
      pullUpDnControl(GPIO_KEYBOARD_ROW_2, PUD_DOWN);
      wiringPiISR (GPIO_KEYBOARD_ROW_2, INT_EDGE_RISING,
p_teclado->rutinas_ISR[1]);

      pinMode (GPIO_KEYBOARD_ROW_3, INPUT);
      pullUpDnControl(GPIO_KEYBOARD_ROW_3, PUD_DOWN);
      wiringPiISR (GPIO_KEYBOARD_ROW_3, INT_EDGE_RISING,
p_teclado->rutinas_ISR[2]);

      pinMode (GPIO_KEYBOARD_ROW_4, INPUT);
      pullUpDnControl(GPIO_KEYBOARD_ROW_4, PUD_DOWN);
      wiringPiISR (GPIO_KEYBOARD_ROW_4, INT_EDGE_RISING,
p_teclado->rutinas_ISR[3]);

      pinMode (GPIO_KEYBOARD_COL_1, OUTPUT);
      digitalWrite(GPIO_KEYBOARD_COL_1, LOW);

      pinMode (GPIO_KEYBOARD_COL_2, OUTPUT);
      digitalWrite(GPIO_KEYBOARD_COL_2, LOW);

      pinMode (GPIO_KEYBOARD_COL_3, OUTPUT);
      digitalWrite(GPIO_KEYBOARD_COL_3, LOW);

      pinMode (GPIO_KEYBOARD_COL_4, OUTPUT);
      digitalWrite(GPIO_KEYBOARD_COL_4, LOW);

      p_teclado->tmr_duracion_columna = tmr_new
(timer_duracion_columna_isr);

tmr_startms((tmr_t*)(p_teclado->tmr_duracion_columna),TIMEOUT_COLUMNA_T
ECLADO);
}

//-------------------------------------------------------
// OTROS PROCEDIMIENTOS PROPIOS DE LA LIBRERIA
//-------------------------------------------------------
```

```
void ActualizaExcitacionTecladoGPIO (int columna) {
	int gpio[NUM_COLUMNAS_TECLADO];
	gpio[COLUMNA_1] = GPIO_KEYBOARD_COL_1;
	gpio[COLUMNA_2] = GPIO_KEYBOARD_COL_2;
	gpio[COLUMNA_3] = GPIO_KEYBOARD_COL_3;
	gpio[COLUMNA_4] = GPIO_KEYBOARD_COL_4;

	switch(columna){
		case COLUMNA_1:
			digitalWrite(gpio[COLUMNA_4],0);
			digitalWrite(gpio[COLUMNA_1],1);
			break;
		case COLUMNA_2:
			digitalWrite(gpio[COLUMNA_1],0);
			digitalWrite(gpio[COLUMNA_2],1);
			break;
		case COLUMNA_3:
			digitalWrite(gpio[COLUMNA_2],0);
			digitalWrite(gpio[COLUMNA_3],1);
			break;
		case COLUMNA_4:
			digitalWrite(gpio[COLUMNA_3],0);
			digitalWrite(gpio[COLUMNA_4],1);
			break;
		default:
			break;
	}
}

//-------------------------------------------------------
// FUNCIONES DE ENTRADA O DE TRANSICION DE LA MAQUINA DE ESTADOS
//-------------------------------------------------------

int CompruebaTimeoutColumna (fsm_t* this) {
	int result = 0;
	TipoTeclado *p_teclado;
	p_teclado = (TipoTeclado*)(this->user_data);

	piLock(KEYBOARD_KEY);
	result = (teclado.flags & FLAG_TIMEOUT_COLUMNA_TECLADO);
	piUnlock(KEYBOARD_KEY);
```

```
	return result;
}

int CompruebaTeclaPulsada (fsm_t* this) {
	int result = 0;
	TipoTeclado *p_teclado;
	p_teclado = (TipoTeclado*)(this->user_data);

	piLock(KEYBOARD_KEY);
	result = (teclado.flags & FLAG_TECLA_PULSADA);
	piUnlock(KEYBOARD_KEY);

	return result;
}

//------------------------------------------------------
// FUNCIONES DE SALIDA O DE ACCION DE LAS MAQUINAS DE ESTADOS
//------------------------------------------------------

void TecladoExcitaColumna (fsm_t* this) {
	TipoTeclado *p_teclado;
	p_teclado = (TipoTeclado*)(this->user_data);

	piLock (KEYBOARD_KEY);
	teclado.flags &= (~ FLAG_TIMEOUT_COLUMNA_TECLADO);
	piUnlock (KEYBOARD_KEY);

	p_teclado->columna_actual += 1;
	if(p_teclado->columna_actual > COLUMNA_4) {
		p_teclado->columna_actual=COLUMNA_1;
		}
	// Llamada a ActualizaExcitacionTecladoGPIO con columna a activar
como argumento
	ActualizaExcitacionTecladoGPIO (p_teclado->columna_actual);

	tmr_startms(teclado.tmr_duracion_columna,
TIMEOUT_COLUMNA_TECLADO);


}

void ProcesaTeclaPulsada (fsm_t* this) {
```

```
	TipoTeclado *p_teclado;
	p_teclado = (TipoTeclado*)(this->user_data);

	piLock(KEYBOARD_KEY);
	teclado.flags &= (~FLAG_TECLA_PULSADA);
	piUnlock(KEYBOARD_KEY);

	switch(p_teclado->teclaPulsada.col){
		case COLUMNA_1: //
			if(p_teclado->teclaPulsada.row==0){
				piLock(SYSTEM_FLAGS_KEY);
				flags |= FLAG_EXIT;
				piUnlock(SYSTEM_FLAGS_KEY);
			}
			break;
		case COLUMNA_2: // tecla 0 (s14) movimiento izquierda
			if(p_teclado->teclaPulsada.row==3){
				piLock(SYSTEM_FLAGS_KEY);
				flags |= FLAG_MOV_IZQUIERDA;
				piUnlock(SYSTEM_FLAGS_KEY);
			}
			else if(p_teclado->teclaPulsada.row==0){
				piLock(SYSTEM_FLAGS_KEY);
				flags |= FLAG_MOV_IZQUIERDA2;
				piUnlock(SYSTEM_FLAGS_KEY);
			}
			break;

		case COLUMNA_3: // tecla B (s15) movimiento derecha
			if(p_teclado->teclaPulsada.row==3){
				piLock(SYSTEM_FLAGS_KEY);
				flags |= FLAG_MOV_DERECHA;
				piUnlock(SYSTEM_FLAGS_KEY);
			}
			else if(p_teclado->teclaPulsada.row==0){
				piLock(SYSTEM_FLAGS_KEY);
				flags |= FLAG_MOV_DERECHA2;
				piUnlock(SYSTEM_FLAGS_KEY);
			}
			break;

		case COLUMNA_4: // tecla C (s4) "start"
```

```
                if(p_teclado->teclaPulsada.row==0){
                      piLock(SYSTEM_FLAGS_KEY);
                      flags |= FLAG_BOTON;
                      piUnlock(SYSTEM_FLAGS_KEY);
                      fflush(stdout);
                }
                if(p_teclado->teclaPulsada.row==1){
                      piLock(SYSTEM_FLAGS_KEY);
                      flags |= FLAG_PAUSA_JUEGO;
                      piUnlock(SYSTEM_FLAGS_KEY);
                }
                if(p_teclado->teclaPulsada.row==2){
                      piLock(SYSTEM_FLAGS_KEY);
                      flags |= FLAG_MOV_ARRIBA;
                      piUnlock(SYSTEM_FLAGS_KEY);
                }
                if(p_teclado->teclaPulsada.row==3){
                      piLock(SYSTEM_FLAGS_KEY);
                      flags |= FLAG_MOV_ABAJO;
                      piUnlock(SYSTEM_FLAGS_KEY);
                }
                break;

           default:
                printf("\nERROR!!!! invalid number of column
(%d)!!!\n", p_teclado->columna_actual);
                fflush(stdout);

                break;
     }



}


//-------------------------------------------------------
// SUBRUTINAS DE ATENCION A LAS INTERRUPCIONES
//-------------------------------------------------------
void teclado_fila_1_isr (void) {
     if (millis() < teclado.debounceTime[FILA_1]){
          teclado.debounceTime[FILA_1]= millis() + DEBOUNCE_TIME;
```

```
            return;
      }

      piLock(KEYBOARD_KEY);

      teclado.flags |= FLAG_TECLA_PULSADA;

      teclado.teclaPulsada.row = FILA_1;
      teclado.teclaPulsada.col = teclado.columna_actual;
      piUnlock(KEYBOARD_KEY);

      debounceTime[FILA_1] = millis () + DEBOUNCE_TIME;

      //while(digitalRead(GPIO_KEYBOARD_ROW_1) ==HIGH){
      //    delay(debounceTime[FILA_1]);
            //delay(1);
      //}
};


void teclado_fila_2_isr (void) {
      if (millis() < teclado.debounceTime[FILA_2]){
            teclado.debounceTime[FILA_2]= millis() + DEBOUNCE_TIME;
            return;
      }

      piLock(KEYBOARD_KEY);
      teclado.teclaPulsada.row = FILA_2;
      teclado.teclaPulsada.col = teclado.columna_actual;
      teclado.flags |= FLAG_TECLA_PULSADA;
      piUnlock(KEYBOARD_KEY);

      teclado.debounceTime[FILA_2]= millis() + DEBOUNCE_TIME;

      //while(digitalRead(GPIO_KEYBOARD_ROW_2) ==HIGH){
            //delay(1);
      //    delay(debounceTime[FILA_2]);
      //}
};

void teclado_fila_3_isr (void) {
      if (millis() < teclado.debounceTime[FILA_3]){
```

```
            teclado.debounceTime[FILA_3]= millis() + DEBOUNCE_TIME;
            return;
      }

      piLock(KEYBOARD_KEY);
      teclado.flags |= FLAG_TECLA_PULSADA;
      teclado.teclaPulsada.row = FILA_3;
      teclado.teclaPulsada.col = teclado.columna_actual;
      piUnlock(KEYBOARD_KEY);

      teclado.debounceTime[FILA_3]= millis() + DEBOUNCE_TIME;

      //while(digitalRead(GPIO_KEYBOARD_ROW_3) ==HIGH){
            //delay(1);
      //    delay(debounceTime[FILA_3]);
      //}

};

void teclado_fila_4_isr (void) {
      if (millis() < teclado.debounceTime[FILA_4]){
            teclado.debounceTime[FILA_4]= millis() + DEBOUNCE_TIME;
            return;
      }

      piLock(KEYBOARD_KEY);
      teclado.teclaPulsada.row = FILA_4;
      teclado.teclaPulsada.col = teclado.columna_actual;
      teclado.flags |= FLAG_TECLA_PULSADA;
      piUnlock(KEYBOARD_KEY);

      debounceTime[FILA_4] = millis () + DEBOUNCE_TIME;

      //while(digitalRead(GPIO_KEYBOARD_ROW_4) ==HIGH){
            //delay(1);
      //    delay(debounceTime[FILA_4]);
      //}
};

void timer_duracion_columna_isr (union sigval value) {
      piLock(KEYBOARD_KEY);
      teclado.flags |= FLAG_TIMEOUT_COLUMNA_TECLADO;
```

```
      piUnlock(KEYBOARD_KEY);
};
```