DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA
UNIVERSIDAD POLITÉCNICA DE MADRID

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

Memoria del proyecto desarrollado en Sistemas Digitales II (SDII)

Curso 2016/2017

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

Autores (orden alfabético): Pablo Fernández Fernández

Santiago Romero Pomar

Código de la pareja: JT-13

ÍNDICE GENERAL

1 INTRODUCCIÓN 2

1.1 CARACTERÍSTICAS ESPECÍFICAS DEL APARTADO DE INTRODUCCIÓN 2

2 DIAGRAMA DE SUBSISTEMAS 3

3 DESCRIPCIÓN DEL SUBSISTEMA HARDWARE 4

3.1 DESCRIPCIÓN DEL MÓDULO ENTRADAS 4
3.2 DESCRIPCIÓN DEL MÓDULO SALIDAS 5

4 DESCRIPCIÓN DEL SUBSISTEMA SOFTWARE 6

4.1 FLUJO DE EJECUCIÓN DEL PROGRAMA PRINCIPAL 6
4.1.1 Funcionalidad o subrutina inicio de juego 7
4.1.2 Funcionalidad o subrutina movimiento pelota 7
4.1.3 Funcionalidad o subrutina movimiento raqueta derecha 8
4.1.4 Funcionalidad o subrutina movimiento raqueta izquierda 8
4.1.5 Funcionalidad o subrutina fin de juego 8
4.1.6 Funcionalidad o subrutina reinicio de juego 9
4.2 PROCESOS DE LAS INTERRUPCIONES 9
4.2.1 Interrupción del temporizador del refresco de columnas 9
4.2.2 Interrupción del temporizador del movimiento automático de la pelota 9
4.2.3 Interrupción del botón derecha 9
4.2.4 Interrupción del botón izquierda 9
4.2.5 Interrupción del botón "GO" 9
4.2.6 Interrupción del botón "EXIT" 10

5 DESCRIPCIÓN DE LAS MEJORAS 11

5.1 MEJORA NÚMERO DE IMPACTOS PARA ROMPER LADRILLOS 11
5.2 MEJORA NÚMERO DE VIDAS 11
5.3 MEJORA VELOCIDAD PELOTA 11
5.4 MEJORA PATRONES DE LADRILLOS 11
5.5 MEJORA DISTINTOS NIVELES 12
5.6 MEJORA PAUSA DE JUEGO 12
5.7 MEJORA EFECTOS DE SONIDO 13
5.8 MEJORA PONGPI 13
5.9 MEJORA SNAKEPI 14
5.10 MEJORA JUEGOSPI 14

6 PRINCIPALES PROBLEMAS ENCONTRADOS 15

7 MANUAL DE USUARIO 16

8 BIBLIOGRAFÍA 18

9 ANEXO I: CÓDIGO DEL PROGRAMA DEL PROYECTO FINAL 19

1 Introducción

1.1 Características específicas del apartado de Introducción

Objetivo:

El objetivo del proyecto es llevar a cabo la realización de una simulación del videojuego arcade ArkanoPi usando para ello una Raspberry Pi, además de la placa entrenadora del laboratorio (Raspberry Pi B+), el entorno Eclipse para desarrollar el software en C, la plataforma SmarTTY para la comunicación entre ambas, una matriz de leds como subsistema de salida y una placa de inserción para desarrollar el resto del hardware (buffers, decodificador de 4x16, resistencias de 220 Ω y pulsadores).

Descripción general:

Para llevar a cabo nuestro proyecto, hemos utilizado un programa desarrollado en C, el cual consiste en una máquina de estados de Mealy, en la que las salidas dependen de los estados de transición. Contamos con tres estados: WAIT_START, WAIT_PUSH y WAIT_END.

Funcionamiento:

Antes de empezar el juego aparece un mensaje inicial en la matriz de leds, en nuestro caso una "A". Tras accionar un pulsador concreto del mando (etiquetado con el mensaje "GO"), podremos observar las disposición de los ladrillos, la raqueta y la pelota. A continuación, según el pulsador del mando accionado, se recogen distintas funciones:

- etiquetado con el mensaje "←": la raqueta se desplaza hacia la izquierda.
- etiquetado con el mensaje "→": la raqueta se desplaza hacia la derecha.

Una regla adicional, elegida por nosotros, es que la raqueta sólo puede sobrepasar en una casilla los límites del juego. Cada vez que la pelota choque con una pared o la raqueta, ésta rebota. En caso de que choque con un ladrillo, aparte del rebote correspondiente, dicho ladrillo "se rompe" y desaparece. Existen dos maneras de terminar el juego, teniendo asociado cada una de ellas un mensaje: rompiendo todos los ladrillos (victoria, aparece una cara sonriente) o sobrepasando la pelota a la raqueta (derrota, aparece una cara triste). Para salir hemos implementado el pulsador "EXIT".

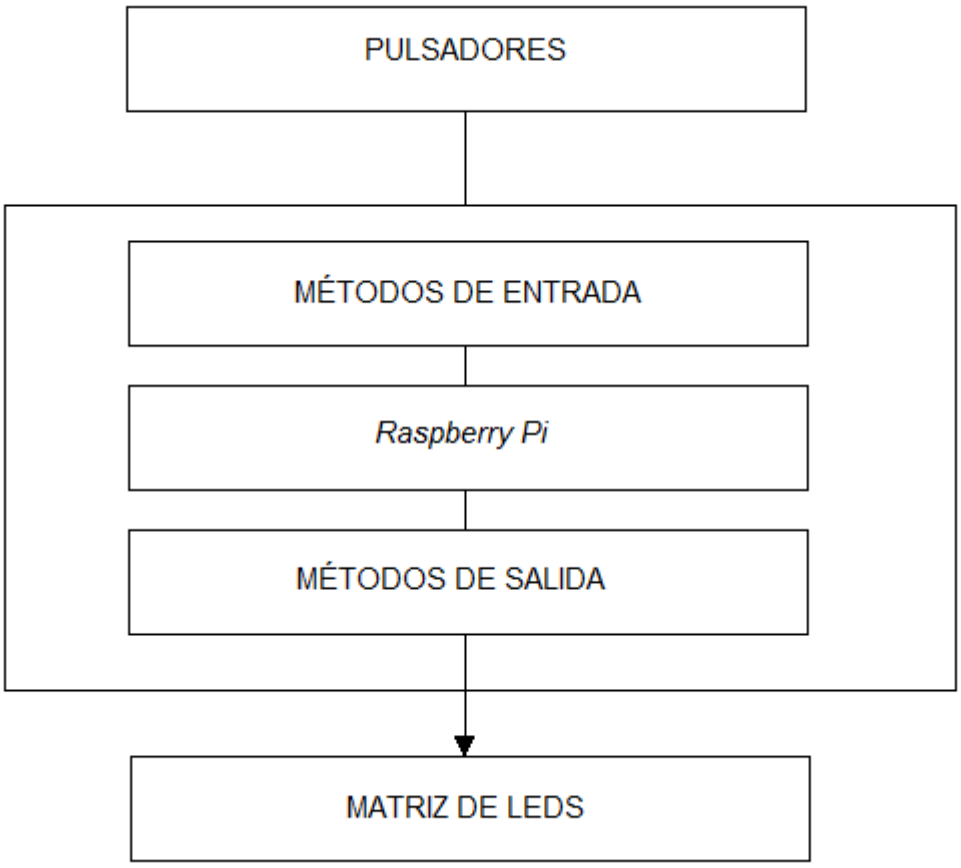
Interrupciones:

Para detectar las diferentes pulsaciones, hemos utilizado interrupciones, las cuales activaban un *flag*. Hemos utilizado 5 de ellos, una para cada uno de los movimientos de la raqueta, otro para el movimiento de la pelota (para las versiones anteriores, en las que se utilizaba una tecla del teclado para el movimiento de la misma), otro para el botón "GO" y el último para el final del juego. Adicionalmente, utilizamos un temporizador para el refresco de la matriz de leds por columnas y para el movimiento automático de la pelota.

Estructuras:

Para llevar a cabo estas implementaciones, hemos usado estructuras definidas y aportadas por los profesores, siendo éstas: *dprintf.h* para mostrar texto por terminal; *kbhit.c* y *kbhit.h*, para la lectura del programa de las teclas pulsadas; *fsm.c* y *fsm.h*, para la máquina de estados; y *tmr.c* y *tmr.h*, para el temporizador.

2 Diagrama de subsistemas



Métodos de entrada:

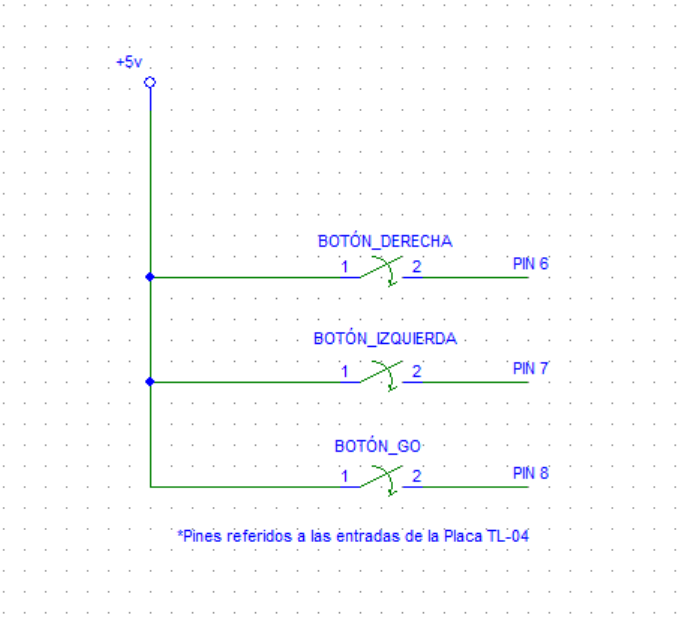
Con el fin de optimizar el encendido de la matriz de leds, hemos eliminado la hebra y su lectura. Dependiendo de qué pulsador haya sido accionado se actuará en consecuencia utilizando otros métodos.

Método de salida:

Se encarga del encendido de los leds según corresponda con la situación del juego. En nuestro caso lo hemos llamado *excitaColumna*, ya que se excita una columna cada 1 ms (*TIMEOUT*) por medio de un temporizador y se explora la situación de cada una de las filas.

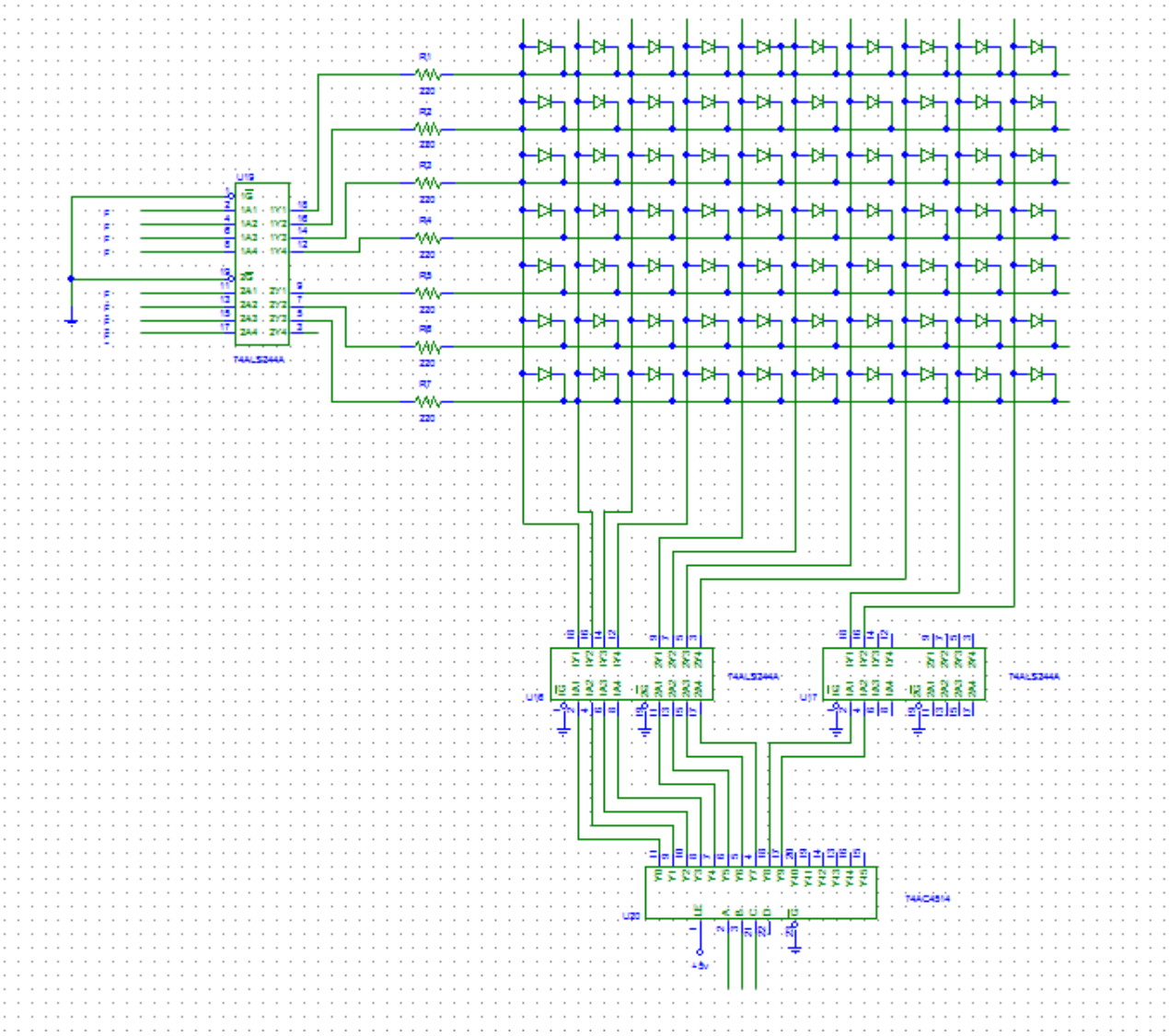
3 Descripción del subsistema Hardware

3.1 Descripción del módulo Entradas



Para el diseño de este módulo se ha escogido el esquema eléctrico anterior. El pulsador, al ser accionado entrega 5 V a la entrada de la Raspberry Pi, así detecta que ha habido una pulsación. Se podría haber realizado un diseño eléctrico alternativo, utilizando el mismo esquema, pero incluyendo resistencias, sin embargo, la placa entrenadora ya incluye las mismas y no es necesario que las incluyamos nosotros.

3.2 Descripción del módulo Salidas



Para el diseño de este módulo se ha escogido el esquema eléctrico anterior. Los valores son calculados teóricamente mediante la siguiente expresión:

$V_{out} = I_{led}*R + V_{led}$

Los datos han sido sacados del datasheet de la matriz de leds (I_{led} = 20 mA y V_{led} = 0.7 V) y sabiendo que la alimentación en la placa de inserción es de 5 V.

$R = (V_{out} - V_{led})/I_{led} = (5 - 0.7)\ V/20\ mA \approx 215\ \Omega$

Los valores comerciales utilizados en la práctica son de 220 Ω,, valor aproximado al valor teórico obtenido.

4 Descripción del subsistema Software

4.1 Flujo de ejecución del programa principal

En el diagrama de flujo del proceso se detallarán las principales rutinas realizadas por el proceso. Cada una de estas rutinas corresponderán a un módulo software que si el diseño ha sido realizado correctamente, será una subrutina (en ensamblador) o función del código (en C).

Una vez ejecutado el programa y antes de empezar el juego aparece un mensaje inicial en la matriz de leds, en nuestro caso una "A". Tras accionar un pulsador concreto del mando (etiquetado con el mensaje "GO"), podremos observar la disposición inicial de los elementos del juego. A partir de aquí dispondremos de varias opciones en cuanto al accionamiento de pulsadores: "←", para el movimiento de la raqueta hacia la izquierda o tecla "→", para el movimiento de la raqueta hacia la derecha. Para salir del programa hemos implementado el pulsador "EXIT". Al finalizar el juego, en función de si se ha superado la prueba o no, aparecerán mensajes por la matriz de leds y por pantalla, incluyendo el número de ladrillos restantes en caso de derrota. Para volver a jugar otra partida, únicamente será necesario pulsar dos veces el anteriormente mencionado botón "GO".

Para llevar a cabo el encendido de la matriz de leds, utilizaremos un método basado en un temporizador, el cual irá refrescando columna a columna cada milisegundo. Cada pulsador de movimiento tendrá asociado un *flag*, aparte del final del juego, que permitirá al programa seguir su función.

```
// FLAGS DEL SISTEMA
#define FLAG_TECLA                  0x01
#define FLAG_PELOTA                 0x02
#define FLAG_RAQUETA_DERECHA        0x04
#define FLAG_RAQUETA_IZQUIERDA      0x08
#define FLAG_FINAL_JUEGO            0x10
```

La ejecución el programa se realiza en el main(), en esta función creamos la máquina de estados, inicializándola; creamos también tanto el temporizador del refresco de las columnas como el del movimiento automático de la pelota, inicializando solamente el primero; la inicialización del hardware, tanto para las entradas como para las salidas y hacemos el bucle infinito que ejecutará la máquina de estados cada 10 ms, valor definido en la variable *CLK_MS*, que evitará en su totalidad la aparición de rebote y permitirá que el juego tenga un funcionamiento correcto, evitando que detecte pulsaciones innecesarias. Por último, se destruirá tanto la máquina de estados como los temporizadores.

```
/*Creación de la tabla de transiciones de la máquina de estados*/
fsm_trans_t fsm_tabla[] = {
      { WAIT_START, compruebaTeclaPulsada,  WAIT_PUSH, InicializaJuegoFSM },
      { WAIT_PUSH, compruebaTeclaPelota, WAIT_PUSH,   MovimientoPelotaFSM },
      { WAIT_PUSH, compruebaTeclaRaquetaDerecha, WAIT_PUSH,
MueveRaquetaDerechaFSM },
      { WAIT_PUSH, compruebaTeclaRaquetaIzquierda, WAIT_PUSH,
MueveRaquetaIzquierdaFSM },
      { WAIT_PUSH, compruebaFinalJuego, WAIT_END,   FinalJuegoFSM },
      { WAIT_END, compruebaTeclaPulsada, WAIT_START,   ReseteaJuegoFSM },
      { -1, NULL, -1, NULL },
};

/*Inicialización de la máquina de estados*/
fsm_t* inicio_fsm = fsm_new (WAIT_START, fsm_tabla, NULL);
```

```
/*Creación y activación de timers-------*/
tmr = tmr_new(excitaColumna);
tmr_startms(tmr, TIMEOUT);
tmr_pelota = tmr_new(pelota_auto);

/*Inicialización del Hardware*/
wiringPiSetupGpio();

/* Asociación de entradas del Hardware */
pinMode(GPIO_BOTON_DER,INPUT);
pullUpDnControl(GPIO_BOTON_DER, 0);
wiringPiISR(GPIO_BOTON_DER, INT_EDGE_FALLING, boton_derecho);

/* Asociación de salidas del Hardware */
pinMode(GPIO_FILA_0,OUTPUT);
```

4.1.1 Funcionalidad o subrutina inicio de juego

La funcionalidad de inicio de juego se da cuando se inicia el programa por primera vez. Esta subrutina tiene dos fases: el mensaje inicial y el comienzo del juego.

Mensaje inicial: al ejecutar el programa, a la salida, se iluminará el mensaje inicial en la matriz de leds.

Comienzo del juego: a la entrada, al accionar el pulsador "GO", se activa el *flag* correspondiente (*FLAG_TECLA*).

A la salida, se limpia el *flag* y se iluminarán en la matriz de leds los elementos del juego en sus posiciones iniciales.

```
void fsm_setup(fsm_t* inicio_fsm) {
      piLock (FLAGS_KEY);
      flags = 0;
      piUnlock (FLAGS_KEY);

      piLock (STD_IO_BUFFER_KEY);
      printf("\nINICIO DEL JUEGO\n");
      PintaMensajeInicialPantalla((tipo_pantalla*)(&juego.arkanoPi.pantalla),(ti
po_pantalla*)(&mensaje_inicial));
      PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
      piUnlock (STD_IO_BUFFER_KEY);
}

void InicializaJuegoFSM(fsm_t* fsm)
```

4.1.2 Funcionalidad o subrutina movimiento pelota

La funcionalidad de movimiento de la pelota, se da según un temporizador, cada determinado tiempo (*TIMEOUT_PELOTA*) se llama al método que hace que la pelota se mueva automáticamente, que a su vez llama al método encargado del movimiento de la pelota. Como se ha mencionado anteriormente, en versiones anteriores se activaba el *flag* correspondiente (*FLAG_PELOTA*).

A la salida, se limpia el *flag* (si se hubiera activado) y la posición de la pelota en la matriz de leds se verá modificada según la trayectoria de la misma.

Para esto, hemos dotado a la pelota de las variables de posición y trayectoria, siendo estas últimas modificadas cuando ocurra un rebote. Si ocurre con paredes laterales, la componente horizontal se verá invertida, mientras que contra el techo y los ladrillos, será la componente vertical la que se vea afectada. Un caso especial será cuando rebota contra la raqueta, la modificación dependerá de la zona de la raqueta contra la que golpee. Dicha funcionalidad ha sido lograda mediante el uso de múltiples condicionales.

```
void MovimientoPelotaFSM(fsm_t* fsm)
```

4.1.3 Funcionalidad o subrutina movimiento raqueta derecha

La funcionalidad de movimiento de la raqueta hacia la derecha, se da cuando se ha accionado el pulsador "→". A la entrada, al activar dicho pulsador, se activa el *flag* correspondiente (*FLAG_RAQUETA_DERECHA*).

A la salida, se limpia el *flag* y la posición de la raqueta en la matriz de leds se verá modificada en una posición hacia la derecha.

Para esto, mediante condicionales, hemos limitado el espacio del movimiento de la raqueta permitiendo que una posición de la misma salga de la matriz.

```
void MueveRaquetaDerechaFSM(fsm_t* fsm)
```

4.1.4 Funcionalidad o subrutina movimiento raqueta izquierda

La funcionalidad de movimiento de la raqueta hacia la izquierda, se da cuando se ha accionado el pulsador "←". A la entrada, al activar dicho pulsador, se activa el *flag* correspondiente (*FLAG_RAQUETA_IZQUIERDA*).

A la salida, se limpia el *flag* y la posición de la raqueta en la matriz de leds se verá modificada en una posición hacia la izquierda.

Para esto, mediante condicionales, hemos limitado el espacio del movimiento de la raqueta permitiendo que una posición de la misma salga de la matriz.

```
void MueveRaquetaIzquierdaFSM(fsm_t* fsm)
```

4.1.5 Funcionalidad o subrutina fin de juego

La funcionalidad de fin de juego, se puede dar en dos ocasiones: cuando la pelota sobrepasa la raqueta y cuando se han destruido todos los ladrillos. Cuando alguna de estas dos posibilidades ocurre, se activa el *flag* correspondiente (*FLAG_FINAL_JUEGO*).

A la salida, se limpia el *flag*, aparece un mensaje en la matriz de leds y un mensaje por pantalla, dando la enhorabuena o recordando el número de ladrillos restantes, según el caso.

```
void MovimientoPelotaFSM(fsm_t* fsm) {
    [...]

    if
(CalculaLadrillosRestantes((tipo_pantalla*)(&juego.arkanoPi.ladrillos.matriz))
== 0) { // no quedan más ladrillos
                piLock (FLAGS_KEY);
                flags |= FLAG_FINAL_JUEGO;
                piUnlock (FLAGS_KEY);
    }

    if (pelota_y == 5){
                if (pelota_yv > 0){

                    [...]

                    }else{
                        piLock (FLAGS_KEY);
                        flags |= FLAG_FINAL_JUEGO;
                        piUnlock (FLAGS_KEY);
                    }
                }
            }
}

void FinalJuegoFSM(fsm_t* fsm)
```

4.1.6 Funcionalidad o subrutina reinicio de juego

La funcionalidad de reinicio de juego se da cuando se reinicia el juego tras una victoria o una derrota. A la entrada, al accionar el pulsador "GO", se activa el *flag* correspondiente (*FLAG_TECLA*).

A la salida, se limpia el *flag* y se iluminará en la matriz de leds los elementos en sus posiciones iniciales. Sin embargo, no se podrá iniciar el juego hasta que no se vuelva a accionar el pulsador "GO" de nuevo (InicializaJuegoFSM).

```
void ReseteaJuegoFSM(fsm_t* fsm)
```

4.2 Procesos de las interrupciones

4.2.1 Interrupción del temporizador del refresco de columnas

Esta interrupción se activa una vez superado el tiempo máximo para la excitación de cada columna en el método *excitaColumna*. Este tiempo máximo está definido por la variable *TIMEOUT*, que en nuestro caso vale 1 milisegundo. Este valor se define en el *tmr_startms (tmr_startms(tmr, TIMEOUT);*). Una vez que este plazo se cumple, se atiende a la función *excitaColumna*.

4.2.2 Interrupción del temporizador del movimiento automático de la pelota

Esta interrupción se activa una vez superado el tiempo máximo para la llamada al método *pelota_auto*. Este tiempo máximo está definido por la variable *TIMEOUT_PELOTA*, que en nuestro caso vale 1000 milisegundos, es decir, 1 segundo. Este valor se define en el *tmr_startms (tmr_startms(tmr_pelota, TIMEOUT_PELOTA);*). Una vez que este plazo se cumple, se atiende a la función *pelota_auto*.

4.2.3 Interrupción del botón derecha

Esta interrupción se activa una vez que hemos acabado de pulsar el botón derecho, en ella se activa el *flag* correspondiente a dicho botón (*FLAG_RAQUETA_DERECHA*), que ocurre en una función llamada por *wiringPiISR(GPIO_BOTON_DER, INT_EDGE_FALLING, boton_derecho)*, siendo ésta *boton_derecho.*

4.2.4 Interrupción del botón izquierda

Esta interrupción se activa una vez que hemos acabado de pulsar el botón izquierdo, en ella se activa el *flag* correspondiente a dicho botón (*FLAG_RAQUETA_IZQUIERDA*), que ocurre en una función llamada por *wiringPiISR(GPIO_BOTON_IZQ, INT_EDGE_FALLING, boton_izquierdo)*, siendo ésta *boton_izquierdo.*

4.2.5 Interrupción del botón "GO"

Esta interrupción se activa una vez que hemos acabado de pulsar el botón "GO", en ella se activa el *flag* correspondiente a dicho botón (*FLAG_TECLA*), que ocurre en una función llamada por *wiringPiISR(GPIO_BOTON_TECLA, INT_EDGE_FALLING, boton_tecla)*, siendo ésta *boton_tecla.*

4.2.6 Interrupción del botón "EXIT"

Esta interrupción se activa una vez que hemos acabado de pulsar el botón "EXIT", en ella se ejecuta la sentencia *exit(0);*, que ocurre en una función llamada por *wiringPiISR(GPIO_BOTON_SALIDA, INT_EDGE_FALLING, boton_salida)*, siendo ésta *boton_salida.*

5 Descripción de las mejoras

5.1 Mejora número de impactos para romper ladrillos

En esta mejora, lo que hemos hecho ha sido pintar los ladrillos inicialmente con un valor mayor que 1. En el método del movimiento de la pelota hemos adaptado el condicional ya existente para identificar los choques con los ladrillos y la sentencia con la que rompíamos los mismos.

```
if (juego.arkanoPi.ladrillos.matriz [pelota_sig_x][pelota_sig_y] >= 1)
      juego.arkanoPi.ladrillos.matriz [pelota_sig_x][pelota_sig_y] -= 1;
```

5.2 Mejora número de vidas

En esta mejora, lo que hemos hecho ha sido crear una variable global (*vidas*) que lleve la cuenta del número de vidas que tiene el jugador, siendo este inicialmente igual a 3. En el método del movimiento de la pelota hemos adaptado el condicional para que cada vez que la pelota supere a la raqueta se pierda una vida y cuando éstas sean nulas, se termine el juego.

```
vidas--;
if (vidas == 0) {
      piLock (FLAGS_KEY);
      flags |= FLAG_FINAL_JUEGO;
      piUnlock (FLAGS_KEY);
} else {
      ReseteaPelota((tipo_pelota*)(&juego.arkanoPi.pelota));
      ReseteaRaqueta((tipo_raqueta*)(&juego.arkanoPi.raqueta));}
```

5.3 Mejora velocidad pelota

En esta mejora, lo que hemos hecho ha sido crear una variable global (*cnt_rebotes*) que lleva el contador de rebotes contra cualquier tipo de obstáculo (ladrillos, paredes y raqueta) para que cada cierto número de rebotes, 5 en nuestro caso, aumente la velocidad de la pelota reduciendo el tiempo de llamada del temporizador *tmr_pelota* hasta llegar a la velocidad máxima. Para esto, hemos utilizado otra variable global (*v_pelota*) cuyo valor representa el tiempo de llamada del anteriormente mencionado temporizador.

```
if ((cnt_rebotes % 5) == 0) {
      v_pelota -= 10;
      if (v_pelota <= 200){
            v_pelota = 200;}
}
```

5.4 Mejora patrones de ladrillos

En esta mejora, lo que hemos hecho ha sido cambiar la disposición en la que se presentan los ladrillos inicialmente.

```
int ladrillos_basico[MATRIZ_ANCHO][MATRIZ_ALTO] = {
            {0,1,0,0,0,0,0},
            {2,0,0,0,0,0,0},
            {0,1,0,0,0,0,0},
            {2,0,0,0,0,0,0},
            {0,1,0,0,0,0,0},
            {2,0,0,0,0,0,0},
            {0,1,0,0,0,0,0},
            {2,0,0,0,0,0,0},
            {0,1,0,0,0,0,0},
            {2,0,0,0,0,0,0},};
```

5.5 Mejora distintos niveles

En esta mejora, lo que hemos hecho ha sido establecer 3 niveles, cada uno más difícil que el anterior, apoyándonos en las mejoras anteriores: aumentando el número de vidas y la velocidad de la pelota, cambiando los patrones de los ladrillos y aumentando el número de impactos necesario para romperlos.

El funcionamiento es simple, al limpiar la pantalla de ladrillos aparecerá el siguiente nivel y cuando se supere el tercero y último se finalizará el juego con una victoria.

```
tipo_pantalla ladrillos_nv2 = {
                        {{3,2,0,0,0,0,0},
                        {3,0,0,0,0,0,0},
                        {3,2,0,0,0,0,0},
                        {3,0,0,0,0,0,0},
                        {3,2,0,0,0,0,0},
                        {3,0,0,0,0,0,0},
                        {3,2,0,0,0,0,0},
                        {3,0,0,0,0,0,0},
                        {3,2,0,0,0,0,0},
                        {3,0,0,0,0,0,0},}};

if(CalculaLadrillosRestantes((tipo_pantalla*)(&juego.arkanoPi.ladrillos.matriz))
== 0) {
      nivel++;
      if (nivel == 2){
            ReseteaMatriz((tipo_pantalla*)(&juego.arkanoPi.pantalla.matriz));
            juego.arkanoPi.ladrillos = ladrillos_nv2;
            PintaLadrillos((tipo_pantalla*)(&ladrillos_nv2),
(tipo_pantalla*)(&juego.arkanoPi.pantalla));
            ReseteaPelota((tipo_pelota*)(&juego.arkanoPi.pelota));
            ReseteaRaqueta((tipo_raqueta*)(&juego.arkanoPi.raqueta));
            cnt_rebotes = 1;
            v_pelota = TIMEOUT_PELOTA - 200;
            vidas++;
      }else if (nivel == 3){
             [...]
      }else if (nivel == 4){
            piLock (FLAGS_KEY);
            flags |= FLAG_FINAL_JUEGO;
            piUnlock (FLAGS_KEY); }}
```

5.6 Mejora pausa de juego

En esta mejora, lo que hemos hecho ha sido crear un nuevo estado *WAIT_PAUSE*, al cual se accede con la pulsación simultánea de los pulsadores encargados del movimiento de la raqueta. Se saldrá de él de la misma forma. Al entrar en dicho estado, se destruirá el temporizador encargado del movimiento automático de la pelota y al salir de él se vuelve a crear y a inicializar, ambas acciones acompañadas de sendos mensajes por pantalla.

```
void PauseIt (fsm_t *fsm) {
      piLock (FLAGS_KEY);
      flags &= ~(FLAG_RAQUETA_DERECHA || FLAG_RAQUETA_IZQUIERDA);
      piUnlock (FLAGS_KEY);

      piLock (STD_IO_BUFFER_KEY);
      printf("Press the buttons again to continue: ... ");
      ActualizaPantalla((tipo_arkanoPi*)(&juego.arkanoPi));
      PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
      piUnlock (STD_IO_BUFFER_KEY);

      tmr_destroy(tmr_pelota);}
```

```
void PlayIt (fsm_t *fsm) {
      piLock (FLAGS_KEY);
      flags &= ~(FLAG_RAQUETA_DERECHA || FLAG_RAQUETA_IZQUIERDA);
      piUnlock (FLAGS_KEY);

      piLock (STD_IO_BUFFER_KEY);
      printf("Continue... ");
      ActualizaPantalla((tipo_arkanoPi*)(&juego.arkanoPi));
      PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
      piUnlock (STD_IO_BUFFER_KEY);

      tmr_pelota = tmr_new(pelota_auto);
      tmr_startms(tmr_pelota, v_pelota);
}
void boton_derecho(void) {
      [...]
      piLock (FLAGS_KEY);
      flags |= FLAG_RAQUETA_DERECHA;
      piUnlock (FLAGS_KEY);

      // Wait for key to be released
      while (digitalRead (GPIO_BOTON_DER) == HIGH) {
            delay (1) ;
            if (digitalRead (GPIO_BOTON_IZQ) == HIGH) {
                  piLock (FLAGS_KEY);
                  flags |= (FLAG_RAQUETA_DERECHA || FLAG_RAQUETA_IZQUIERDA);
                  piUnlock (FLAGS_KEY);}
      }
      [...]
}
```

5.7 Mejora efectos de sonido

En esta mejora, lo que hemos hecho ha sido implementar efectos de sonido cuando la pelota impacta contra cualquier tipo de obstáculo, generando una frecuencia distinta para cada uno, mediante la librería *softPwm.h* perteneciente a *wiringPi.h*: utilizando los métodos *softPwmCreate* y *softPwmWrite*. En el *main()*, hemos definido un pin *GPIO* de salida para esta función, definiendo su rango de salida entre 0 y 100. Como timeout de la duración de estos efectos, hemos puesto a 0 los sonidos cada vez que se llame al método del movimiento de la pelota y en cada rebote se establece una frecuencia.

A nivel de hardware, hemos empleado un jack hembra conectado a la entrenadora mediante el pin establecido con la finalidad de introducir unos auriculares.

```
pinMode(GPIO_PWM,OUTPUT);
softPwmCreate(GPIO_PWM, 0, 100);

softPwmWrite(GPIO_PWM, 0);
softPwmWrite(GPIO_PWM, 75);
```

5.8 Mejora PongPi

En esta mejora, lo que hemos hecho ha sido crear el conocido juego "Pong" basándonos en la estructura del "Arkano", con la diferencia de eliminar los ladrillos pero añadiendo una raqueta más y utilizando la matriz horizontalmente (raquetas en columnas 0 y 9). Además, hemos introducido también mejoras ya existentes en el "Arkano": la velocidad de la pelota, la pausa y los efectos de sonido.

A nivel de software, ha habido que implementar métodos, tanto de movimiento como de pulsadores, para cada una de las raquetas, además del par de pausa y reanudación, lo que conlleva un estado adicional de pausa para cada jugador.

A nivel de hardware, hemos introducido un par de pulsadores adicionales para permitir un segundo jugador.

5.9 Mejora SnakePi

En esta mejora, lo que hemos hecho ha sido crear el conocido juego "Snake" basándonos en la estructura del "Arkano". Además, hemos introducido también mejoras ya existentes en el "Arkano": la velocidad de la serpiente (similar a la velocidad de la pelota), la pausa y los efectos de sonido.

A nivel de software, ha habido que implementar métodos, tanto de movimiento como de pulsadores, para cada uno de los posibles movimientos de la serpiente, además del par de pausa y reanudación, lo que conlleva un estado adicional de pausa para cada par de botones (Arriba-Abajo y Derecha-Izquierda). Por otro lado, las frutas que la serpiente debe comerse aparecen en posiciones aleatorias cada vez que las ya existentes son devoradas.

A nivel de hardware, hemos introducido un par de pulsadores adicionales para permitir el movimiento completo de la serpiente.

5.10 Mejora JuegosPi

En esta mejora, lo que hemos hecho ha sido crear un nuevo proyecto en el que los tres juegos están implementados en un mismo ejecutable. Para ello, ha sido necesario una máquina de estados en la que existen algunos comunes: *WAIT_START* y *WAIT_END*; mientras que cada juego tiene sus estados propios de *WAIT_PUSH_X* y *WAIT_PAUSE_X*.

Esta mejora engloba todas las mejoras anteriores tanto a nivel de software como de hardware.

6 Principales problemas encontrados

En este apartado se resumirán los principales problemas encontrados en la realización del proyecto.

A la hora de avanzar en el proyecto desde la entrega de la correspondiente memoria del hito intermedio, no se nos han presentado grandes dificultades aparte de algunas mejoras.

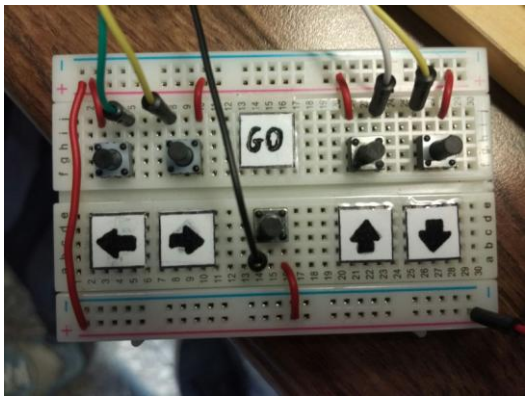
Cuando empezamos a implementar el juego "Snake" nos encontramos con la dificultad de hacer una serpiente "flexible", es decir, que no fuera rígida y pudiera girar e imitar el movimiento de una serpiente real. Éste fue uno de los mayores problemas, ya que en los anteriores juegos utilizábamos tanto estructuras como elementos comunes, sin embargo, en el saso de la serpiente era bastante diferente. Tras múltiples pruebas con infinidad de variables y bucles, podemos afirmar que hemos creado una serpiente cuyos movimientos son verosímiles. Aunque el juego está bastante logrado, en algunos momentos presenta algunos fallos.

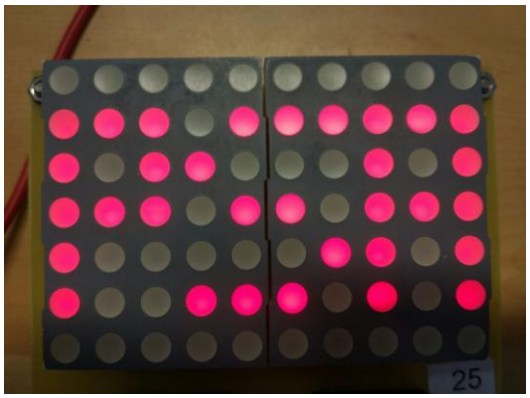
Finalmente, al querer juntar todos los juegos en un mismo proyecto, nos dimos cuenta que de la forma en que los habíamos implementado por separado, algunos métodos, *flags* y pines *GPIO* podrían ser utilizados por uno, dos o incluso los tres juegos. Por esta misma razón, aparecían múltiples conflictos, como por ejemplo que todos y cada uno de los juegos poseen un elemento *tipo_pantalla pantalla*, por lo que ha sido necesario crear otro objeto del mismo tipo que no pertenezca ninguno de los juegos, para que se vuelque en él todo lo que deba salir por la matriz de leds.

7 Manual de usuario

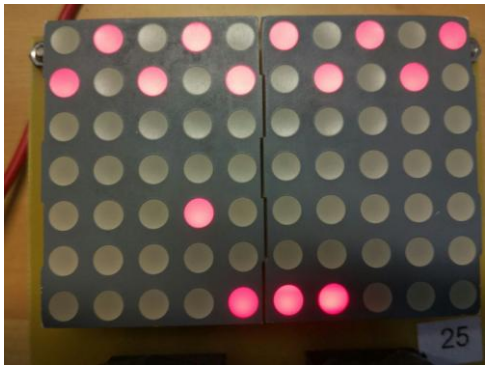
Breve descripción de cómo utilizar el sistema desarrollado desde el punto de vista de un usuario: funcionalidad de las teclas, posibilidades de configuración, opciones de funcionamiento,... Su extensión máxima es de **1 página**.

El sistema desarrollado sirve como plataforma de entretenimiento y consta de 3 juego distintos, siendo uno de ellos apto para dos jugadores simultáneos. Para que el usuario interactúe, dispone de un pequeño mando con 5 pulsadores y podrá seguir el desarrollo del juego por una pantalla formada por una matriz de 10x7 leds de color rojo. Además, se ha incluido una salida de audio por si quiere una experiencia completa con los efectos sonoros del juego.

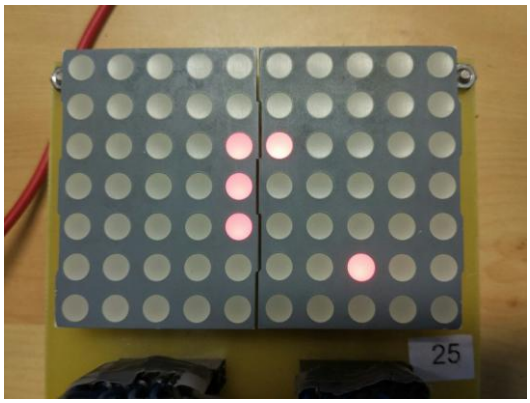




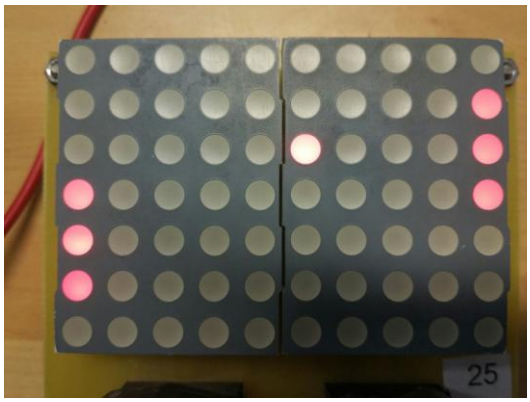
Una vez se ejecute el programa, tras observar un mensaje de recibimiento en la pantalla (PSA), el usuario deberá elegir a qué juego quiere acceder: si se decide por jugar al entretenido "Arkano", deberá accionar el pulsador "→"; mientras que si prefiere el archiconocido "Snake", será el etiquetado como "GO". Si quiere retar a un amigo a ver quién posee mejores reflejos, su elección no puede ser otra que el "Pong", accionando "←". Una vez se haya decidido, aparecerán los elementos del juego seleccionado en sus posiciones iniciales y comenzará el juego inmediatamente.



En el "Arkano", aparecen varios ladrillos que deberán ser destruidos por medio de una pelota y con la ayuda de una raqueta. Se permiten todo tipo de rebotes, pero la pelota no puede superar a la raqueta si no se quiere perder una vida de las 3 que se disponen inicialmente. El juego consta de 3 niveles en los que los ladrillos serán más resistentes y la pelota se moverá más rápidamente. La partida se acabará cuando se acaben los ladrillos del tercer nivel o las vidas del jugador.



En el "Snake", aparece una serpiente y una fruta que deberá devorar. Cada vez que esto ocurra, la serpiente crecerá y otro nuevo objetivo aparecerá. El juego acabará cuando la serpiente choque contra alguna de las paredes o contra sí misma.



En el "Pong", aparece una pelota y dos raquetas en los extremos izquierdo y derecho de la pantalla. La finalidad del juego es que la pelota supere la raqueta del contrincante utilizando la raqueta y ayudándose de las paredes superiores. El juego finalizará cuando esto ocurra.

Una vez terminado cualquiera de los juegos, aparecerá un mensaje en la pantalla, bien de victoria o bien de derrota. Para reiniciar la partida o para cambiar de juego, será necesario accionar el pulsador "GO". Volverá a aparecer el mensaje que aparecía al inicio y el sistema de elección de juegos vuelve a ser el mencionado.

Para abandonar una partida, se accionará el pulsador "EXIT".

8 Bibliografía

Listado de documentos en papel o electrónicos consultados a lo largo de la realización del proyecto. Se valorará que la bibliografía sea comentada como en el enunciado.

[1] Malagón, Pedro J. *"Introducción al entorno de desarrollo en C para Raspberry Pi"*. Enero 2017.
https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/939946/mod_resource/content/12/introducci%C3%B3n_alumnos_v1.3_angelfh.pdf
[2] De córdoba, Ricardo. *"Prácticas de lenguaje C para Raspberry Pi"*. Enero 2017.
https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/952826/mod_resource/content/5/Tutorialpractica2%20-%20v7.1.pdf
[3] Malagón, Pedro J.; Moya, José M.; Montero, Juan J.; Fernández Fernando. *"Introducción a las máquinas de estados en C para Raspberry Pi"*. Febrero 2017.
https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/966221/mod_resource/content/6/tutorial-FSMs-v4.0.pdf
[4] Pardo, José M.; San Segundo, Rubén; Fernández, Fernando; Malagón, Pedro J. *"Iniciación al Manejo de las Entradas/Salidas del BCM 2835"*. Marzo 2017.
https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/959983/mod_resource/content/8/Manejo_entradas_salidas-v4.1.pdf
[5] Fernández, Fernando; San Segundo, Rubén; Montero, Juan M.; Malagón, Pedro J.; Moya, José M.; Pardo, José M. *"Manejo de temporizaciones, interrupciones y procesos con la Raspberry Pi"*. Marzo 2017.
https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/962523/mod_resource/content/5/tutorial-interrupciones.4.1.pdf
[6] Kingbright. *"Hoja de especificaciones de la matriz de leds"*.
https://moodle.upm.es/titulaciones/oficiales/pluginfile.php/1245707/mod_resource/content/9/57155-hoja-especificaciones-matriz-leds.pdf
[7] Texas Instruments. *"SN74HC244N Datasheet"*.
http://www.ti.com/lit/ds/symlink/sn74hc244.pdf
[8] Philips. *"74HC4514N Datasheet"*.
http://pdf.datasheetcatalog.com/datasheet2/1/02a7za3p3qsr6f4l5xxwewj23oyy.pdf
[9] Wiring Pi. *"Software PWM Library"*.
http://wiringpi.com/reference/software-pwm-library/

9 ANEXO I: Código del programa del proyecto final

```
/** File name : arkanoPi_1.c
*     Description : Cuerpo del programa principal
*/

/*Includes----------------------------------------*/
#include "arkanoPi_1.h"

/*Private variables-------------------------------*/
static volatile tipo_juego juego;
volatile int flags = 0;
tmr_t* tmr;
tmr_t* tmr_pelota;
static int gpio_cols[4]={GPIO_COL_0,GPIO_COL_1,GPIO_COL_2,GPIO_COL_3};  // array de columnas
unsigned volatile int columna = 0;
int debounceTime;
fsm_t* inicio_fsm;
tipo_pantalla mensaje_inicial;
int cnt_rebotes = 1;
int v_pelota = TIMEOUT_PELOTA;
int vidas = 3;
int nivel = 1;
tipo_pantalla ladrillos_nv2;
tipo_pantalla ladrillos_nv3;
tipo_pantalla mensaje_victoria;
tipo_pantalla mensaje_derrota;

//------------------------------------------------------
// FUNCIONES DE ACCION
//------------------------------------------------------
/**
* @brief espera hasta la próxima activación del reloj
* @param next
*/
```

```
void delay_until (unsigned int next) {
      unsigned int now = millis();
      if (next > now) {
            delay (next - now);
    }
}

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

// FUNCIONES DE ENTRADA O COMPROBACIÓN DE LA MAQUINA DE ESTADOS
/**
* @brief función donde se comprueba la interrupción de la pulsación del boton_tecla
* @param this
* @return devuelve 1 si la iterrupción ha sido causada por esta pulsación; 0 en caso contrario
*/
int compruebaTeclaPulsada (fsm_t* this) {
      int result;

      piLock (FLAGS_KEY);
      result = (flags & FLAG_TECLA);
      piUnlock (FLAGS_KEY);

      return result;
}
/**
* @brief función donde se comprueba la interrupción de la pulsación del boton_derecho
* @param this
* @return devuelve 1 si la iterrupción ha sido causada por esta pulsación; 0 en caso contrario
*/
int compruebaTeclaRaquetaDerecha (fsm_t* this) {
      int result;

      piLock (FLAGS_KEY);
      result = (flags & FLAG_RAQUETA_DERECHA);
      piUnlock (FLAGS_KEY);

      return result;
}
```

```
/**
* @brief función donde se comprueba la interrupción de la pulsación del boton_izquierdo
* @param this
* @return devuelve 1 si la iterrupción ha sido causada por esta pulsación; 0 en caso contrario
*/
int compruebaTeclaRaquetaIzquierda (fsm_t* this) {
      int result;

      piLock (FLAGS_KEY);
      result = (flags & FLAG_RAQUETA_IZQUIERDA);
      piUnlock (FLAGS_KEY);

      return result;
}

/**
* @brief función donde se comprueba la interrupción de la pulsación simultánea del boton_derecho y boron_izquierdo
* @param this
* @return devuelve 1 si la iterrupción ha sido causada por estas pulsaciones; 0 en caso contrario
*/
int compruebaPausePlay (fsm_t* this) {
      int result;

      piLock (FLAGS_KEY);
      result = (flags & (FLAG_RAQUETA_DERECHA || FLAG_RAQUETA_IZQUIERDA));
      piUnlock (FLAGS_KEY);

      return result;
}

/**
* @brief función donde se comprueba la interrupción del final de juego
* @param this
* @return devuelve 1 si la iterrupción ha sido causada por el fin del juego; 0 en caso contrario
*/
```

```
int compruebaFinalJuego (fsm_t* this) {
	int result;

	piLock (FLAGS_KEY);
	result = (flags & FLAG_FINAL_JUEGO);
	piUnlock (FLAGS_KEY);

	return result;
}

// FUNCIONES DE SALIDA O ACCION DE LA MAQUINA DE ESTADOS

/**
* @brief rutina donde se inicia el juego, sacando por pantalla y dibujando ladrillos_basico,
*	junto con el inicio del temporizador del movimiento pelota
* @param fsm
*/
void InicializaJuegoFSM(fsm_t* fsm) {
	piLock (FLAGS_KEY);
	flags &= ~FLAG_TECLA;
	piUnlock (FLAGS_KEY);

	InicializaArkanoPi((tipo_arkanoPi*)(&juego.arkanoPi));
	tmr_startms(tmr_pelota, TIMEOUT_PELOTA);

	piLock (STD_IO_BUFFER_KEY);
	ActualizaPantalla((tipo_arkanoPi*)(&juego.arkanoPi));
	PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
	piUnlock (STD_IO_BUFFER_KEY);

}




/**
* @brief rutina donde se modifica la pelota en una posición siguiendo su trayectoria, pudiendo
*	avanzar de nivel, ganar o perder el juego.
* @param fsm
*/
```

```
void MovimientoPelotaFSM(fsm_t* fsm) {

      int pelota_x = juego.arkanoPi.pelota.x;
      int pelota_y = juego.arkanoPi.pelota.y;
      int pelota_xv = juego.arkanoPi.pelota.xv;
      int pelota_yv = juego.arkanoPi.pelota.yv;
      int pelota_sig_x = pelota_x + pelota_xv;
      int pelota_sig_y = pelota_y + pelota_yv;
      int raqueta_x = juego.arkanoPi.raqueta.x;
      int raqueta_y = juego.arkanoPi.raqueta.y;

      softPwmWrite(GPIO_PWM, 0);

      if (pelota_x == 0 || pelota_x == 9 || pelota_y == 0 || pelota_y == 5){
            if (pelota_x == 0){
                  if (pelota_xv < 0){
                        juego.arkanoPi.pelota.xv = 1;
                        pelota_xv = juego.arkanoPi.pelota.xv;
                        pelota_sig_x = pelota_x + pelota_xv;
                        cnt_rebotes++;
                        softPwmWrite(GPIO_PWM, 75);
                  }
            }
            if (pelota_x == 9){
                  if (pelota_xv > 0){
                        juego.arkanoPi.pelota.xv = -1;
                        pelota_xv = juego.arkanoPi.pelota.xv;
                        pelota_sig_x = pelota_x + pelota_xv;
                        cnt_rebotes++;
                        softPwmWrite(GPIO_PWM, 75);
                  }
            }
            if (pelota_y == 0){
                  if (pelota_yv < 0){
                        juego.arkanoPi.pelota.yv = 1;
                        pelota_yv = juego.arkanoPi.pelota.yv;
                        pelota_sig_y = pelota_y + pelota_yv;
                        cnt_rebotes++;
                        softPwmWrite(GPIO_PWM, 75);
                  }
```

```
}
if (pelota_y == 5){
      if (pelota_yv > 0){
            if ((pelota_sig_x == (raqueta_x + 2)) && (pelota_sig_y == raqueta_y)) { // derecha
                  if (pelota_x == 9){
                        juego.arkanoPi.pelota.xv = -1;
                        juego.arkanoPi.pelota.yv = -1;
                        pelota_xv = juego.arkanoPi.pelota.xv;
                        pelota_yv = juego.arkanoPi.pelota.yv;
                        pelota_sig_x = pelota_x + pelota_xv;
                        pelota_sig_y = pelota_y + pelota_yv;
                  }else{
                        juego.arkanoPi.pelota.xv = 1;
                        juego.arkanoPi.pelota.yv = -1;
                        pelota_xv = juego.arkanoPi.pelota.xv;
                        pelota_yv = juego.arkanoPi.pelota.yv;
                        pelota_sig_x = pelota_x + pelota_xv;
                        pelota_sig_y = pelota_y + pelota_yv;
                  }
                  cnt_rebotes++;
                  softPwmWrite(GPIO_PWM, 50);
            }else if ((pelota_sig_x == (raqueta_x + 1)) && (pelota_sig_y == raqueta_y)) { // centro
                        juego.arkanoPi.pelota.xv = 0;
                        juego.arkanoPi.pelota.yv = -1;
                        pelota_xv = juego.arkanoPi.pelota.xv;
                        pelota_yv = juego.arkanoPi.pelota.yv;
                        pelota_sig_x = pelota_x + pelota_xv;
                        pelota_sig_y = pelota_y + pelota_yv;
                        cnt_rebotes++;
                        softPwmWrite(GPIO_PWM, 50);
            }else if ((pelota_sig_x == (raqueta_x)) && (pelota_sig_y == raqueta_y)) { // izquierda
                  if (pelota_x == 0){
                        juego.arkanoPi.pelota.xv = 1;
                        juego.arkanoPi.pelota.yv = -1;
                        pelota_xv = juego.arkanoPi.pelota.xv;
                        pelota_yv = juego.arkanoPi.pelota.yv;
                        pelota_sig_x = pelota_x + pelota_xv;
                        pelota_sig_y = pelota_y + pelota_yv;
                  }else{
                        juego.arkanoPi.pelota.xv = -1;
```

```
                                    juego.arkanoPi.pelota.yv = -1;
                                    pelota_xv = juego.arkanoPi.pelota.xv;
                                    pelota_yv = juego.arkanoPi.pelota.yv;
                                    pelota_sig_x = pelota_x + pelota_xv;
                                    pelota_sig_y = pelota_y + pelota_yv;
                              }
                              cnt_rebotes++;
                              softPwmWrite(GPIO_PWM, 50);
                        }else{
                              juego.arkanoPi.pelota.y = 6;
                              vidas--;
                              if (vidas == 0) {
                                    piLock (FLAGS_KEY);
                                    flags |= FLAG_FINAL_JUEGO;
                                    piUnlock (FLAGS_KEY);
                              } else {
                                    ReseteaPelota((tipo_pelota*)(&juego.arkanoPi.pelota));
                                    ReseteaRaqueta((tipo_raqueta*) (&juego. arkanoPi.raqueta));

                                    cnt_rebotes = 1;
                                    v_pelota = TIMEOUT_PELOTA;
                              }
                        }
                  }
            }
      }
      if (juego.arkanoPi.ladrillos.matriz [pelota_sig_x][pelota_sig_y] >= 1){ // ladrillo
            juego.arkanoPi.ladrillos.matriz [pelota_sig_x][pelota_sig_y] -= 1;
            cnt_rebotes++;
            softPwmWrite(GPIO_PWM, 25);
            if (CalculaLadrillosRestantes((tipo_pantalla*)(&juego.arkanoPi.ladrillos.matriz)) == 0) { // no más ladrillos
                  nivel++;
                  if (nivel == 2){
                        ReseteaMatriz((tipo_pantalla*)(&juego.arkanoPi.pantalla.matriz));
                        juego.arkanoPi.ladrillos = ladrillos_nv2;
                        PintaLadrillos((tipo_pantalla*)(&ladrillos_nv2), (tipo_pantalla*)(&juego.arkanoPi.pantalla));
                        ReseteaPelota((tipo_pelota*)(&juego.arkanoPi.pelota));
                        ReseteaRaqueta((tipo_raqueta*)(&juego.arkanoPi.raqueta));
                        cnt_rebotes = 1;
                        v_pelota = TIMEOUT_PELOTA - 200;
```

```
                        vidas++;
                  }else if (nivel == 3){
                        ReseteaMatriz((tipo_pantalla*)(&juego.arkanoPi.pantalla.matriz));
                        juego.arkanoPi.ladrillos = ladrillos_nv3;
                        PintaLadrillos((tipo_pantalla*)(&ladrillos_nv3), (tipo_pantalla*)(&juego.arkanoPi.pantalla));
                        ReseteaPelota((tipo_pelota*)(&juego.arkanoPi.pelota));
                        ReseteaRaqueta((tipo_raqueta*)(&juego.arkanoPi.raqueta));
                        cnt_rebotes = 1;
                        v_pelota = TIMEOUT_PELOTA - 400;
                        vidas++;
                  }else if (nivel == 4){
                        softPwmWrite(GPIO_PWM, 0);

                        piLock (FLAGS_KEY);
                        flags |= FLAG_FINAL_JUEGO;
                        piUnlock (FLAGS_KEY);
                  }
            }
            if (pelota_y == 0){
                  juego.arkanoPi.pelota.xv = - pelota_xv;
                  pelota_xv = juego.arkanoPi.pelota.xv;
                  pelota_sig_x = pelota_x + pelota_xv;
            } else {
                  juego.arkanoPi.pelota.yv = - pelota_yv;
                  pelota_yv = juego.arkanoPi.pelota.yv;
                  pelota_sig_y = pelota_y + pelota_yv;
            }
      }

      if (juego.arkanoPi.pelota.y != 6){
            juego.arkanoPi.pelota.x += juego.arkanoPi.pelota.xv;
            juego.arkanoPi.pelota.y += juego.arkanoPi.pelota.yv;
      }

      piLock (STD_IO_BUFFER_KEY);
      ActualizaPantalla((tipo_arkanoPi*)(&juego.arkanoPi));
      PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
      piUnlock (STD_IO_BUFFER_KEY);
}
```

```
/**
* @brief método asignado al temporizador que hace mover la pelota automáticamente
*/
void pelota_auto(union sigval value) {
	if ((cnt_rebotes % 5) == 0) {
		v_pelota -= 10;
		if (v_pelota <= 200){
			v_pelota = 200;
		}
	}
	MovimientoPelotaFSM(inicio_fsm);
	tmr_startms(tmr_pelota, v_pelota);
}

/**
* @brief rutina donde se modifica la raqueta en una posición hacia la derecha
* @param fsm
*/
void MueveRaquetaDerechaFSM(fsm_t *fsm) {
	piLock (FLAGS_KEY);
	flags &= ~FLAG_RAQUETA_DERECHA;
	piUnlock (FLAGS_KEY);

	if(juego.arkanoPi.raqueta.x < (MATRIZ_ANCHO - RAQUETA_ANCHO)+1){
		juego.arkanoPi.raqueta.x = juego.arkanoPi.raqueta.x+1;
	} else {
		juego.arkanoPi.raqueta.x = (MATRIZ_ANCHO - RAQUETA_ANCHO)+1;
	}
	piLock (STD_IO_BUFFER_KEY);
	ActualizaPantalla((tipo_arkanoPi*)(&juego.arkanoPi));
	PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
	piUnlock (STD_IO_BUFFER_KEY);
}

/**
* @brief método asignado a la inturrupción por la pulsación del boton asignado,
*	junto con la posible interrupción de un segundo botón para pausar el juego
*/
```

```
void boton_derecho(void) {
	// Pin event (key / button event) debouncing procedure
	if (millis () < debounceTime){
		debounceTime = millis () + DEBOUNCE_TIME ;
		return;
	}
	piLock (FLAGS_KEY);
	flags |= FLAG_RAQUETA_DERECHA;
	piUnlock (FLAGS_KEY);
	// Wait for key to be released
	while (digitalRead (GPIO_BOTON_DER) == HIGH) {
		delay (1) ;
		if (digitalRead (GPIO_BOTON_IZQ) == HIGH) {
			piLock (FLAGS_KEY);
			flags |= (FLAG_RAQUETA_DERECHA || FLAG_RAQUETA_IZQUIERDA);
			piUnlock (FLAGS_KEY);
		}
	}
	debounceTime = millis () + DEBOUNCE_TIME ;
}

/**
* @brief rutina donde se modifica la raqueta en una posición hacia la izquierda
* @param fsm
*/
void MueveRaquetaIzquierdaFSM(fsm_t *fsm) {
	piLock (FLAGS_KEY);
	flags &= ~FLAG_RAQUETA_IZQUIERDA;
	piUnlock (FLAGS_KEY);

	if(juego.arkanoPi.raqueta.x > -1){
			juego.arkanoPi.raqueta.x = juego.arkanoPi.raqueta.x-1;
		} else {
			juego.arkanoPi.raqueta.x = -1;
		}
	piLock (STD_IO_BUFFER_KEY);
	ActualizaPantalla((tipo_arkanoPi*)(&juego.arkanoPi));
	PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
	piUnlock (STD_IO_BUFFER_KEY);
}
```

```
/**
* @brief método asignado a la inturrupción por la pulsación del boton asignado,
*     junto con la posible interrupción de un segundo botón para pausar el juego
*/
void boton_izquierdo(void) {
      // Pin event (key / button event) debouncing procedure
      if (millis () < debounceTime){
            debounceTime = millis () + DEBOUNCE_TIME ;
            return;
      }

      piLock (FLAGS_KEY);
      flags |= FLAG_RAQUETA_IZQUIERDA;
      piUnlock (FLAGS_KEY);

      // Wait for key to be released
      while (digitalRead (GPIO_BOTON_IZQ) == HIGH) {
            delay (1) ;
            if (digitalRead (GPIO_BOTON_DER) == HIGH) {
                  piLock (FLAGS_KEY);
                  flags |= (FLAG_RAQUETA_DERECHA || FLAG_RAQUETA_IZQUIERDA);
                  piUnlock (FLAGS_KEY);
            }
      }
      debounceTime = millis () + DEBOUNCE_TIME ;
}

/**
* @brief rutina donde se detiene el juego, sacando por pantalla la instrucción a seguir,
*     junto con la destrucción del temporizador del movimiento pelota
* @param fsm
*/
void PauseIt (fsm_t *fsm) {
      piLock (FLAGS_KEY);
      flags &= ~(FLAG_RAQUETA_DERECHA || FLAG_RAQUETA_IZQUIERDA);
      piUnlock (FLAGS_KEY);
```

```
	piLock (STD_IO_BUFFER_KEY);
	printf("Press the buttons again to continue: ... ");
	ActualizaPantalla((tipo_arkanoPi*)(&juego.arkanoPi));
	PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
	piUnlock (STD_IO_BUFFER_KEY);

	tmr_destroy(tmr_pelota);
}

/**
* @brief rutina donde se reanuda el juego, creando y empezando de nuevo el temporizador del
*     movimiento pelota
* @param fsm
*/
void PlayIt (fsm_t *fsm) {
	piLock (FLAGS_KEY);
	flags &= ~(FLAG_RAQUETA_DERECHA || FLAG_RAQUETA_IZQUIERDA);
	piUnlock (FLAGS_KEY);

	piLock (STD_IO_BUFFER_KEY);
	printf("Continue... ");
	ActualizaPantalla((tipo_arkanoPi*)(&juego.arkanoPi));
	PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
	piUnlock (STD_IO_BUFFER_KEY);

	tmr_pelota = tmr_new(pelota_auto);
	tmr_startms(tmr_pelota, v_pelota);
}

/**
* @brief método asignado a la inturrupción por la pulsación del boton asignado
*/
void boton_tecla(void) {
	// Pin event (key / button event) debouncing procedure
	if (millis () < debounceTime){
		debounceTime = millis () + DEBOUNCE_TIME ;
		return;
	}
```

```
	piLock (FLAGS_KEY);
	flags |= FLAG_TECLA;
	piUnlock (FLAGS_KEY);

	// Wait for key to be released
	while (digitalRead (GPIO_BOTON_TECLA) == HIGH) {
		delay (1) ;
	}
	debounceTime = millis () + DEBOUNCE_TIME ;
}

/**
* @brief método asignado a la inturrupción por la pulsación del boton asignado,
*	permitiendo así salir de la ejecución del programa
*/
void boton_salida(void) {
	exit(0);
}

/**
* @brief rutina donde se finaliza el juego y se muestra un mensaje por pantalla dependiendo
*	de si se ha ganado o se ha perdido
* @param fsm
*/
void FinalJuegoFSM(fsm_t* fsm) {
	piLock (FLAGS_KEY);
	flags &= ~FLAG_FINAL_JUEGO;
	piUnlock (FLAGS_KEY);

	piLock (STD_IO_BUFFER_KEY);
	if (nivel == 4) {// suponiendo 3 el numero niveles m□ximo
		printf("\n");
		printf("You win!");
		printf("\n");

	PintaMensajeInicialPantalla((tipo_pantalla*)(&juego.arkanoPi.pantalla),(tipo_pantalla*)(&mensaje_victoria));
		PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
```

```
	} else {
		printf("\n");
		printf("You lose in the %d level, %d bricks remaining!", nivel,
CalculaLadrillosRestantes((tipo_pantalla*)(&juego.arkanoPi.ladrillos.matriz)));
		printf("\n");

	PintaMensajeInicialPantalla((tipo_pantalla*)(&juego.arkanoPi.pantalla),(tipo_pantalla*)(&mensaje_derrota));
		PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
	}
	piUnlock (STD_IO_BUFFER_KEY);

	tmr_destroy(tmr_pelota);
}

/**
* @brief rutina donde se reinicia el juego y los elementos vuelven a sus estados iniciales
* @param fsm
*/
void ReseteaJuegoFSM(fsm_t* fsm) {
	piLock (FLAGS_KEY);
	flags &= ~FLAG_TECLA;
	piUnlock (FLAGS_KEY);

	ReseteaMatriz((tipo_pantalla*)(&juego.arkanoPi.pantalla.matriz));
	ReseteaLadrillos((tipo_pantalla*)(&juego.arkanoPi.ladrillos.matriz));
	ReseteaPelota((tipo_pelota*)(&juego.arkanoPi.pelota));
	ReseteaRaqueta((tipo_raqueta*)(&juego.arkanoPi.raqueta));

	tmr_pelota = tmr_new(pelota_auto);
	cnt_rebotes = 1;
	v_pelota = TIMEOUT_PELOTA;
	vidas = 3;
	nivel = 1;

	piLock (STD_IO_BUFFER_KEY);
	ActualizaPantalla((tipo_arkanoPi*)(&juego.arkanoPi));
	PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
	piUnlock (STD_IO_BUFFER_KEY);
}
```

```
/**
* @brief método asignado al temporizador que excita las columnas una a una y, en cada una, explora las flas de la matriz
*/
void excitaColumna(union sigval value){

	switch (columna) {
	case 0:
		digitalWrite(gpio_cols[0], 0);
		digitalWrite(gpio_cols[1], 0);
		digitalWrite(gpio_cols[2], 0);
		digitalWrite(gpio_cols[3], 0);
		break;
	case 1:
		digitalWrite(gpio_cols[0], 1);
		digitalWrite(gpio_cols[1], 0);
		digitalWrite(gpio_cols[2], 0);
		digitalWrite(gpio_cols[3], 0);
		break;

	case 2:
		digitalWrite(gpio_cols[0], 0);
		digitalWrite(gpio_cols[1], 1);
		digitalWrite(gpio_cols[2], 0);
		digitalWrite(gpio_cols[3], 0);
		break;
	case 3:
		digitalWrite(gpio_cols[0], 1);
		digitalWrite(gpio_cols[1], 1);
		digitalWrite(gpio_cols[2], 0);
		digitalWrite(gpio_cols[3], 0);
		break;
	case 4:
		digitalWrite(gpio_cols[0], 0);
		digitalWrite(gpio_cols[1], 0);
		digitalWrite(gpio_cols[2], 1);
		digitalWrite(gpio_cols[3], 0);
		break;
	case 5:
		digitalWrite(gpio_cols[0], 1);
		digitalWrite(gpio_cols[1], 0);
```

```
            digitalWrite(gpio_cols[2], 1);
            digitalWrite(gpio_cols[3], 0);
            break;
      case 6:
            digitalWrite(gpio_cols[0], 0);
            digitalWrite(gpio_cols[1], 1);
            digitalWrite(gpio_cols[2], 1);
            digitalWrite(gpio_cols[3], 0);
            break;
      case 7:
            digitalWrite(gpio_cols[0], 1);
            digitalWrite(gpio_cols[1], 1);
            digitalWrite(gpio_cols[2], 1);
            digitalWrite(gpio_cols[3], 0);
            break;

      case 8:
            digitalWrite(gpio_cols[0], 0);
            digitalWrite(gpio_cols[1], 0);
            digitalWrite(gpio_cols[2], 0);
            digitalWrite(gpio_cols[3], 1);
            break;
      case 9:
            digitalWrite(gpio_cols[0], 1);
            digitalWrite(gpio_cols[1], 0);
            digitalWrite(gpio_cols[2], 0);
            digitalWrite(gpio_cols[3], 1);
            break;
      }


digitalWrite(GPIO_FILA_0, !juego.arkanoPi.pantalla.matriz[columna][0]);
      digitalWrite(GPIO_FILA_1, !juego.arkanoPi.pantalla.matriz[columna][1]);
      digitalWrite(GPIO_FILA_2, !juego.arkanoPi.pantalla.matriz[columna][2]);
      digitalWrite(GPIO_FILA_3, !juego.arkanoPi.pantalla.matriz[columna][3]);
      digitalWrite(GPIO_FILA_4, !juego.arkanoPi.pantalla.matriz[columna][4]);
      digitalWrite(GPIO_FILA_5, !juego.arkanoPi.pantalla.matriz[columna][5]);
      digitalWrite(GPIO_FILA_6, !juego.arkanoPi.pantalla.matriz[columna][6]);

      columna++;
```

```
	if (columna == 10) {
		columna = 0;
	}
	tmr_startms(tmr, TIMEOUT);
}

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

/**
* @brief procedimiento de configuracion del sistema:
*     configurar el uso de posibles librerías
*     configurar las interrupciones externas asociadas a los pines GPIO
*     configurar las interrupciones periódicas y sus correspondientes temporizadores
*     crear, si fuese necesario, los threads adicionales que pueda requerir el sistema
* @return 1, si el resultado es satisfactorio; -1, en caso contrario
*/
int systemSetup (void) {

	piLock (STD_IO_BUFFER_KEY);

	// sets up the wiringPi library
	if (wiringPiSetupGpio () < 0) {
		printf ("Unable to setup wiringPi\n");
		piUnlock (STD_IO_BUFFER_KEY);
		return -1;
	}

	piUnlock (STD_IO_BUFFER_KEY);

	return 1;
}

/**
* @brief función donde se inicia la máquina de estados, además se incluye un mensaje por pantalla
* @param inicio_fsm
*/
```

```
void fsm_setup(fsm_t* inicio_fsm) {
	piLock (FLAGS_KEY);
	flags = 0;
	piUnlock (FLAGS_KEY);

	piLock (STD_IO_BUFFER_KEY);
	printf("\nINICIO DEL JUEGO\n");
	PintaMensajeInicialPantalla((tipo_pantalla*)(&juego.arkanoPi.pantalla),(tipo_pantalla*)(&mensaje_inicial));
	PintaPantallaPorTerminal((tipo_pantalla*)(&juego.arkanoPi.pantalla));
	piUnlock (STD_IO_BUFFER_KEY);
}

/**
* @brief función donde se ejecuta el flujo principal del programa, incluyendo la máquina de estados
*	con su inicialización, junto con la creación de los diferentes temporizadores y el establecimiento
*	de los diferentes puertos GPIO con sus distintas funciones
*/
int main (){
	// Maquina de estados: lista de transiciones
	// {EstadoOrigen,FunciónDeEntrada,EstadoDestino,FunciónDeSalida}
	fsm_trans_t fsm_tabla[] = {
			{ WAIT_START, compruebaTeclaPulsada,  WAIT_PUSH, InicializaJuegoFSM },
			{ WAIT_PUSH, compruebaPausePlay, WAIT_PAUSE,   PauseIt },
			{ WAIT_PUSH, compruebaTeclaRaquetaDerecha, WAIT_PUSH,   MueveRaquetaDerechaFSM },
			{ WAIT_PUSH, compruebaTeclaRaquetaIzquierda, WAIT_PUSH,   MueveRaquetaIzquierdaFSM },
			{ WAIT_PAUSE, compruebaPausePlay, WAIT_PUSH,   PlayIt },
			{ WAIT_PUSH, compruebaFinalJuego, WAIT_END,   FinalJuegoFSM },
			{ WAIT_END, compruebaTeclaPulsada, WAIT_START,   ReseteaJuegoFSM },
			{ -1, NULL, -1, NULL },
	};


	/*Inicialización del Hardware*/
	wiringPiSetupGpio();

	/* Asociación de entradas del Hardware */
	pinMode(GPIO_BOTON_DER,INPUT);
	pinMode(GPIO_BOTON_IZQ,INPUT);
	pullUpDnControl(GPIO_BOTON_IZQ, 0);
	pullUpDnControl(GPIO_BOTON_DER, 0);
```

```
wiringPiISR(GPIO_BOTON_DER, INT_EDGE_FALLING, boton_derecho);
wiringPiISR(GPIO_BOTON_IZQ, INT_EDGE_FALLING, boton_izquierdo);

pinMode(GPIO_BOTON_TECLA,INPUT);
pullUpDnControl(GPIO_BOTON_TECLA, 0);

wiringPiISR(GPIO_BOTON_TECLA, INT_EDGE_FALLING, boton_tecla);

pinMode(GPIO_BOTON_SALIDA,INPUT);
pullUpDnControl(GPIO_BOTON_SALIDA, 0);

wiringPiISR(GPIO_BOTON_SALIDA, INT_EDGE_FALLING, boton_salida);

/*Local variable*/
unsigned int next;

pinMode(GPIO_FILA_0,OUTPUT);
pinMode(GPIO_FILA_1,OUTPUT);
pinMode(GPIO_FILA_2,OUTPUT);
pinMode(GPIO_FILA_3,OUTPUT);
pinMode(GPIO_FILA_4,OUTPUT);
pinMode(GPIO_FILA_5,OUTPUT);
pinMode(GPIO_FILA_6,OUTPUT);

pinMode(GPIO_COL_0,OUTPUT);
pinMode(GPIO_COL_1,OUTPUT);
pinMode(GPIO_COL_2,OUTPUT);
pinMode(GPIO_COL_3,OUTPUT);

pinMode(GPIO_PWM,OUTPUT);
softPwmCreate(GPIO_PWM, 0, 100);

fsm_t* inicio_fsm = fsm_new (WAIT_START, fsm_tabla, NULL);

// Configuracion e inicializacion del sistema
systemSetup();
fsm_setup (inicio_fsm);
```

```
	/* Creación y activación del timer------- */
	tmr = tmr_new(excitaColumna);
	tmr_startms(tmr, TIMEOUT);
	tmr_pelota = tmr_new(pelota_auto);

	next = millis();
	while (1) {
		fsm_fire (inicio_fsm);
		next += CLK_MS;
		delay_until (next);
	}

	/*Destrucción de la máquina de estados*/
	fsm_destroy (inicio_fsm);

	/*Destrucción del timer*/
	tmr_destroy(tmr);
	tmr_destroy(tmr_pelota);
}


/** File name     : arkanoPi_1.h
 *  Description   : Archivo de cabecera del programa principal
 */

#ifndef _ARKANOPI_H_
#define _ARKANOPI_H_

/*Includes--------------------------------------*/
#include <time.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <unistd.h>
#include <sys/time.h>
#include <wiringPi.h>
#include <softPwm.h>
#include "arkanoPiLib.h"
```

```
#include "kbhit.h"      // para poder detectar teclas pulsadas sin bloqueo y leer las teclas pulsadas
#include "fsm.h"  // para poder crear y ejecutar la m□quina de estados
#include "tmr.h"  // para poder crear y ejecutar el temporizador

/*Defines----------------------------------------*/
#define CLK_MS 10       // PERIODO DE ACTUALIZACION DE LA MAQUINA ESTADOS
#define TIMEOUT 1 // tiempo para llamar al temporizador
#define TIMEOUT_PELOTA  1000
#define DEBOUNCE_TIME   200

// FLAGS DEL SISTEMA
#define FLAG_TECLA                  0x01
#define FLAG_RAQUETA_DERECHA        0x04
#define FLAG_RAQUETA_IZQUIERDA      0x08
#define FLAG_FINAL_JUEGO            0x10

// A 'key' which we can lock and unlock - values are 0 through 3
//    This is interpreted internally as a pthread_mutex by wiringPi
//    which is hiding some of that to make life simple.
#define     FLAGS_KEY   1
#define     STD_IO_BUFFER_KEY 2
//    Salidas del HW
#define GPIO_FILA_0     0
#define GPIO_FILA_1     1
#define GPIO_FILA_2     2
#define GPIO_FILA_3     3
#define GPIO_FILA_4     4
#define GPIO_FILA_5     7
#define GPIO_FILA_6     23
#define GPIO_COL_0      14
#define GPIO_COL_1      17
#define GPIO_COL_2      18
#define GPIO_COL_3      22
#define GPIO_PWM        24
//    Entradas del HW
#define GPIO_BOTON_DER        16
#define GPIO_BOTON_IZQ        15
#define GPIO_BOTON_TECLA      21
#define GPIO_BOTON_SALIDA     9
```

```
void delay_until (unsigned int next);
void fsm_setup(fsm_t* inicio_fsm);

//-------------------------------------------------------
// FUNCIONES DE ENTRADA O DE TRANSICION DE LA MAQUINA DE ESTADOS
//-------------------------------------------------------
int compruebaTeclaPulsada (fsm_t* this);
int compruebaTeclaRaquetaDerecha (fsm_t* this);
int compruebaTeclaRaquetaIzquierda (fsm_t* this);
int compruebaPausePlay (fsm_t* this);
int compruebaFinalJuego (fsm_t* this);

//-------------------------------------------------------
// FUNCIONES DE SALIDA O DE ACCION DE LA MAQUINA DE ESTADOS
//-------------------------------------------------------
void InicializaJuegoFSM(fsm_t* fsm);
void MovimientoPelotaFSM(fsm_t* fsm);
void MueveRaquetaDerechaFSM(fsm_t* fsm);
void MueveRaquetaIzquierdaFSM(fsm_t* fsm);
void PauseIt (fsm_t *fsm)
void PlayIt (fsm_t *fsm);
void FinalJuegoFSM(fsm_t* fsm);
void ReseteaJuegoFSM(fsm_t* fsm);

typedef enum {
      WAIT_START,
      WAIT_PUSH,
      WAIT_PAUSE,
      WAIT_END} tipo_estados_juego;

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

//-------------------------------------------------------
// FUNCIONES DE INICIALIZACION
//-------------------------------------------------------
int systemSetup (void);
#endif /* ARKANOPI_H_ */
```

```
/** File name     : arkanoPiLib.c
 *  Description   : Funcionalidades referidas a la matriz, los ladrillos, la raqueta y la pelota
 */
/*Includes----------------------------------------*/
#include "arkanoPiLib.h"

/*Private variables-------------------------------*/
int ladrillos_basico[MATRIZ_ANCHO][MATRIZ_ALTO] = {
            {0,1,0,0,0,0,0},
            {2,0,0,0,0,0,0},
            {0,1,0,0,0,0,0},
            {2,0,0,0,0,0,0},
            {0,1,0,0,0,0,0},
            {2,0,0,0,0,0,0},
            {0,1,0,0,0,0,0},
            {2,0,0,0,0,0,0},
            {0,1,0,0,0,0,0},
            {2,0,0,0,0,0,0},};
tipo_pantalla mensaje_inicial = {
            {{0,0,0,0,0,0,0},
            {0,0,0,0,0,0,0},
            {0,1,1,1,1,1,0},
            {0,1,1,1,1,1,0},
            {0,1,1,0,1,0,0},
            {0,1,1,0,1,0,0},
            {0,1,1,1,1,1,0},
            {0,1,1,1,1,1,0},
            {0,0,0,0,0,0,0},
            {0,0,0,0,0,0,0},}};
tipo_pantalla mensaje_victoria = {
            {{0,0,0,0,0,0,0},
            {0,0,0,0,0,0,0},
            {0,1,1,0,1,0,0},
            {0,1,1,0,0,1,0},
            {0,0,0,0,0,1,0},
            {0,0,0,0,0,1,0},
            {0,1,1,0,0,1,0},
            {0,1,1,0,1,0,0},
            {0,0,0,0,0,0,0},
            {0,0,0,0,0,0,0},}};
```

```
tipo_pantalla mensaje_derrota = {
                  {{0,0,0,0,0,0,0},
                  {0,0,0,0,0,0,0},
                  {0,1,1,0,0,0,1},
                  {0,1,1,0,0,1,0},
                  {0,0,0,0,0,1,0},
                  {0,0,0,0,0,1,0},
                  {0,1,1,0,0,1,0},
                  {0,1,1,0,0,0,1},
                  {0,0,0,0,0,0,0},
                  {0,0,0,0,0,0,0},}};

tipo_pantalla ladrillos_nv2 = {
                  {{3,2,0,0,0,0,0},
                  {3,0,0,0,0,0,0},
                  {3,2,0,0,0,0,0},
                  {3,0,0,0,0,0,0},
                  {3,2,0,0,0,0,0},
                  {3,0,0,0,0,0,0},
                  {3,2,0,0,0,0,0},
                  {3,0,0,0,0,0,0},
                  {3,2,0,0,0,0,0},
                  {3,0,0,0,0,0,0},}};


tipo_pantalla ladrillos_nv3 = {
                  {{4,3,0,0,0,0,0},
                  {4,0,2,0,0,0,0},
                  {4,3,0,0,0,0,0},
                  {4,0,2,0,0,0,0},
                  {4,3,0,0,0,0,0},
                  {4,0,2,0,0,0,0},
                  {4,3,0,0,0,0,0},
                  {4,0,2,0,0,0,0},
                  {4,3,0,0,0,0,0},
                  {4,0,2,0,0,0,0},}};
```

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

/**
 * @brief   método que pone la matriz entera a cero
 * @param   p_pantalla
 */
void ReseteaMatriz(tipo_pantalla *p_pantalla) {
      /* Local variables */
      int i, j = 0;

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

/**
 * @brief   método que restablece los ladrillos a su estado inicial
 * @param   p_ladrillos
 */
void ReseteaLadrillos(tipo_pantalla *p_ladrillos) {
      /* Local variables */
      int i, j = 0;

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

/**
 * @brief   método que restablece la pelota a su estado inicial
 * @param   p_pelota
 */
```

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

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

/**
 * @brief   método que restablece la raqueta a su estado inicial
 * @param   p_raqueta
 */
void ReseteaRaqueta(tipo_raqueta *p_raqueta) {
	// Raqueta inicialmente en el centro de la pantalla
	p_raqueta->x = MATRIZ_ANCHO/2 - p_raqueta->ancho/2;
	p_raqueta->y = MATRIZ_ALTO - 1;
	p_raqueta->ancho = RAQUETA_ANCHO;
	p_raqueta->alto = RAQUETA_ALTO;
}

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

/**
 * @brief   método encargado de aprovechar el display para presentar un mensaje de bienvenida al usuario
 * @param   p_pantalla
 * @param   p_pantalla_inicial
 */
void PintaMensajeInicialPantalla (tipo_pantalla *p_pantalla, tipo_pantalla *p_pantalla_inicial) {
	/* Local variables */
	int i, j = 0;
	for(i=0;i<MATRIZ_ANCHO;i++) {
		for(j=0;j<MATRIZ_ALTO;j++) {
			p_pantalla->matriz[i][j] = p_pantalla_inicial->matriz[i][j];
		}
	}
}
```

```
/**
 * @brief   método encargado de mostrar el contenido o la ocupación de la matriz de leds en la ventana de terminal o
consola
 * @param   p_pantalla
 */
void PintaPantallaPorTerminal  (tipo_pantalla *p_pantalla) {
      /* Local variables */
      int i, j = 0;

      printf("\n");
      for(i=0;i<MATRIZ_ALTO;i++) {
            for(j=0;j<MATRIZ_ANCHO;j++) {
                  printf("%d ",p_pantalla->matriz[j][i]);
            }
            printf("\n");
      }
      printf("\n");
}

/**
 * @brief   método encargado de pintar los ladrillos en sus correspondientes posiciones dentro del área de juego
 * @param   p_ladrillos
 * @param   p_pantalla
 */
void PintaLadrillos(tipo_pantalla *p_ladrillos, tipo_pantalla *p_pantalla) {
      /* Local variables */
      int i, j = 0;

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

/**
 * @brief   método encargado de pintar la raqueta en su posición correspondiente dentro del área de juego
 * @param   p_raqueta
 * @param   p_pantalla
 */
```

```
void PintaRaqueta(tipo_raqueta *p_raqueta, tipo_pantalla *p_pantalla) {
	/* Local variables */
	int i, j = 0;

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

/**
 * @brief	método encargado de pintar la pelota en su posición correspondiente dentro del área de juego
 * @param	p_pelota
 * @param	p_pantalla
 */
void PintaPelota(tipo_pelota *p_pelota, tipo_pantalla *p_pantalla) {
	if( (p_pelota->x >= 0) && (p_pelota->x < MATRIZ_ANCHO) ) {
		if( (p_pelota->y >= 0) && (p_pelota->y < MATRIZ_ALTO) ) {
			p_pantalla->matriz[p_pelota->x][p_pelota->y] = 1;
		}
		else {
			printf("\n\nPROBLEMAS!!!! posicion y=%d de la pelota INVALIDA!!!\n\n", p_pelota->y);
			fflush(stdout);
		}
	}
	else {
		printf("\n\nPROBLEMAS!!!! posicion x=%d de la pelota INVALIDA!!!\n\n", p_pelota->x);
		fflush(stdout);
	}
}

/**
 * @brief	método cuya ejecucion estara ligada a cualquiera de los movimientos de la raqueta o de la pelota y que
 *		será el encargado de actualizar convenientemente la estructura de datos
 *		en memoria que representa el área de juego y su correspondiente estado
 * @param	p_arkanoPi
 */
```

```
void ActualizaPantalla(tipo_arkanoPi* p_arkanoPi) {
    // Borro toda la pantalla
      ReseteaMatriz((tipo_pantalla*)(&(p_arkanoPi->pantalla)));

      // Completo la pantalla con los elementos correspondientes
      PintaLadrillos((tipo_pantalla*)(&(p_arkanoPi->ladrillos)), (tipo_pantalla*)(&(p_arkanoPi->pantalla)));
      PintaRaqueta((tipo_raqueta*)(&(p_arkanoPi->raqueta)), (tipo_pantalla*)(&(p_arkanoPi->pantalla)));
      PintaPelota((tipo_pelota*)(&(p_arkanoPi->pelota)), ((tipo_pantalla*)(&(p_arkanoPi->pantalla))));
}

/**
 * @brief   método encargado de la inicialización de toda variable o estructura de datos especificamente ligada al
 *              desarrollo del juego y su visualización
 * @param   p_arkanoPi
 */
void InicializaArkanoPi(tipo_arkanoPi *p_arkanoPi) {
      // Borro toda la pantalla
      ReseteaMatriz((tipo_pantalla*)(&(p_arkanoPi->pantalla)));

      // Completo la pantalla con los elementos correspondientes en sus posiciones originales
      ReseteaLadrillos((tipo_pantalla*)(&(p_arkanoPi->ladrillos)));
      ReseteaPelota((tipo_pelota*)(&(p_arkanoPi->pelota)));
      ReseteaRaqueta((tipo_raqueta*)(&(p_arkanoPi->raqueta)));

      // Se procede a mostrar por pantalla
      PintaPantallaPorTerminal((tipo_pantalla*)(&(p_arkanoPi->pantalla)));
}

/**
 * @brief   método encargado de evaluar el estado de ocupación del área de juego por los ladrillos y devolver
 *              el número de estos
 * @param   p_ladrillos
 * @return  número de ladrillos restantes
 */
int CalculaLadrillosRestantes(tipo_pantalla *p_ladrillos) {
      /* Local variables */
      int num_ladrillos_restantes = 0;
      int i, j = 0;
```

```
	for(i=0;i<LADRILLOS_ANCHO;i++) {
		for(j=0;j<LADRILLOS_ALTO;j++) {
			if (p_ladrillos->matriz[i][j] >= 1){
				num_ladrillos_restantes++;
			}
		}
	}
	return num_ladrillos_restantes;
}

/** File name     : arkanoPiLib.h
 *  Description   : Archivo de cabecera que recoge las funcionalidades referidas
 *                  a la matriz, los ladrillos, la raqueta y la pelota
 */

#ifndef _ARKANOPILIB_H_
#define _ARKANOPILIB_H_

/*Includes----------------------------------------*/
#include <stdio.h>
#include <wiringPi.h>

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

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

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

typedef struct {
      // Matriz de ocupación de las distintas posiciones que conforman el display
      int matriz[MATRIZ_ANCHO][MATRIZ_ALTO];
} tipo_pantalla;

typedef struct {
  tipo_pantalla ladrillos;
  tipo_pantalla pantalla;
  tipo_raqueta raqueta;
  tipo_pelota pelota;
} tipo_arkanoPi;

extern tipo_pantalla pantalla_inicial;

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

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

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

#endif /* _ARKANOPILIB_H_ */
```