

---

# **Memoria de las mejoras realizadas**

## **Curso 2020/2021**

---

### ***Mejoras De ArkanoPi***

Autores (orden alfabético):      Alfaro Fernandez Andres  
Barajas Carracedo Alejandro

Código de la pareja:              JT-01

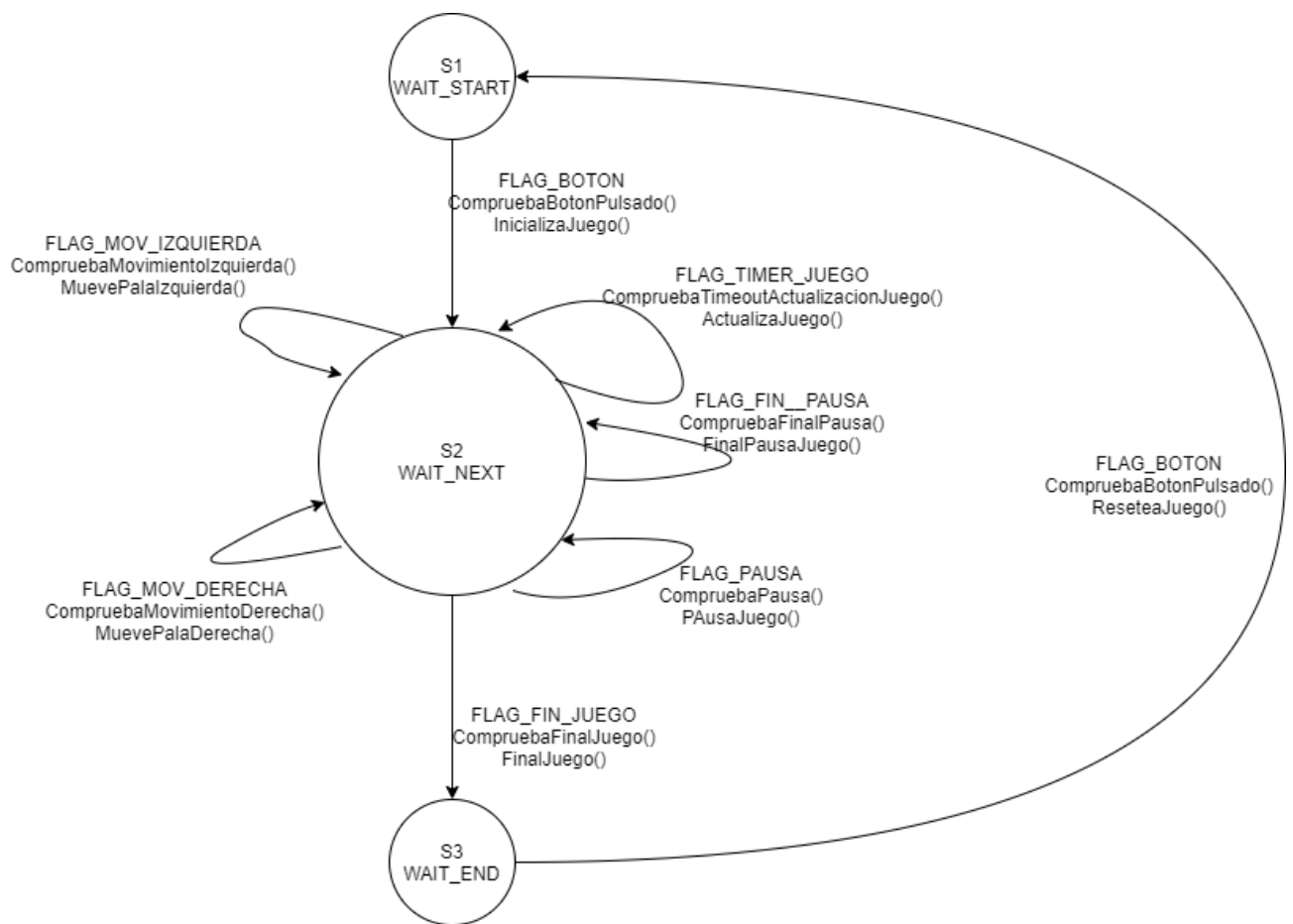
---

# ÍNDICE GENERAL

<b>1</b>	<b>DIAGRAMA DE SUBSISTEMAS AMPLIADO.....</b>	<b>2</b>
<b>2</b>	<b>MEJORA 1: FUNCIONALIDAD SCORES.....</b>	<b>3</b>
2.1	OBJETIVOS DE LA MEJORA .....	3
2.2	DESCRIPCIÓN DEL SUBSISTEMA SOFTWARE .....	3
2.2.1	<i>Flujo de ejecución del programa principal.....</i>	<i>4</i>
<b>3</b>	<b>MEJORA 2: NIVELES E INCREMENTO DE DIFICULTAD.....</b>	<b>5</b>
3.1	OBJETIVOS DE LA MEJORA .....	5
3.2	DESCRIPCIÓN DEL SUBSISTEMA SOFTWARE .....	5
3.2.1	<i>Flujo de ejecución del programa principal.....</i>	<i>6</i>
3.2.2	<i>Procesos de las interrupciones .....</i>	<i>6</i>
<b>4</b>	<b>MEJORA 3: FUNCIONALIDAD DE VIDAS.....</b>	<b>7</b>
4.1	OBJETIVOS DE LA MEJORA .....	7
4.2	DESCRIPCIÓN DEL SUBSISTEMA SOFTWARE .....	7
4.2.1	<i>Flujo de ejecución del programa principal.....</i>	<i>7</i>
4.2.2	<i>Procesos de las interrupciones .....</i>	<i>7</i>
<b>5</b>	<b>MEJORA 4: FUNCIONALIDAD DE PAUSA.....</b>	<b>8</b>
5.1	OBJETIVOS DE LA MEJORA .....	8
5.2	DESCRIPCIÓN DEL SUBSISTEMA SOFTWARE .....	8
5.2.1	<i>Flujo de ejecución del programa principal.....</i>	<i>8</i>
5.2.2	<i>Procesos de las interrupciones .....</i>	<i>8</i>
<b>6</b>	<b>MEJORA 5: MEJORA DE LA INTERFAZ DEL JUEGO.....</b>	<b>9</b>
6.1	OBJETIVOS DE LA MEJORA .....	9
6.2	DESCRIPCIÓN DEL SUBSISTEMA SOFTWARE .....	9
<b>7</b>	<b>MEJORA 6: AUSENCIA DE REBOTRES VERTICALES.....</b>	<b>10</b>
7.1	OBJETIVOS DE LA MEJORA .....	10
7.2	DESCRIPCIÓN DEL SUBSISTEMA SOFTWARE .....	10
7.2.1	<i>Flujo de ejecución del programa principal.....</i>	<i>10</i>
7.2.2	<i>Procesos de las interrupciones .....</i>	<i>10</i>
<b>8</b>	<b>PRINCIPALES PROBLEMAS ENCONTRADOS.....</b>	<b>11</b>
<b>9</b>	<b>MANUAL DE USUARIO .....</b>	<b>12</b>
<b>10</b>	<b>BIBLIOGRAFÍA.....</b>	<b>13</b>
<b>11</b>	<b>ANEXO I: CÓDIGO DEL PROGRAMA DEL PROYECTO FINAL.....</b>	<b>14</b>

## 1 Diagrama de subsistemas ampliado

La máquina de estados implementada en las mejoras es la misma de la versión 4 de arkanoPi. El único cambio implementado es la funcionalidad de pausa, que será explicada más adelante en la séptima mejora.



## **2 Mejora 1: Funcionalidad scores**

### **2.1 Objetivos de la mejora**

El objetivo de esta mejora es diseñar un contador que se vaya incrementando cada vez que la pelota rompa un ladrillo.

### **2.2 Descripción del subsistema Software**

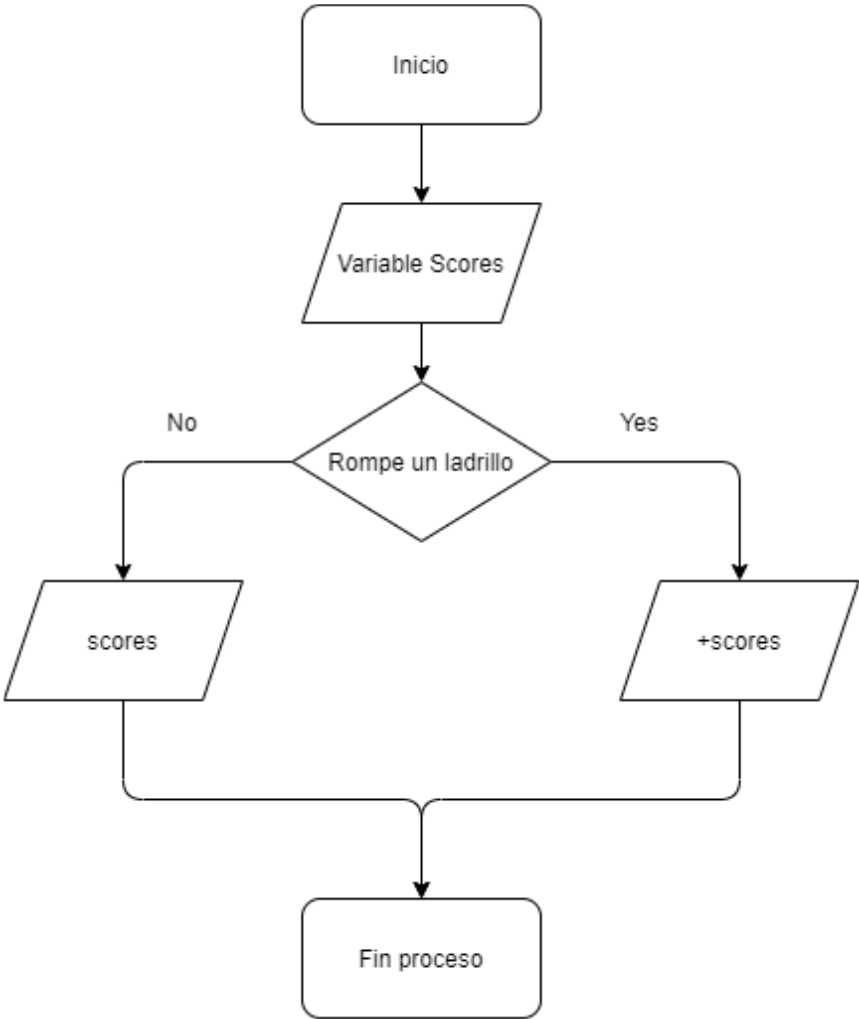
Para el desarrollo de dicha mejora ha sido necesario modificar el archivo `systemLib.h` para introducir la nueva variable entera `score` y el archivo `arkanoPiLib.c` para ir incrementando el contador cada vez que la pelota rompe un ladrillo.

Respecto a la descripción más detallada del software. Para la creación de la variable `scores`, debemos introducir en el archivo `systemLib.h` la siguiente sentencia (`extern int scores;`), de esta forma crearemos una variable global donde podremos actualizar la puntuación del jugador en cada momento. También tenemos una variable `betsscore` que almacena una puntuación máxima de forma que, si el jugador pierde una vida, se queda guardada la puntuación para la siguiente vida.

Para poder actualizar el valor de la variable `scores` cada vez que se rompa un ladrillo debemos dirigirnos al archivo `arkanoPiLib.c`, en la función `CompruebaReboteLadrillo`, cuando comprobamos la condición donde la pelota ha entrado en la zona de ladrillos, además de retornar 1, añadimos que la variable `scores` se incremente cada vez que se cumpla dicha condición.

También será necesario resetear el valor de `scores` cada vez que el juego se reinicie, para ello, en el archivo `arkanoPiLib.c` en las funciones `ReseteaJuego`, y `InicializaJuego` debemos añadir el valor 0 a la variable `scores`, de esta forma cada vez que iniciemos o reseteemos el juego el valor de `scores` se reiniciará.

2.2.1 Flujo de ejecución del programa principal



...

## **3 Mejora 2: NIVELES E INCREMENTO DE DIFICULTAD**

### **3.1 Objetivos de la mejora**

El objetivo de esta mejora es introducir un sistema de dificultad por niveles en el juego mediante el cual, cuantos más puntos consiga el jugador, más dificultad tendrá el juego, por lo tanto, más rápido se moverá la pelota.

### **3.2 Descripción del subsistema Software**

Para el desarrollo de esta mejora ha sido necesario modificar el archivo `arkanoPiLib.c` mejorando la función `ActualizaJuego`, de forma que sea posible aumentar la velocidad de actualización de la pelota según se aumenta de nivel. Esto lo hacemos cambiando el timeout del timer actualiza juego, utilizando una variable externa `speed` que va disminuyendo cada vez que el jugador llega a una puntuación determinada.

El sistema de niveles está configurado de esta manera:

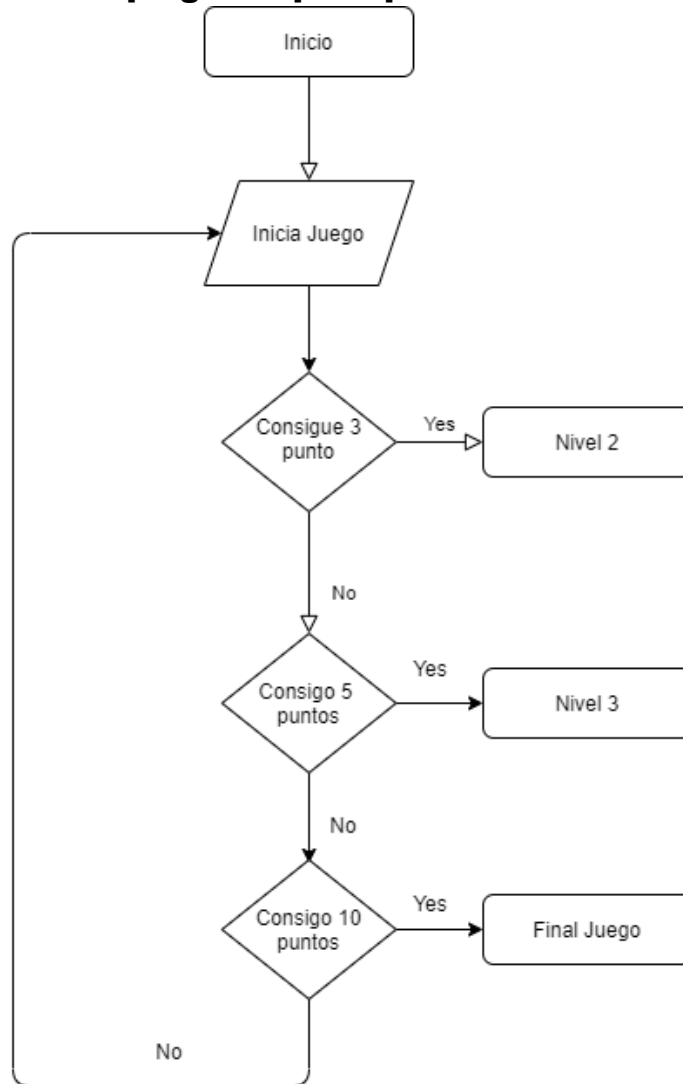
- Nivel 1: al principio del juego.
- Nivel 2: cuando el jugador llega a los 3 puntos.
- Nivel 3: cuando el jugador llega a 5 puntos

Una vez que el usuario llega a una puntuación de 10 puntos, el juego se para, ya que se considera que el jugador ha ganado la partida. Si por el contrario, el jugador no llega a 10 puntos, se considera que ha perdido la partida.

A continuación, podemos ver la funcionalidad descrita anteriormente con el siguiente enlace:

<https://youtu.be/9Gj0c4keVt4>

### 3.2.1 Flujo de ejecución del programa principal



### 3.2.2 Procesos de las interrupciones

Cuando la variable score es igual a 3 puntos, se interrumpe el timer para aumentar la velocidad de la pelota (que la velocidad de la pelota es el timeout del timer). De igual forma, cuando la variable score es igual a 5 tiene el mismo proceso, interrumpe el timer y aumenta la velocidad de la pelota.

Por último, si la variable score es igual a 10 puntos, se interrumpe el timer de inicialización de juego, y llama al flag de fin de juego.

## 4 Mejora 3: FUNCIONALIDAD DE VIDAS

### 4.1 Objetivos de la mejora

El objetivo de esta mejora es darle al jugador 3 vidas para poder ganar en el juego a pesar de perder cuando la pelota no rebota en la pala. Y en el caso que el número de vidas sea menor que cero, el jugador perderá la partida.

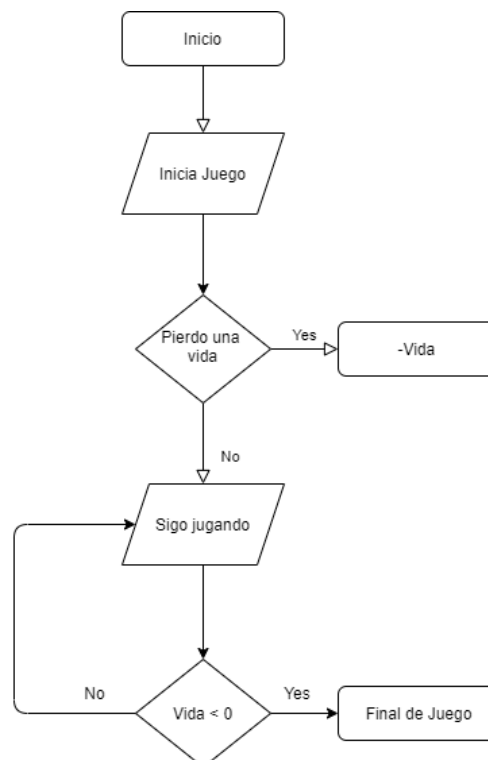
### 4.2 Descripción del subsistema Software

Para la implementación de esta funcionalidad se ha declarado una variable externa llamada `lives` en `systemLib.h` que está iniciada a tres. A su vez tenemos un contador en la función `ActualizaJuego` de `arkanoPiLib.c`, de esta forma, cuando la función `CompruebaFallo` devuelve uno, se decrementará uno la variable `lives`. Además, en esta función también está implementada la condición de que si el contador es menor que cero el juego termina.

En el siguiente enlace disponemos de un video donde se muestra la funcionalidad comentada:

<https://youtu.be/mQEpm39EvZE>

#### 4.2.1 Flujo de ejecución del programa principal



#### 4.2.2 Procesos de las interrupciones

Cuando la variable vidas es menor que 0, se interrumpe el timer que inicializa el juego, y se realiza la llamada al flag de fin de juego.



## 5 Mejora 4: FUNCIONALIDAD DE PAUSA

### 5.1 Objetivos de la mejora

El objetivo de esta mejora es establecer un botón con el cual podamos pausar el juego y a su vez, si lo volvemos a pulsar, volver a iniciar el juego.

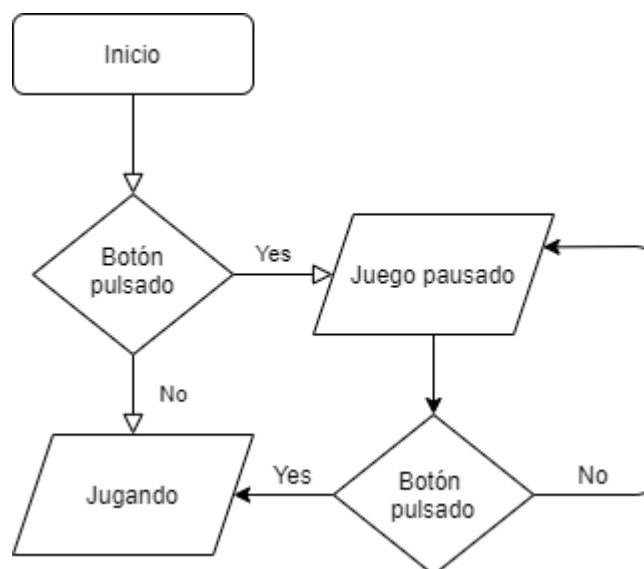
### 5.2 Descripción del subsistema Software

Para el desarrollo de esta funcionalidad, se han añadido dos flags (FLAG\_PAUSA y FLAG\_FIN\_PAUSA), además, se ha actualizado la máquina de estados de arkanopi.c dentro de la función main, para su correcto funcionamiento, por otro lado se han implementado las funciones que llama dicha máquina cuando se pulsa el botón de pausa (al cual se le ha asignado el botón 'F' en teclado\_TL04.c), ya sea para pausar o para continuar con el juego.

En el siguiente enlace, tenemos un video de la funcionalidad comentada:

<https://youtu.be/JKPmNZDU0d4>

#### 5.2.1 Flujo de ejecución del programa principal



#### 5.2.2 Procesos de las interrupciones

Cuando se pulsa la tecla F se interrumpe el timeout y se le cambia el valor a cero, para pausar el movimiento de la pelota. Cuando se vuelve a pulsar la tecla, se interrumpe el timeout y se cambia a su valor antes de ser pulsada la tecla F.

## 6 Mejora 5: MEJORA DE LA INTERFAZ DEL JUEGO

### 6.1 Objetivos de la mejora

El objetivo de esta mejora es implementar todos los marcadores anteriormente descritos, además de realizar sutiles cambios en la disposición del juego, para darle más dinamismo.

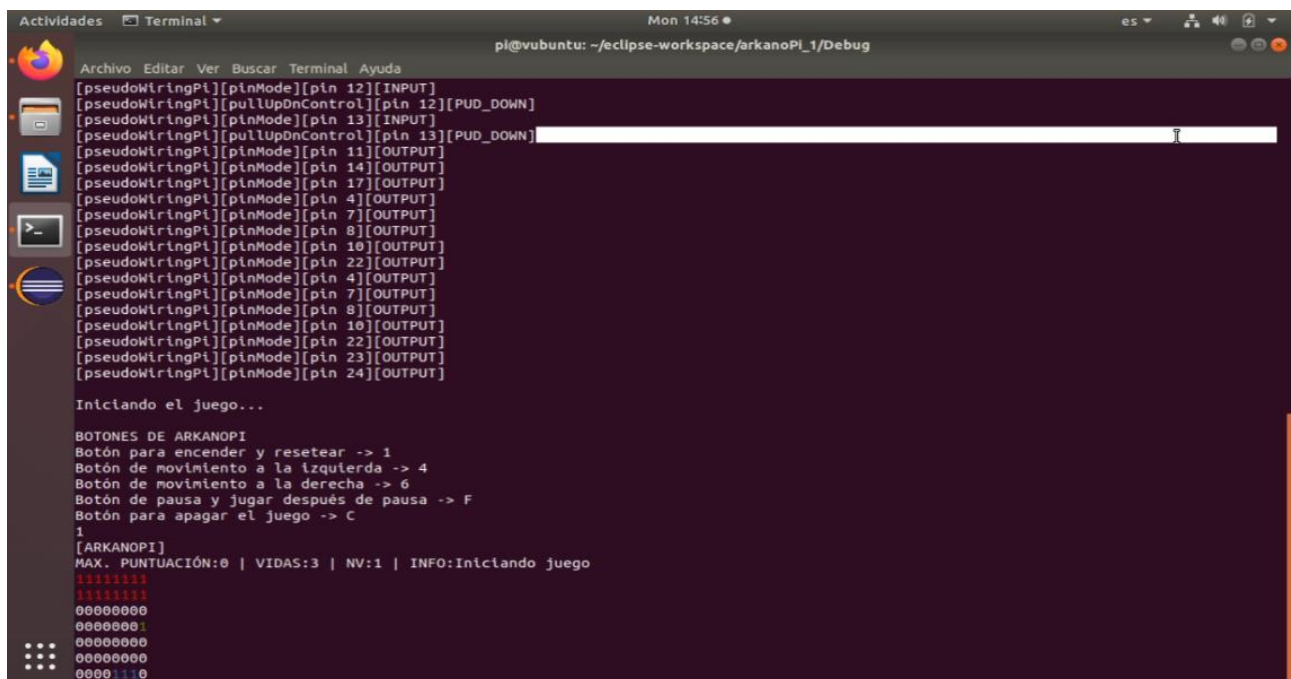
### 6.2 Descripción del subsistema Software

En primer lugar, se han añadido tanto el marcador como el contador de vidas del jugador en la parte superior de la pantalla. Por otro lado, se ha cambiado el color de la pelota para poder distinguirla de mejor manera frente al resto de elementos del juego. Por último, hemos añadido la información de la partida por pantalla, es decir, avisos de lo que está sucediendo en el juego en ese momento.

Además, hemos cambiado el color de la pelota, en el archivo `pseudowiringPi` en la función `pseudoMatrizColor`, cambiando las filas intermedias que tenían el número 0 (asignación de color blanco) por el número 32 que representa el color verde, de esta forma cada vez que la pelota pase por una de esas filas su color cambiará a verde.

Cada uno de los parámetros que se va a mostrar por la pantalla se realiza declarando una variable externa que permita en todo momento la actualización de dichos valores.

En la siguiente imagen podemos observar todos los elementos anteriormente comentados:



```
Archivo Editar Ver Buscar Terminal Ayuda
Mon 14:56
pi@vubuntu: ~/eclipse-workspace/arkanoPi_1/Debug
[pseudoWiringPi][pinMode][pin 12][INPUT]
[pseudoWiringPi][pullUpDnControl][pin 12][PUD_DOWN]
[pseudoWiringPi][pinMode][pin 13][INPUT]
[pseudoWiringPi][pullUpDnControl][pin 13][PUD_DOWN]
[pseudoWiringPi][pinMode][pin 11][OUTPUT]
[pseudoWiringPi][pinMode][pin 14][OUTPUT]
[pseudoWiringPi][pinMode][pin 17][OUTPUT]
[pseudoWiringPi][pinMode][pin 4][OUTPUT]
[pseudoWiringPi][pinMode][pin 7][OUTPUT]
[pseudoWiringPi][pinMode][pin 8][OUTPUT]
[pseudoWiringPi][pinMode][pin 10][OUTPUT]
[pseudoWiringPi][pinMode][pin 22][OUTPUT]
[pseudoWiringPi][pinMode][pin 4][OUTPUT]
[pseudoWiringPi][pinMode][pin 7][OUTPUT]
[pseudoWiringPi][pinMode][pin 8][OUTPUT]
[pseudoWiringPi][pinMode][pin 10][OUTPUT]
[pseudoWiringPi][pinMode][pin 22][OUTPUT]
[pseudoWiringPi][pinMode][pin 23][OUTPUT]
[pseudoWiringPi][pinMode][pin 24][OUTPUT]

Iniciando el juego...

BOTONES DE ARKANOPI
Botón para encender y resetear -> 1
Botón de movimiento a la izquierda -> 4
Botón de movimiento a la derecha -> 6
Botón de pausa y jugar después de pausa -> F
Botón para apagar el juego -> c
1
[ARKANOPI]
MAX. PUNTUACIÓN:0 | VIDAS:3 | NV:1 | INFO:Iniciando juego
11111111
11111111
00000000
00000000
00000000
00000000
00001110
```

## 7 Mejora 6: AUSENCIA DE REBOTRES VERTICALES

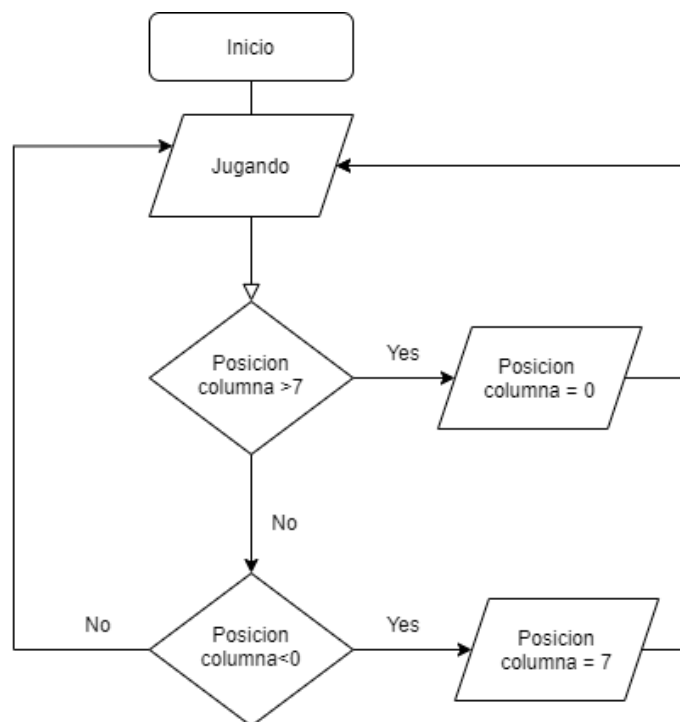
### 7.1 Objetivos de la mejora

El objetivo de esta mejora es aumentar la dificultad en el nivel 3 de nuestro juego, haciendo que la pelota vaya de la pared derecha a la izquierda o viceversa, todo esto sin que la pelota cambie su trayectoria.

### 7.2 Descripción del subsistema Software

Para la realización de esta mejora hemos modificado el archivo `arkanoPiLib.c`, cambiando algunas condiciones de las funciones predefinidas `CompruebaLadrillo` y `PintaPelota`, ya que en algunas situaciones daban un “error grave”. Después hemos modificado la función `ActualizarJuego`, para que cuando la función `CompruebaReboteVerticales` devuelva uno si la pelota está en la columna 7, al actualizar la pantalla, en el siguiente frame, la pelota aparecerá en la columna 0 o viceversa.

#### 7.2.1 Flujo de ejecución del programa principal



#### 7.2.2 Procesos de las interrupciones

Cuando se pulsa la tecla F se activa el `FLAG_PAUSA`. Al comprobar la activación de `FLAG_PAUSA`, y si esta devuelve 1, la función de salida pondrá el timeout del timer `ActualizarJuego` a cero para que el juego se quede congelado. Cuando se vuelve a pulsar la tecla, se interrumpe el timeout y se le cambia a su valor antes de ser pulsada la tecla F.

## 8 Principales problemas encontrados

Respecto a los principales problemas encontrados, no hemos tenido muchos problemas en las mejoras, pero los más complicados de resolver han sido estos:

- Al modificar la `pseudoWiringPi.c` para la mejora de la interfaz, hemos necesitado muchos ensayos y errores, ya que al colocar `printf` para imprimir el marcador, la matriz del juego no salía como debía, pero aun así conseguimos entender su funcionalidad y pudimos poner el marcador.
- Con el botón de pausa también tuvimos algunos problemas, ya que al entrar el juego en pausa no sabíamos bien como salir de la pausa, porque no se activaba el flag de fin de pausa. Inicialmente se nos ocurrió declarar una variable externa llamada `pausa` para que se activase a uno cuando el sistema estaba en pausa y se pusiera a cero cuando se estaba jugando. Finalmente, no utilizamos dicha variable, debido a que nos dimos cuenta que si el `timeout` del `timer` actualiza juego estaba a cero, esto significaba que el sistema estaría en pausa. Por ello utilizamos esta condición para que al pulsar la tecla F (botón de pausa) el sistema sepa que flag activar en todo momento.
- Por último, al quitar los rebotes verticales cuando llegaba la pelota a rebotar con la pared de la derecha, el juego imprimía un mensaje de “error grave”, puesto que según su programación la pelota estaba fuera de la matriz. Esto se solucionó gracias a cambiar las condiciones donde las funciones `CompruebaLadrillo` y `PintaPelota` cuando imprimían el error.

El resto de problemas que hemos ido obteniendo, los hemos solventado de forma rápida.

## 9 Manual de usuario

Para poder jugar al juego en primer lugar necesitamos describir la función de nuestras teclas asignadas.

Teclas asignadas:

- La tecla 1, se encarga de iniciar el juego
- La tecla F tiene la función de pausar, y reanudar el juego
- La tecla C se encarga de terminar el juego
- La tecla 4 realiza el movimiento hacia la izquierda de la barra
- La tecla 6 realiza el movimiento hacia la derecha de la barra

El jugador dispone de un marcador con el cual, a medida que vaya consiguiendo puntos se incrementará la dificultad del juego, hasta alcanzar los 10 puntos que sería cuando finaliza el juego.

Se implementa un sistema de vidas, por lo tanto, el jugador dispone de 3 oportunidades para acabar la partida, una vez gastadas estas 3 oportunidades, el juego finalizará.

A continuación, podemos observar una captura de los controles descritos en el propio juego:

```

pi@vubuntu: ~/eclipse-workspace/arkanoPi_1/Debug$ ./arkanoPi_1
[pseudoWiringPi][pinMode][pin 0][OUTPUT]
[pseudoWiringPi][pinMode][pin 1][OUTPUT]
[pseudoWiringPi][pinMode][pin 2][OUTPUT]
[pseudoWiringPi][pinMode][pin 3][OUTPUT]
[pseudoWiringPi][pinMode][pin 5][INPUT]
[pseudoWiringPi][pullUpDnControl][pin 5][PUD_DOWN]
[pseudoWiringPi][pinMode][pin 6][INPUT]
[pseudoWiringPi][pullUpDnControl][pin 6][PUD_DOWN]
[pseudoWiringPi][pinMode][pin 12][INPUT]
[pseudoWiringPi][pullUpDnControl][pin 12][PUD_DOWN]
[pseudoWiringPi][pinMode][pin 13][INPUT]
[pseudoWiringPi][pullUpDnControl][pin 13][PUD_DOWN]
[pseudoWiringPi][pinMode][pin 11][OUTPUT]
[pseudoWiringPi][pinMode][pin 14][OUTPUT]
[pseudoWiringPi][pinMode][pin 17][OUTPUT]
[pseudoWiringPi][pinMode][pin 4][OUTPUT]
[pseudoWiringPi][pinMode][pin 7][OUTPUT]
[pseudoWiringPi][pinMode][pin 8][OUTPUT]
[pseudoWiringPi][pinMode][pin 10][OUTPUT]
[pseudoWiringPi][pinMode][pin 22][OUTPUT]
[pseudoWiringPi][pinMode][pin 4][OUTPUT]
[pseudoWiringPi][pinMode][pin 7][OUTPUT]
[pseudoWiringPi][pinMode][pin 8][OUTPUT]
[pseudoWiringPi][pinMode][pin 10][OUTPUT]
[pseudoWiringPi][pinMode][pin 22][OUTPUT]
[pseudoWiringPi][pinMode][pin 23][OUTPUT]
[pseudoWiringPi][pinMode][pin 24][OUTPUT]

Iniciando el juego...

BOTONES DE ARKANOPI
Botón para encender y resetear -> 1
Botón de movimiento a la izquierda -> 4
Botón de movimiento a la derecha -> 6
Botón de pausa y jugar después de pausa -> F
Botón para apagar el juego -> C
    
```

## **10 Bibliografía**

En referencia a la bibliografía utilizada para el desarrollo de la práctica, hemos utilizado los documentos adjuntos de Moodle de la asignatura de SDG2:

1. Tutorial1: Introducción al entorno de desarrollo en C para Raspberry Pi
2. Tutorial2: Introducción a las máquinas de estados en C para Raspberry Pi
3. Tutorial3: Iniciación al manejo de las Entradas/Salidas del BCM 2835
4. Tutorial4: Manejo de temporizadores, interrupciones y procesos con la Raspberry Pi
5. Tutorial5: Manejo de periféricos mediante SPI con la Raspberry Pi
6. Display LCD
7. Los diversos videotutoriales sobre programación en C

## 11 ANEXO I: Código del programa del proyecto final

### Ejemplo de código en C:

#### arkanoPi.c

```
#include
"arkanoPi.h"

int flags = 0;
int speed = 0;
int bestscore = 0;
int lifes = 3;//El usuario empieza con tres vidas
int nivel = 1;//El usuario empieza en el nivel 1
char *info="";//Se para la informacion del juego en cada momento

TipoSistema sistema;

// Declaracion del objeto teclado
TipoTeclado teclado = { .columnas = {
// A completar por el alumno...
// ...
        GPIO_KEYBOARD_COL_1,
        GPIO_KEYBOARD_COL_2,
        GPIO_KEYBOARD_COL_3,
        GPIO_KEYBOARD_COL_4 },
        .filas = {
// A completar por el alumno...
// ...
```

```
        GPIO_KEYBOARD_ROW_1,
        GPIO_KEYBOARD_ROW_2,
        GPIO_KEYBOARD_ROW_3,
        GPIO_KEYBOARD_ROW_4 },
    .rutinas_ISR = {
        // A completar por el alumno...
        // ...
        teclado_fila_1_isr,
        teclado_fila_2_isr,
        teclado_fila_3_isr,
        teclado_fila_4_isr },

// A completar por el alumno...
// ...
};

// Declaracion del objeto display
TipoLedDisplay led_display = {
    .pines_control_columnas = {
        // A completar por el alumno...
        // ...
        GPIO_LED_DISPLAY_COL_1,
        GPIO_LED_DISPLAY_COL_2,
        GPIO_LED_DISPLAY_COL_3,
        GPIO_LED_DISPLAY_COL_4
    },
    .filas = {
        // A completar por el alumno...
        // ...
        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
    },
// A completar por el alumno...
// ...
    /*      .rutinas_ISR = {
            },*/
};

//-----
// 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;
    //Faltan cosas
    //p_sistema->arkanoPi.p_pantalla = &(led_display.pantalla);
```

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

wiringPiSetupGpio();
InicializaTeclado(&teclado);
InicializaLedDisplay(&led_display);
p_sistema->arkanoPi.p_pantalla = &(led_display.pantalla);
p_sistema->arkanoPi.tmr_actualizacion_juego = tmr_new(tmr_actualizacion_juego_isr);

piLock(STD_IO_BUFFER_KEY);
printf("\nIniciando el juego...\n");
printf("\nBOTONES DE ARKANOPi"
        "\nBotón para encender y resetear -> 1"
        "\nBotón de movimiento a la izquierda -> 4"
        "\nBotón de movimiento a la derecha -> 6"
        "\nBotón de pausa y jugar después de pausa -> F"
        "\nBotón para apagar el juego -> C\n");
piUnlock(STD_IO_BUFFER_KEY);

//
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;
}
```

```
// Lanzamos thread para exploracion del teclado convencional del PC
//result = piThreadCreate (thread_explora_teclado_PC);
/*
if (result != 0) {
    printf("Thread didn't start!!!\n");
    piUnlock(STD_IO_BUFFER_KEY); //
    return -1;
}*/

piUnlock(STD_IO_BUFFER_KEY);

return result;
//return 1;
}

//-----
// FUNCIONES LIGADAS A THREADS ADICIONALES
//-----
/*
PI_THREAD (thread_explora_teclado_PC) {
int teclaPulsada;
while(1) {
delay(10); // Wiring Pi function: pauses program execution for at least 10 ms
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 'c':
    piLock (SYSTEM_FLAGS_KEY);
    flags |= FLAG_TIMER_JUEGO;
    piUnlock (SYSTEM_FLAGS_KEY);
    break;
    case 'd':
    piLock (SYSTEM_FLAGS_KEY);
    flags |= FLAG_MOV_DERECHA;
    piUnlock (SYSTEM_FLAGS_KEY);
    break;
    case 't':
    //Editar por el alumno..
    piLock (SYSTEM_FLAGS_KEY);
    flags |= FLAG_BOTON;
    piUnlock (SYSTEM_FLAGS_KEY);
    printf("Tecla T pulsada!\n");
    fflush(stdout);
    break;
    case 'q':
    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_START, CompruebaBotonPulsado, WAIT_PUSH, InicializaJuego },
            { WAIT_PUSH, CompruebaTimeoutActualizacionJuego, WAIT_PUSH, ActualizarJuego },
            { WAIT_PUSH, CompruebaMovimientoIzquierda, WAIT_PUSH, MuevePalaIzquierda },
            { WAIT_PUSH, CompruebaMovimientoDerecha, WAIT_PUSH, MuevePalaDerecha },
            { WAIT_PUSH, CompruebaPausa, WAIT_PUSH, PausaJuego },
            { WAIT_PUSH, CompruebaFinalPausa, WAIT_PUSH, FinalPausaJuego },
            { WAIT_PUSH, CompruebaFinalJuego, WAIT_END, FinalJuego },
            { WAIT_END, CompruebaBotonPulsado, WAIT_START, ReseteaJuego },
            { -1, NULL, -1, NULL }, };

    // Configuración e inicialización del sistema
    ConfiguraInicializaSistema(&sistema);
```

```
//Iniciación de las máquinas de estados
fsm_t *arkanoPi_fsm = fsm_new(WAIT_START, arkanoPi, &sistema.arkanoPi);

fsm_t *teclado_fsm = fsm_new(TECLADO_ESPERA_COLUMNNA,
                             fsm_trans_excitacion_columnas, &(teclado));
fsm_t *tecla_fsm = fsm_new(TECLADO_ESPERA_TECLA,
                           fsm_trans_deteccion_pulsaciones, &(teclado));
fsm_t *display_fsm = fsm_new(DISPLAY_ESPERA_COLUMNNA,
                              fsm_trans_excitacion_display, &(led_display));

next = millis();
while (1) {
    fsm_fire(arkanoPi_fsm);
    fsm_fire(teclado_fsm);
    fsm_fire(tecla_fsm);
    fsm_fire(display_fsm);

    next += CLK_MS;
    delay_until(next);
}
tmr_destroy((tmr_t*) (sistema.arkanoPi.tmr_actualizacion_juego));

return 0;
}
```

## arkanoPi.h

```
#ifndef
_ARKANOPi_H_

#define _ARKANOPi_H_

#include "systemLib.h"
#include "kbhit.h" // para poder detectar teclas pulsadas sin bloqueo y leer las teclas pulsadas
#include "fsm.h"
#include "tmr.h"
#include "teclado_TL04.h"
#include "arkanoPiLib.h"
#include "ledDisplay.h"

typedef struct {
    tipo_arkanoPi arkanoPi;
    int debug;
} TipoSistema;

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

int CompruebaBotonPulsado (fsm_t* this);
int CompruebaMovimientoIzquierda(fsm_t* this);
int CompruebaMovimientoDerecha(fsm_t* this);
int CompruebaTimeoutActualizacionJuego (fsm_t* this);
int CompruebaFinalJuego(fsm_t* this);
int CompruebaPausa(fsm_t* this);
int CompruebaFinalPausa(fsm_t* this);

//-----
```

```
// FUNCIONES DE ACCION DE LA MAQUINA DE ESTADOS
//-----
void InicializaJuego (fsm_t* this);
void MuevePalaIzquierda (fsm_t* this);
void MuevePalaDerecha (fsm_t* this);
void ActualizarJuego (fsm_t* this);
void FinalJuego (fsm_t* this);
void ReseteaJuego (fsm_t* this);
void PausaJuego (fsm_t* this);
void FinalPausaJuego(fsm_t* this);

//-----
// FUNCIONES DE CONFIGURACION/INICIALIZACION
//-----
int ConfiguraInicializaSistema (TipoSistema *p_sistema);

//-----
// SUBROUTINAS DE ATENCION A LAS INTERRUPCIONES
//-----
void tmr_actualizacion_juego_isr(union sigval value);

//-----
// FUNCIONES LIGADAS A THREADS ADICIONALES
//-----
//PI_THREAD(thread_explora_teclado_PC);

#endif /* ARKANOPH_H_ */
```



## arkanoPiLib.c

```
#include  
"arkanoPiLib.h"
```

```
int ladrillos_basico[NUM_FILAS_DISPLAY][NUM_COLUMNAS_DISPLAY] = {  
    {1,1,1,1,1,1,1,1},  
    {1,1,1,1,1,1,1,1},  
    {0,0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0,0},  
    {0,0,0,0,0,0,0,0},  
};  
  
//-----  
// 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 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;
        }
    }
}

void PausaPantalla (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];
        }
    }
}

//-----
// 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 PausaLadrillos(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];
        }
    }
}

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);
    }

void PausaPelota(tipo_pelota *p_pelota) {
    p_pelota->x;
    p_pelota->y;
}

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 PausaPala(tipo_pala *p_pala) {
    // Pala inicialmente en el centro de la pantalla
    p_pala->x;
    p_pala->y;
    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 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 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] = 8;
        }
        else {
            printf("\n\nPROBLEMAS!!!! posicion y=%d de la pelota INVALIDA!!!\n\n", p_pelota->y);
            fflush(stdout);
        }
    }
    else {
        printf("\n\nPROBLEMAS!!!! posicion x=%d de la pelota INVALIDA!!!\n\n", p_pelota->x);
        fflush(stdout);
    }
}
```

```
void ActualizaPantalla(tipo_arkanoPi* p_arkanoPi, int debug) {

    // 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));

    if (debug) {
        printf("\nDESPUES DE PintaPelota\n");
        fflush(stdout);
        PintaPantallaPorTerminal((tipo_pantalla*)(p_arkanoPi->p_pantalla));
    }
}

void InicializaArkanoPi(tipo_arkanoPi *p_arkanoPi, int debug) {
    ResetArkanoPi(p_arkanoPi);
}
```

```
        ActualizaPantalla(p_arkanoPi, debug);
    }

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));
}

void PausaArkanoPi(tipo_arkanoPi *p_arkanoPi){
    PausaPantalla((tipo_pantalla*)(p_arkanoPi->p_pantalla));
    PausaLadrillos((tipo_pantalla*)&(p_arkanoPi->ladrillos));
    PausaPelota((tipo_pelota*)&(p_arkanoPi->pelota));
    PausaPala((tipo_pala*)&(p_arkanoPi->pala));
}

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;
}

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_ladrillo_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;
        //scores++;
        return 1;
    }
    return 0;
}

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

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

int CompruebaRebotePala (tipo_arkanoPi arkanoPi) {
    if(arkanoPi.pelota.trayectoria.yv > 0) { // Esta condicion solo tiene sentido si la pelota va hacia abajo en la pantalla
        if ((arkanoPi.pelota.x + arkanoPi.pelota.trayectoria.xv >= arkanoPi.pala.x ) &&
            (arkanoPi.pelota.x + arkanoPi.pelota.trayectoria.xv < arkanoPi.pala.x + NUM_COLUMNAS_PALA)) {
            if ((arkanoPi.pelota.y + arkanoPi.pelota.trayectoria.yv >= arkanoPi.pala.y) &&
                (arkanoPi.pelota.y + arkanoPi.pelota.trayectoria.yv < arkanoPi.pala.y + NUM_FILAS_PALA)) {
                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 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 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 CompruebaFinalJuego(fsm_t* this) {
        int result = 0;

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

        return result;
    }

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

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

        return result;
    }

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

        piLock (SYSTEM_FLAGS_KEY);
        result = (flags & FLAG_FINAL_PAUSA);
        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

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

    p_arkanoPi->score=0;
    info="";
    info="Iniciando juego";

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

    InicializaArkanoPi(p_arkanoPi, 0); //valor del parametro debug?? 1?

    tmr_startms((tmr_t*)(p_arkanoPi->tmr_actualizacion_juego), TIMEOUT_ACTUALIZA_JUEGO);
    pseudoWiringPiEnableDisplay(1);
}
```

```
// 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);           //para pasar la direcciopn y que me diga el puntero

    piLock(MATRIX_KEY);
    ActualizaPantalla(p_arkanoPi,0);
    piUnlock(MATRIX_KEY);
}

// 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);
        ActualizaPantalla(p_arkanoPi,0);
        piUnlock(MATRIX_KEY);
    }

    // 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);

        info="Jugando... ";

        piLock (SYSTEM_FLAGS_KEY);
```

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

if(CompruebaReboteParedesVerticales(*p_arkanoPi)){
    //PAREDES SIN REBOTE VERTICAL EN EL NIVEL 3
    if(nivel==3){
        //Si la pelota se encuentra en los limites verticales de la matriz
        //Las coordenadas de la pelota cambian, y se actualizan al lado contrario
        if(p_arkanoPi->pelota.x >= NUM_COLUMNAS_DISPLAY){
            p_arkanoPi->pelota.x = 0;
        }else if(p_arkanoPi->pelota.x <= 0){
            p_arkanoPi->pelota.x = 7;
        }
    }else{
        p_arkanoPi->pelota.trayectoria.xv = -p_arkanoPi->pelota.trayectoria.xv;
    }
}

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

//Si la pelota no choca con la pala, se produce un fallo y se resta 1 a las vidas
if(CompruebaFallo(*p_arkanoPi)) {
    lifes--;
    //Mientras la variable vidas sea mayor que cero, el juego seguira ejecutandose
    //Cada vez que se pierde una vida, se empieza desde 0
    if(lifes>=0){
        p_arkanoPi->score=0;
    }
}
```

```
InicializaPelota(&(p_arkanoPi->pelota));
InicializaLadrillos((tipo_pantalla*)&(p_arkanoPi->ladrillos));

tmr_startms((tmr_t*)(p_arkanoPi->tmr_actualizacion_juego), TIMEOUT_ACTUALIZA_JUEGO);

}else{
    lifes = 0;
    info="Has perdido          ";
    piLock(SYSTEM_FLAGS_KEY);
    flags |= FLAG_FIN_JUEGO;
    piUnlock(SYSTEM_FLAGS_KEY);
    //exit(0);
}
return;
}else if(CompruebaRebotePala(*p_arkanoPi)){
    //Cuando al pelota choca con la pala, se produce un sonido de choque
    printf("%c",7);
    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;
    }
}
```

```
//Si la pelota destruye un ladrillo, se suma 1 a la variable scores
if(CompruebaReboteLadrillo(p_arkanoPi)){
    p_arkanoPi->score++;
    p_arkanoPi->pelota.trayectoria.yv = -p_arkanoPi->pelota.trayectoria.yv;
    //Cuando al pelota choca con la pala, se produce un sonido de choque
    printf("%c",7);
    //Cuando el nivel del juego es 3, puede producirse errores si la pelota está en la coordenada y=1 y x=0 o x=7,
    //por ello con esta condición nos aseguramos que no se produzca ningún error mientras jugamos.
    if(nivel==3){
        if(p_arkanoPi->pelota.x >= NUM_COLUMNAS_DISPLAY){
            p_arkanoPi->pelota.x = 0;
        }else if(p_arkanoPi->pelota.x <= 0){
            p_arkanoPi->pelota.x = 7;
        }
    }
}

if(CalculaLadrillosRestantes(&(p_arkanoPi->ladrillos)) <= 0){
    info="Has perdido ";
    piLock(SYSTEM_FLAGS_KEY);
    flags |= FLAG_FIN_JUEGO;
    piUnlock(SYSTEM_FLAGS_KEY);
    //exit(0);
}
}

ActualizaPosicionPelota(&(p_arkanoPi->pelota));
//Sistema de niveles por puntos
if(p_arkanoPi->score < 3){
    nivel=1;
    speed = TIMEOUT_ACTUALIZA_JUEGO;
```

```
        }else if(p_arkanoPi->score < 5){
            nivel=2;
            speed=1300;
        }else if(p_arkanoPi->score < 10){
            nivel=3;
            speed=1000;

        }else if(p_arkanoPi->score >= 10){
            info="Has ganado!! ";

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

        }else{
            speed = TIMEOUT_ACTUALIZA_JUEGO;
        }

        //Se guarda en la variable bestscore la mejor marca de todos los intentos,
        //luego se muestra por pantalla en el marcador
        if(p_arkanoPi->score>bestscore){
            bestscore=p_arkanoPi->score;
        }

        piLock(MATRIX_KEY);
        ActualizaPantalla(p_arkanoPi,0);
        piUnlock(MATRIX_KEY);

        tmr_startms((tmr_t*)(p_arkanoPi->tmr_actualizacion_juego), speed);
    }
```

```
// 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) {

    info="Juego terminado      ";

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

    //pseudowiringPiEnableDisplay(0);
    //NO se utiliza la ultima sentencia para que se puede imprimir la informacion del juego
}

//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);

    info="Reseteo      ";

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

```
        ResetArkanoPi(p_arkanoPi);
        p_arkanoPi->score=0;
        lifes=3;

    }

//void PausaJuego (void): función encargada de llevar a cabo la
// pausa de la partida.
void PausaJuego(fsm_t* this){
    tipo_arkanoPi* p_arkanoPi;
    p_arkanoPi = (tipo_arkanoPi*)(this->user_data);

    info="Pausa          ";

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

    PausaArkanoPi(p_arkanoPi);

    piLock(MATRIX_KEY);
    ActualizaPantalla(p_arkanoPi,0);
    piUnlock(MATRIX_KEY);
    //El timeout del timer es cero para que el juego se quede congelado
    speed=0;
    tmr_startms((tmr_t*)(p_arkanoPi->tmr_actualizacion_juego), speed);
}

//void PausaJuego (void): función encargada de llevar a cabo el
// fin de pausa de la partida
```

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

    piLock (SYSTEM_FLAGS_KEY);
    flags &= ~FLAG_FINAL_PAUSA;
    piUnlock(SYSTEM_FLAGS_KEY);
    //Determina el valor del timeout después de la pausa, segun el número de scores
    if(p_arkanoPi->score < 5){
        speed=1300;
    }else if(p_arkanoPi->score < 10){
        speed=1000;
    }else{
        speed = TIMEOUT_ACTUALIZA_JUEGO;
    }

    tmr_startms((tmr_t*)(p_arkanoPi->tmr_actualizacion_juego), speed);
}

//-----
// SUBROUTINAS DE ATENCION A LAS INTERRUPCIONES
//-----

void tmr_actualizacion_juego_isr(union sigval value) {
    // A completar por el alumno
    // ...
    piLock (SYSTEM_FLAGS_KEY);
    flags |= FLAG_TIMER_JUEGO;
    piUnlock(SYSTEM_FLAGS_KEY);
}
```



## arkanoPiLib.h

```
#ifndef
_ARKANOPILIB_H_

#define _ARKANOPILIB_H_


#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>


#include "ledDisplay.h"


enum t_direccion {
    ARRIBA_IZQUIERDA,
    ARRIBA,
    ARRIBA_DERECHA,
    ABAJO_DERECHA,
    ABAJO,
    ABAJO_IZQUIERDA,
    IZQUIERDA, // NO PERMITIDA salvo para la pala
    DERECHA, // NO PERMITIDA salvo para la pala
};


// CONSTANTES DEL JUEGO
#define NUM_COLUMNAS_PALA    3
```

```
#define NUM_FILAS_PALA          1
#define MAX_NUM_TRAYECTORIAS  8

typedef struct {
    int ancho;
    int alto;
    int x;
    int y;
} tipo_pala;

typedef struct {
    int xv;
    int yv;
} tipo_trayectoria;

typedef struct {
    tipo_trayectoria posibles_trayectorias[MAX_NUM_TRAYECTORIAS];
    int num_posibles_trayectorias;
    tipo_trayectoria trayectoria;
    int x;
    int y;
} tipo_pelota;

typedef struct {
    tipo_pantalla *p_pantalla; // Esta es nuestra pantalla de juego (matriz 10x7 de labo)
    tipo_pantalla ladrillos;
    tipo_pala pala;
    tipo_pelota pelota;
    tmr_t *tmr_actualizacion_juego;
    int score;
```

```
} tipo_arkanoPi;

//-----
// FUNCIONES DE INICIALIZACION / RESET  DE LOS OBJETOS ESPECIFICOS
//-----

void InicializaLadrillos(tipo_pantalla *p_ladrillos);
void InicializaPelota(tipo_pelota *p_pelota);
void InicializaPala(tipo_pala *p_pala);
void InicializaPosiblesTrayectorias(tipo_pelota *p_pelota);
void InicializaArkanoPi(tipo_arkanoPi *p_arkanoPi, int debug);
void ResetArkanoPi(tipo_arkanoPi *p_arkanoPi);
void ReseteaMatriz(tipo_pantalla *p_pantalla);

//-----
// PROCEDIMIENTOS PARA LA GESTION DEL JUEGO
//-----
void CambiarDireccionPelota(tipo_pelota *p_pelota, enum t_direccion direccion);
void ActualizaPosicionPala(tipo_pala *p_pala, enum t_direccion direccion);
void ActualizaPosicionPelota (tipo_pelota *p_pelota);
int CompruebaReboteLadrillo (tipo_arkanoPi *p_arkanoPi);
int CompruebaReboteParedesVerticales (tipo_arkanoPi arkanoPi);
int CompruebaReboteTecho (tipo_arkanoPi arkanoPi);
int CompruebaRebotePala (tipo_arkanoPi arkanoPi);
int CompruebaFallo (tipo_arkanoPi arkanoPi);
int CalculaLadrillosRestantes(tipo_pantalla *p_ladrillos);

//-----
// PROCEDIMIENTOS PARA LA VISUALIZACION DEL JUEGO
//-----
```

```
void PintaMensajeInicialPantalla (tipo_pantalla *p_pantalla, tipo_pantalla *p_pantalla_inicial);
void PintaMensajeFinalPantalla (tipo_pantalla *p_pantalla);
void PintaPantallaPorTerminal (tipo_pantalla *p_pantalla);
void PintaLadrillos(tipo_pantalla *p_ladrillos, tipo_pantalla *p_pantalla);
void PintaPala(tipo_pala *p_pala, tipo_pantalla *p_pantalla);
void PintaPelota(tipo_pelota *p_pelota, tipo_pantalla *p_pantalla);
void ActualizaPantalla(tipo_arkanoPi* p_arkanoPi, int debug);
void PausaPantalla(tipo_pantalla *p_pantalla);

//-----
// FUNCIONES DE TRANSICION DE LA MAQUINA DE ESTADOS
//-----
int CompruebaBotonPulsado (fsm_t* this);
int CompruebaMovimientoArriba (fsm_t* this);
int CompruebaMovimientoAbajo (fsm_t* this);
int CompruebaMovimientoIzquierda (fsm_t* this);
int CompruebaMovimientoDerecha (fsm_t* this);
int CompruebaTimeoutActualizacionJuego (fsm_t* this);
int CompruebaFinalJuego (fsm_t* this);
int CompruebaPausa(fsm_t* this);
int CompruebaFinalPausa(fsm_t* this);

//-----
// FUNCIONES DE ACCION DE LA MAQUINA DE ESTADOS
//-----
void InicializaJuego (fsm_t* this);
void MuevePalaIzquierda (fsm_t* this);
void MuevePalaDerecha (fsm_t* this);
void ActualizarJuego (fsm_t* this);
void FinalJuego (fsm_t* this);
```

```
void ReseteaJuego (fsm_t* this);
void PausaJuego (fsm_t* this);
void FinalPausaJuego(fsm_t* this);

//-----
// SUBROUTINAS DE ATENCION A LAS INTERRUPCIONES
//-----
void tmr_actualizacion_juego_isr(union sigval value);

#endif /* _ARKANOPILIB_H_ */
```

## dprintf.h

```
#ifndef
__DPRINTF_H__

#define __DPRINTF_H__

#ifdef DEBUG
#include <stdio.h>
# define DPRINTF(fmt, args...) printf(fmt, ## args)
#else
```

```
# define DPRINTF(fmt, args...)
#endif
#endif
```

## fsm.c

```
/*
 * fsm.c
 *
 * Created on: 1 de mar. de 2016
 * Author: Administrador
 */

#include <stdlib.h>
#include "fsm.h"

fsm_t*
fsm_new (int state, fsm_trans_t* tt, void* user_data)
{
    fsm_t* this = (fsm_t*) malloc (sizeof (fsm_t));
    fsm_init (this, state, tt, user_data);
    return this;
}

void
fsm_init (fsm_t* this, int state, fsm_trans_t* tt, void* user_data)
{
```

```
    this->current_state = state;
    this->tt = tt;
    this->user_data = user_data;
}

void
fsm_destroy (fsm_t* this)
{
    free(this);
}

void
fsm_fire (fsm_t* this)
{
    fsm_trans_t* t;
    for (t = this->tt; t->orig_state >= 0; ++t) {
        if ((this->current_state == t->orig_state) && t->in(this)) {
            this->current_state = t->dest_state;
            if (t->out)
                t->out(this);
            break;
        }
    }
}
```

fsm.h

/\*

```
* fsm.h
*
* Created on: 1 de mar. de 2016
* Author: Administrador
*/

#ifndef FSM_H_
#define FSM_H_

typedef struct fsm_t fsm_t;

typedef int (*fsm_input_func_t) (fsm_t*);
typedef void (*fsm_output_func_t) (fsm_t*);

typedef struct fsm_trans_t {
    int orig_state;
    fsm_input_func_t in;
    int dest_state;
    fsm_output_func_t out;
} fsm_trans_t;

struct fsm_t {
    int current_state;
    fsm_trans_t* tt;
    void* user_data;
};

fsm_t* fsm_new (int state, fsm_trans_t* tt, void* user_data);
void fsm_init (fsm_t* this, int state, fsm_trans_t* tt, void* user_data);
void fsm_fire (fsm_t* this);
```



```
void fsm_destroy (fsm_t* this);
```

```
#endif /* FSM_H_ */
```

## kbhit.c

```
#include
"kbhit.h"

#include <sys/select.h>
#include <stdlib.h> // para poder usar NULL
#include <stdio.h> // para poder usar getc, printf...
// #define DEBUG
#include "dprintf.h" // para poder usar DPRINTF

static char ch2=0;

int kbread(void)
{
    char ch=ch2;
    DPRINTF("kbread '%c'\n",ch2);
    ch2=0;
    return ch;
}

int kbhit(void)
{
    struct timeval tv;
```

```
fd_set read_fd;
if (ch2)
{
    DPRINTF("kbhit buffer '%c'\n",ch2);
    return ch2;
}

system ("/bin/stty raw");

/* Do not wait at all, not even a microsecond */
tv.tv_sec=0;
tv.tv_usec=0;

/* Must be done first to initialize read_fd */
FD_ZERO(&read_fd);

/* Makes select() ask if input is ready:
 * 0 is the file descriptor for stdin */
FD_SET(0,&read_fd);

/* The first parameter is the number of the
 * largest file descriptor to check + 1. */
if(select(1, &read_fd,NULL, /*No writes*/NULL, /*No exceptions*/&tv) == -1)
{
    system ("/bin/stty cooked");
    return 0; /* An error occurred */
}

/* read_fd now holds a bit map of files that are
 * readable. We test the entry for the standard
```

```
    * input (file 0). */

if(FD_ISSET(0,&read_fd))
    /* Character pending on stdin */
    {
        ch2=getc(stdin);
        system ("/bin/stty cooked");
        DPRINTF("kbhit '%c'\n",ch2);
        // ungetc(ch2,stdin);
        return ch2;
    }
    /* no characters were pending */
    system ("/bin/stty cooked");
    return 0;
}
```

## ledDisplay.c

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

```
tipo_pantalla pantalla_inicial = {
    .matriz = {
        {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},
    }
};

tipo_pantalla pantalla_final = {
    .matriz = {
        {0,0,0,0,0,0,0},
        {0,0,1,0,0,1,0},
        {0,1,1,0,1,0,0},
        {0,0,0,0,1,0,0},
        {0,0,0,0,1,0,0},
        {0,1,1,0,1,0,0},
        {0,0,1,0,0,1,0},
        {0,0,0,0,0,0,0},
    }
};

// Maquina de estados: lista de transiciones
// {EstadoOrigen, CondicionDeDisparo, EstadoFinal, AccionesSiTransicion }
fsm_trans_t fsm_trans_excitacion_display[] = { { DISPLAY_ESPERA_COLUMNNA,
        CompruebaTimeoutColumnaDisplay, DISPLAY_ESPERA_COLUMNNA,
        ActualizaExcitacionDisplay }, { -1, NULL, -1, NULL }, };

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

```
void InicializaLedDisplay(TipoLedDisplay *led_display) {
    // A completar por el alumno...
    // ...
    for (int i = 0; i < NUM_COLUMNAS_DISPLAY; i++) {
        pinMode(led_display->pines_control_columnas[i], OUTPUT);
        digitalWrite(led_display->pines_control_columnas[i], LOW);
    }
    for (int i = 0; i < NUM_FILAS_DISPLAY; i++) {
        pinMode(led_display->filas[i], OUTPUT);
        digitalWrite(led_display->filas[i], HIGH);
    }
    led_display->tmr_refresco_display = tmr_new(timer_refresco_display_isr);
    tmr_startms((tmr_t*) (led_display->tmr_refresco_display),
        TIMEOUT_COLUMNAS_DISPLAY);
}

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

void ApagaFilas(TipoLedDisplay *led_display) {
    // A completar por el alumno...
    // ...
    for (int i = 0; i < NUM_FILAS_DISPLAY; i++) {
        digitalWrite(led_display->filas[i], HIGH);
    }
}

void ExcitaColumnas(int columna) { //cambiar los LOW y HIGH
```

```
switch (columna) {  
    // A completar por el alumno...  
    // ...  
    case 0:  
        digitalWrite(led_display.pines_control_columnas[0], LOW);  
        digitalWrite(led_display.pines_control_columnas[1], LOW);  
        digitalWrite(led_display.pines_control_columnas[2], LOW);  
        break;  
    case 1:  
        digitalWrite(led_display.pines_control_columnas[0], HIGH);  
        digitalWrite(led_display.pines_control_columnas[1], LOW);  
        digitalWrite(led_display.pines_control_columnas[2], LOW);  
        break;  
    case 2:  
        digitalWrite(led_display.pines_control_columnas[0], LOW);  
        digitalWrite(led_display.pines_control_columnas[1], HIGH);  
        digitalWrite(led_display.pines_control_columnas[2], LOW);  
        break;  
    case 3:  
        digitalWrite(led_display.pines_control_columnas[0], HIGH);  
        digitalWrite(led_display.pines_control_columnas[1], HIGH);  
        digitalWrite(led_display.pines_control_columnas[2], LOW);  
        break;  
    case 4:  
        digitalWrite(led_display.pines_control_columnas[0], LOW);  
        digitalWrite(led_display.pines_control_columnas[1], LOW);  
        digitalWrite(led_display.pines_control_columnas[2], HIGH);  
        break;  
    case 5:  
        digitalWrite(led_display.pines_control_columnas[0], HIGH);
```

```
        digitalWrite(led_display.pines_control_columnas[1], LOW);
        digitalWrite(led_display.pines_control_columnas[2], HIGH);
        break;
    case 6:
        digitalWrite(led_display.pines_control_columnas[0], LOW);
        digitalWrite(led_display.pines_control_columnas[1], HIGH);
        digitalWrite(led_display.pines_control_columnas[2], HIGH);
        break;
    case 7:
        digitalWrite(led_display.pines_control_columnas[0], HIGH);
        digitalWrite(led_display.pines_control_columnas[1], HIGH);
        digitalWrite(led_display.pines_control_columnas[2], HIGH);
        break;
    }
}

void ActualizaLedDisplay(TipoLedDisplay *led_display) {
    // A completar por el alumno...
    // ...
    ApagaFilas(led_display);
    if(led_display -> p_columna < NUM_COLUMNAS_DISPLAY - 1){
        led_display -> p_columna++;
    }else{
        led_display -> p_columna=0;
    }
    ExcitaColumnas(led_display->p_columna);
    for(int i=0; i<NUM_FILAS_DISPLAY; i++){
        if(led_display->pantalla.matriz[i][led_display->p_columna] != 0){
            digitalWrite(led_display->filas[i],LOW);
        }
    }
}
```

```
    }
}

void PintaPantallaPorTerminal(tipo_pantalla *p_pantalla) {
#ifdef __SIN_PSEUDOWIRINGPI__
    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
}

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

int CompruebaTimeoutColumnaDisplay(fsm_t *this) {
    int result = 0;
    TipoLedDisplay *p_ledDisplay;
    p_ledDisplay = (TipoLedDisplay*) (this->user_data);

    // A completar por el alumno...
```



```
// ...

piLock(KEYBOARD_KEY);
result = (p_ledDisplay->flags & FLAG_TIMEOUT_COLUMN_DISPLAY);
piUnlock(KEYBOARD_KEY);

return result;
}

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

void ActualizaExcitacionDisplay(fsm_t *this) {
    TipoLedDisplay *p_ledDisplay;
    p_ledDisplay = (TipoLedDisplay*) (this->user_data);

    // A completar por el alumno...
    // ...
    tmr_startms((tmr_t*) (led_display.tmr_refresco_display),
                TIMEOUT_COLUMN_DISPLAY);
    piLock(MATRIX_KEY);
    led_display.flags &= (~FLAG_TIMEOUT_COLUMN_DISPLAY);
    piUnlock(MATRIX_KEY);

    ActualizaLedDisplay(p_ledDisplay);
    tmr_startms((tmr_t*) (led_display.tmr_refresco_display),
                TIMEOUT_COLUMN_DISPLAY);
}
```

```
//-----  
// SUBROUTINAS DE ATENCION A LAS INTERRUPCIONES  
//-----  
  
void timer_refresco_display_isr(union sigval value) {  
    // A completar por el alumno...  
    // ...  
    piLock(MATRIX_KEY);  
    led_display.flags |= FLAG_TIMEOUT_COLUMNNA_DISPLAY;  
    piUnlock(MATRIX_KEY);  
}
```

## ledDisplay.h

```
#ifndef  
_LEDDISPLAY_H_  
  
#define _LEDDISPLAY_H_  
  
//#include <wiringPi.h>  
#include "pseudowiringPi.h"  
#include "systemLib.h"  
#include "tmr.h"
```

```
// REFRESCO DISPLAY
// ATENCION: Valor a modificar por el alumno
#define TIMEOUT_COLUMNA_DISPLAY 60

#define NUM_PINES_CONTROL_COLUMNAS_DISPLAY 3
#define NUM_COLUMNAS_DISPLAY 8
#define NUM_FILAS_DISPLAY 7

// FLAGS FSM CONTROL DE EXCITACION DISPLAY
// ATENCION: Valores a modificar por el alumno
#define FLAG_TIMEOUT_COLUMNA_DISPLAY 0x01

enum estados_excitacion_display_fsm {
    DISPLAY_ESPERA_COLUMNA
};

typedef struct {
    int matriz[NUM_FILAS_DISPLAY][NUM_COLUMNAS_DISPLAY];
} tipo_pantalla;

typedef struct {
    int pines_control_columnas[NUM_PINES_CONTROL_COLUMNAS_DISPLAY]; // pines_control_columnas
    int filas[NUM_FILAS_DISPLAY];
    int p_columna;
    tipo_pantalla pantalla;
    tmr_t* tmr_refresco_display;
    int flags;
} TipoLedDisplay;
```

```
extern TipoLedDisplay led_display;
extern tipo_pantalla pantalla_inicial;
extern tipo_pantalla pantalla_final;
extern fsm_trans_t fsm_trans_excitacion_display[];

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

void InicializaLedDisplay (TipoLedDisplay *led_display);

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

void ApagaFilas (TipoLedDisplay *led_display);
void ExcitaColumnas (int columna);
void ActualizaLedDisplay (TipoLedDisplay *led_display);

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

int CompruebaTimeoutColumnaDisplay (fsm_t* this);

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

void ActualizaExcitacionDisplay (fsm_t* this);
```

```
//-----  
// SUBROUTINAS DE ATENCION A LAS INTERRUPCIONES  
//-----  
  
void timer_refresco_display_isr (union sigval value);  
  
#endif /* _LEDDISPLAY_H_ */
```

## pseudoWiringPi.c

```
/*  
 * pseudoWiringPi.c  
 *  
 * Created on: 31 de Mar. de 2020  
 * Author: FFM  
 * COMPATIBLE v4.0  
 */
```

```
#include <stdio.h>
#include <stdarg.h>
#include <stdint.h>
#include <stdlib.h>
#include <ctype.h>
#include <poll.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <fcntl.h>
#include <pthread.h>
#include <sys/time.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/wait.h>
#include <sys/ioctl.h>
#include <asm/ioctl.h>

#include <sched.h>
#include <string.h>

#include "pseudoWiringPi.h"

#define BACKUP_NEWLINE "\033[A\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C"

char colors [] = {};

static int waitForInterruptSTDIN_process = 0;
```

```
pthread_t myThread ;
static pthread_mutex_t piMutexes [4] ;

// Misc

static int wiringPiMode = WPI_MODE_UNINITIALISED ;
static pthread_mutex_t pinMutex ;

// Debugging & Return codes

int wiringPiDebug      = FALSE ;
int wiringPiReturnCodes = FALSE ;

// Time for easy calculations

static uint64_t epochMilli ;

// ISR Data

static void (*isrFunctions [64])(void) ;

static int columnaTecladoActiva = -1;
static int columnaDisplayActiva = -1;

static char pseudoTecladoTL04[4][4] = {
    {'1', '2', '3', 'c'},
    {'4', '5', '6', 'd'},
    {'7', '8', '9', 'e'},
```

```
        {'a', '0', 'b', 'f'}
    };

// Matriz interna de pseudoWiringPi para modelar el estado del display emulado
// NOTA: se emula el conjunto display mas HW acondicionamiento al completo, incluido el decoder correspondiente
// que traduce el estado de los 3 pines GPIO definidos al valor de la columna a excitar en cada momento
static int pseudoMatrizColor[7][8] = {
    {31,31,31,31,31,31,31,31},
    {31,31,31,31,31,31,31,31},
    {32,32,32,32,32,32,32,32},
    {32,32,32,32,32,32,32,32},
    {32,32,32,32,32,32,32,32},
    {32,32,32,32,32,32,32,32},
    {32,32,32,32,32,32,32,32},
    {34,34,34,34,34,34,34,34},
};

static int pseudoMatriz[7][8] = {
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
    {0,0,0,0,0,0,0,0},
};

static int GPIO_to_cols[64]; // Array para el mapeo de pines GPIO a columnas del decoder (el valor codificado corresponde a la columna del
display emulado)
static int GPIO_to_rows[64]; // Array para el mapeo de pines GPIO a filas del display
```



```
static int array_flags_check_columnas_teclado [4] = {0x01, 0x02, 0x04, 0x08};
static int array_flags_check_columnas_display [3] = {0x01, 0x02, 0x04};

static int primeraVezDisplay = 1; // Flag util para determinar cuando hay que borrar la pantalla antes de volver a escribirla
// Al ejecutarse desde la terminal (fuera del entorno) aporta la ventaja de evitar el scroll vertical hacia abajo propio del repintado de
la pantalla

static int numWritesAfterColumnEnable[7]; // Array util para determinar cuando un digitalWrite debe actualizar el contenido de pseudoMatriz
// ACLARACION: el display real exige apagar todas las filas antes de proceder a un cambio de excitacion (i.e. de columna)
// De lo contrario, en el momento de aplicar dicha nueva excitacion, las filas podrian estar reflejando un estado diferente
// del que en realidad corresponde a la nueva columna a visualizar
// Ese apagado no plantea mayor problema desde el punto de vista del display real. Sin embargo, a efectos del emulado,
// cuyo comportamiento y estado esta modelado mediante pseudoMatriz, es MUY IMPORTANTE que dicho apagado NO ALTERE O
// MODIFIQUE el contenido de pseudoMatriz o de lo contrario SOLO VEREMOS UN DISPLAY EMULADO CUYO CONTENIDO ES BASICAMENTE 0s
// (TODO APAGADO)

// La emulación del display se apoya en la idea de emplear la salida estandar, la terminal, como sustituto
// del display real. Para ello se ha considerado que era IMPORTANTE que el uso de la terminal con dicho fin
// no debia impedir u obstaculizar el normal uso de la misma para presentar mensajes via printf.
// Por ese motivo, a efectos de la emulación del display se distingue entre dos posibles estados:
//
// 1) EMULACION DEL DISPLAY: la terminal se pone al servicio de la emulación. Bajo este estado
// NO ES POSIBLE HACER USO DE LA TERMINAL PARA LA ESCRITURA O PRESENTACION DE MENSAJES VIA PRINTF
// El estado viene determinado por JUGANDO = 1, para lo cual sera necesario hacer uso de la llamada a:
//
//           pseudoWiringPiEnableDisplay(1);
//
// El lugar adecuado para realizar dicha llamada es al FINAL de:
//
```

```
// void InicializaJuego(fsm_t* this) {}
//
// 2) USO NORMAL DE LA TERMINAL: funcionamiento NORMAL de la terminal, sin emulacion del display.
// Se puede emplear la terminal del mismo modo en que venimos haciendolo sin mayor problema.
// Este estado viene determinado por JUGANDO = 0, para lo cual sera necesario hacer uso de la llamada a:
//
//          pseudoWiringPiEnableDisplay(0);
//
// El lugar adecuado para realizar dicha llamada es al COMIENZO de:
//
// void FinalJuego (fsm_t* this) {}

static int JUGANDO = 0; // Flag general cuyo valor determina si estamos usando el display emulado o no

/*
 * wiringPiFailure:
 *      Fail. Or not.
 *
 *
 *
 */

int wiringPiFailure (int fatal, const char *message, ...)
{
    va_list argp ;
    char buffer [1024] ;

    if (!fatal && wiringPiReturnCodes)
        return -1 ;

    va_start (argp, message) ;
```

```
    vsnprintf (buffer, 1023, message, argp) ;
    va_end (argp) ;

    fprintf (stderr, "%s", buffer) ;
    exit (EXIT_FAILURE) ;

    return 0 ;
}

/*
 * wiringPiSetupGpio:
 *     Must be called once at the start of your program execution.
 *
 * GPIO setup: Initialises the system into GPIO Pin mode and uses the
 *     memory mapped hardware directly.
 *
 *
 *
 */

int wiringPiSetupGpio (void)
{
    int i = 0;

    if (wiringPiDebug)
        printf ("wiringPi: wiringPiSetupGpio called\n") ;

    wiringPiMode = WPI_MODE_GPIO ;

    for(i=0;i<64;i++) {
        GPIO_to_rows[i] = -1;
        GPIO_to_cols[i] = -1;
    }
}
```

```
    }

    GPIO_to_rows[GPIO_LED_DISPLAY_ROW_1] = 0;
    GPIO_to_rows[GPIO_LED_DISPLAY_ROW_2] = 1;
    GPIO_to_rows[GPIO_LED_DISPLAY_ROW_3] = 2;
    GPIO_to_rows[GPIO_LED_DISPLAY_ROW_4] = 3;
    GPIO_to_rows[GPIO_LED_DISPLAY_ROW_5] = 4;
    GPIO_to_rows[GPIO_LED_DISPLAY_ROW_6] = 5;
    GPIO_to_rows[GPIO_LED_DISPLAY_ROW_7] = 6;

    GPIO_to_cols[GPIO_LED_DISPLAY_COL_1] = 0;
    GPIO_to_cols[GPIO_LED_DISPLAY_COL_2] = 1;
    GPIO_to_cols[GPIO_LED_DISPLAY_COL_3] = 2;

    columnDisplayActiva = 0;

    for(i=0;i<7;i++)
        numWritesAfterColumnEnable[i] = 0;

    return 0 ;
}

/*
 * pinMode:
 *      Sets the mode of a pin to be input, output or PWM output
 *      *****
 */

void pinMode (int pin, int mode)
{
```

```
    if (wiringPiMode != WPI_MODE_GPIO) // Sys mode
    {
        printf("[pseudoWiringPi][ERROR!!!][Modo de configuración incorrecto!!!][Use wiringPiSetupGpio ()]\n");
        fflush(stdout);
        return;
    }

    if (mode == INPUT){
        printf("[pseudoWiringPi][pinMode][pin %d][INPUT]\n", pin);
        fflush(stdout);
    }
    else if (mode == OUTPUT){
        printf("[pseudoWiringPi][pinMode][pin %d][OUTPUT]\n", pin);
        fflush(stdout);
    }
}

/*
 * pullUpDownCtrl:
 *     Control the internal pull-up/down resistors on a GPIO pin.
 *     *****
 */

void pullUpDnControl (int pin, int pud)
{
    if ((pin & PI_GPIO_MASK) == 0) // On-Board Pin
    {
        if (wiringPiMode != WPI_MODE_GPIO) // Sys mode
        {
            printf("[pseudoWiringPi][ERROR!!!][Modo de configuración incorrecto!!!][Use wiringPiSetupGpio ()]\n");
        }
    }
}
```

```
        fflush(stdout);
        return;
    }

    switch (pud)
    {
        case PUD_OFF:
            printf("[pseudoWiringPi][pullUpDnControl][pin %d][PUD_OFF]\n", pin);
            break;
        case PUD_UP:
            printf("[pseudoWiringPi][pullUpDnControl][pin %d][PUD_UP]\n", pin);
            break;
        case PUD_DOWN:
            printf("[pseudoWiringPi][pullUpDnControl][pin %d][PUD_DOWN]\n", pin);
            break;
        default:
            printf("[pseudoWiringPi][ERROR!!!][pullUpDnControl][Modo incorrecto!!!][Use PUD_OFF o PUD_DOWN]\n");
            fflush(stdout);
            return ; /* An illegal value */
    }
}

/*
 * waitForInterrupt:
 *   Pi Specific.
 *   Wait for Interrupt on a GPIO pin.
 *   This is actually done via the /sys/class/gpio interface regardless of
 *   the wiringPi access mode in-use. Maybe sometime it might get a better
 *   way for a bit more efficiency.
 */
```

\*\*\*\*\*

\*/

```
int waitForInterruptSTDIN (int mS)
{
    uint8_t c ;
    int i, flagsColumnsChecked;
    int pinesFilasTeclado[4] = {
        GPIO_KEYBOARD_ROW_1,
        GPIO_KEYBOARD_ROW_2,
        GPIO_KEYBOARD_ROW_3,
        GPIO_KEYBOARD_ROW_4};

    // Wait for it ...
    while(1) {
        delay(50); // Wiring Pi function that pauses program execution for at least 10 millisecond

        piLock (STD_IO_BUFFER_KEY);
        if(kbhit()) {
            c = kbread();
            piUnlock (STD_IO_BUFFER_KEY);
            break;
        }
        piUnlock (STD_IO_BUFFER_KEY);
    }

    flagsColumnsChecked = 0;
    while(flagsColumnsChecked<15) { // antes de tirar una pulsacion me aseguro de haber comprobado las 4 columnas
        piLock(KEYBOARD_KEY); // columnaTecladoActiva lo modifican los digitalWrite
        for(i=0;i<4;i++){
```

```
        if(tolower(c) == pseudoTecladoTL04[i][columnaTecladoActiva]){
            piUnlock(KEYBOARD_KEY);
            isrFunctions [pinesFilasTeclado[i]] () ;
            return c;
        }
    }

    flagsColumnsChecked |= array_flags_check_columnas_teclado[columnaTecladoActiva];
    piUnlock(KEYBOARD_KEY);
    // delay para permitir el cambio de excitacion
    delay(5);
}

return c ;
}

/*
 * interruptHandler:
 *   This is a thread and gets started to wait for the interrupt we're
 *   hoping to catch. It will call the user-function when the interrupt
 *   fires.
 *
 *
 *
 */

static void *interruptHandlerSTDIN (UNU void *arg)
{
    //(void)piHiPri (55) ;    // Only effective if we run as root

    for (;;)
        waitForInterruptSTDIN (-1);
}
```



```
    return NULL ;
}

/*
 * piHiPri:
 *   Attempt to set a high priority schedulling for the running program
 *   *****
 */

int piHiPri (const int pri)
{
    struct sched_param sched ;

    memset (&sched, 0, sizeof(sched)) ;

    if (pri > sched_get_priority_max (SCHED_RR))
        sched.sched_priority = sched_get_priority_max (SCHED_RR) ;
    else
        sched.sched_priority = pri ;

    return sched_setscheduler (0, SCHED_RR, &sched) ;
}

/*
 * wiringPiISR:
 *   Pi Specific.
 *   Take the details and create an interrupt handler that will do a call-
 *   back to the user supplied function.
 *   *****
 */
```

```
*/

int wiringPiISR (int pin, int mode, void (*function)(void))
{
    pthread_t threadId ;

    if ((pin < 0) || (pin > 63))
        return wiringPiFailure (WPI_FATAL, "wiringPiISR: pin must be 0-63 (%d)\n", pin) ;

    else if (wiringPiMode == WPI_MODE_UNINITIALISED)
        return wiringPiFailure (WPI_FATAL, "wiringPiISR: wiringPi has not been initialised. Unable to continue.\n") ;
    else if (wiringPiMode != WPI_MODE_GPIO) // Sys mode
    {
        printf("[pseudoWiringPi][ERROR!!!][Modo de configuración incorrecto!!!][Use wiringPiSetupGpio ()]\n");
        fflush(stdout);
        return wiringPiFailure (WPI_FATAL, "wiringPiISR: wiringPi has not been initialised properly. Unable to continue.\n") ;
    }

    // Now export the pin and set the right edge
    isrFunctions [pin] = function ;

    if(!waitForInterruptSTDIN_process) {
        pthread_mutex_lock (&pinMutex) ;
        pthread_create (&threadId, NULL, interruptHandlerSTDIN, NULL) ;
        pthread_mutex_unlock (&pinMutex) ;
        waitForInterruptSTDIN_process = 1;
    }

    return 0 ;
}
```

```
/*
 * digitalWrite:
 *   Set an output bit
 * ****
 */

void digitalWrite (int pin, int value)
{
    int i=0, j=0;

    if ((pin & PI_GPIO_MASK) == 0) {          // On-Board Pin
        if (wiringPiMode != WPI_MODE_GPIO) {    // Sys mode
            printf("[pseudoWiringPi][ERROR!!!][Modo de configuración incorrecto!!!][Use wiringPiSetupGpio ()]\n");
            fflush(stdout);
            return;
        }

        if (pin == GPIO_LED_DISPLAY_COL_1 || pin == GPIO_LED_DISPLAY_COL_2 || pin == GPIO_LED_DISPLAY_COL_3) { // Pines para control
de columnas display
            if (value == HIGH)
                columnaDisplayActiva |= array_flags_check_columnas_display[GPIO_to_cols[pin]];
            else if (value == LOW)
                columnaDisplayActiva &= (~array_flags_check_columnas_display[GPIO_to_cols[pin]]);
            else {
                printf("[pseudoWiringPi][ERROR!!!][digitalWrite][parametro value incorrecto!!! (use HIGH o LOW)]\n");
                fflush(stdout);
                return;
            }
        }
    }
}
```

```
        // Si cambio de columna reseteo contadores de numero de escrituras
        for(i=0;i<7;i++)
            numWritesAfterColumnEnable[i] = 0;
    }

    // Emulacion teclado matricial
    if (value == HIGH) {
        if(pin >= GPIO_KEYBOARD_COL_1 && pin <= GPIO_KEYBOARD_COL_4)
            columnaTecladoActiva = pin;
    }

    if (pin == GPIO_LED_DISPLAY_ROW_1 ||
        pin == GPIO_LED_DISPLAY_ROW_2 ||
        pin == GPIO_LED_DISPLAY_ROW_3 ||
        pin == GPIO_LED_DISPLAY_ROW_4 ||
        pin == GPIO_LED_DISPLAY_ROW_5 ||
        pin == GPIO_LED_DISPLAY_ROW_6 ||
        pin == GPIO_LED_DISPLAY_ROW_7 ) { // Pines para filas display
        if (numWritesAfterColumnEnable[GPIO_to_rows[pin]] == 0) {
            pseudoMatriz[GPIO_to_rows[pin]][columnaDisplayActiva] = !value; // Para encender se escribe LOW, para apagar
HIGH
            numWritesAfterColumnEnable[GPIO_to_rows[pin]]++;
        }
    }

    if(columnaDisplayActiva == 7 && pin == GPIO_LED_DISPLAY_ROW_7) { // Esta condicion equivale a comprobar cuando se termina de
hacer un barrido COMPLETO del display
        // (se ha completado un barrido para todas las columnas) momento que aprovecharemos para repintar nuestro display
emulado
```

```
if(JUGANDO) {
    piLock (STD_IO_BUFFER_KEY);
    if (!primeraVezDisplay) { //Codigo que permite repintar la pantalla SIN SCROLL VERTICAL HACIA ABAJO
        // OJO! solo funciona bien desde la propia terminal de Ubuntu!
        // Desde la pseudo-terminal de Eclipse NO FUNCIONA ya que no procesa bien ni los \b ni los \n (como
bien sabeis)

        for(i=0;i<7;i++)
            printf("%s", BACKUP_NEWLINE);

        printf("\033[A");
        printf("\033[A\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C\033[C");
    }

    // Actualizo pantalla
    printf("\n[ARKANOPI]\n");
    //MARCADOR DEL JUEGO
    printf("MAX. PUNTUACIÓN:%i | VIDAS:%i | NV:%i | INFO:%s \n",bestscore,lifes,nivel,info);
    for(i=0;i<7;i++) {
        for(j=0;j<8;j++)
            if(pseudoMatriz[i][j]) {
                //Esta condición sirve para que cuando el el juego termina, se pueda actualizar la
pantalla,

                //y muestre el mensaje de "game win"
                if(bestscore==10){
                    printf("\033[%dm", pseudoMatrizColor[i][j]);
                    printf("%d\033[0m", pseudoMatriz[i][j]);
                    //printf("%s\033[0m", matriz_game_win);
                    pseudoWiringPiEnableDisplay(0);
                }else{
                    printf("\033[%dm", pseudoMatrizColor[i][j]);
```

```
                printf("%d\033[0m", pseudoMatriz[i][j]);
            }
        }
        else
            printf("%d", pseudoMatriz[i][j]);
    }
    if(i<6)
        printf("\n");
}
fflush(stdout);

primeraVezDisplay = 0;
piUnlock (STD_IO_BUFFER_KEY);
}
}
}

/*
 * pseudoWiringPiEnableDisplay:
 *     Funcion que habilita o deshabilita el display emulado
 *     *****
 */

void pseudoWiringPiEnableDisplay(int estado) {
    if(estado) {
        primeraVezDisplay = 1;
        JUGANDO = 1;
    }
    else {
        JUGANDO = 0;
    }
}
```

```
    }
}

/*
 * delay:
 *     Wait for some number of milliseconds
 *****
 */

void delay (unsigned int howLong)
{
    struct timespec sleeper, dummy ;

    sleeper.tv_sec  = (time_t)(howLong / 1000) ;
    sleeper.tv_nsec = (long)(howLong % 1000) * 1000000 ;

    nanosleep (&sleeper, &dummy) ;
}

/*
 * millis:
 *     Return a number of milliseconds as an unsigned int.
 *     Wraps at 49 days.
 *****
 */

unsigned int millis (void)
{
    uint64_t now ;
```

```
#ifdef OLD_WAY
    struct timeval tv ;

    gettimeofday (&tv, NULL) ;
    now = (uint64_t)tv.tv_sec * (uint64_t)1000 + (uint64_t)(tv.tv_usec / 1000) ;

#else
    struct timespec ts ;

    clock_gettime (CLOCK_MONOTONIC_RAW, &ts) ;
    now = (uint64_t)ts.tv_sec * (uint64_t)1000 + (uint64_t)(ts.tv_nsec / 1000000L) ;
#endif

    return (uint32_t)(now - epochMilli) ;
}

/*
 * piThreadCreate:
 *      Create and start a thread
 *      *****
 */

int piThreadCreate (void (*fn)(void *))
{
    pthread_t myThread ;

    return pthread_create (&myThread, NULL, fn, NULL) ;
}

/*
```



```
* piLock: piUnlock:
*   Activate/Deactivate a mutex.
*   We're keeping things simple here and only tracking 4 mutexes which
*   is more than enough for out entry-level pthread programming
*****
*/
```

```
void piLock (int key)
{
    pthread_mutex_lock (&piMutexes [key]) ;
}
```

```
void piUnlock (int key)
{
    pthread_mutex_unlock (&piMutexes [key]) ;
}
```

## pseudoWiringPi.h

```
/*
* pseudoWiringPi.h
*
*   Created on: 31 de Mar. de 2020
*   Author: FFM
*   COMPATIBLE v4.0
*/
```

```
#ifndef _PSEUDO_WIRINGPI_H_
```

```
#define _PSEUDO_WIRINGPI_H_

#include <pthread.h>
#include "systemLib.h"

// C doesn't have true/false by default and I can never remember which
// way round they are, so ...
// (and yes, I know about stdbool.h but I like capitals for these and I'm old)

#ifndef TRUE
# define TRUE (1==1)
# define FALSE (!TRUE)
#endif

// GCC warning suppressor

#define UNU __attribute__((unused))

// Mask for the bottom 64 pins which belong to the Raspberry Pi
// The others are available for the other devices

#define PI_GPIO_MASK (0xFFFFFC0)

// Handy defines

// wiringPi modes

#define WPI_MODE_PINS 0
#define WPI_MODE_GPIO 1
#define WPI_MODE_GPIO_SYS 2
```

```
#define WPI_MODE_PHYS          3
#define WPI_MODE_PIFACE        4
#define WPI_MODE_UNINITIALISED -1

// Pin modes

#define INPUT          0
#define OUTPUT         1
#define PWM_OUTPUT     2
#define GPIO_CLOCK     3
#define SOFT_PWM_OUTPUT 4
#define SOFT_TONE_OUTPUT 5
#define PWM_TONE_OUTPUT 6

#define LOW            0
#define HIGH           1

// Pull up/down/none

#define PUD_OFF        0
#define PUD_DOWN       1
#define PUD_UP         2

// PWM

#define PWM_MODE_MS     0
#define P                WM_MODE_BAL    1

// Interrupt levels
```

```
#define INT_EDGE_SETUP      0
#define INT_EDGE_FALLING   1
#define INT_EDGE_RISING    2
#define INT_EDGE_BOTH      3

#define MAX_NUM_INPUT_KEYS  4

// Core wiringPi functions
extern int  wiringPiSetupGpio  (void) ;

extern      void pinMode      (int pin, int mode) ;
extern      void pullUpDnControl (int pin, int pud) ;
extern      void digitalWrite  (int pin, int value) ;

// Interrupts
//      (Also Pi hardware specific)
extern int  waitForInterruptSTDIN (int mS) ;
extern int  piHiPri (const int pri);
extern int  wiringPiISR          (int pin, int mode, void (*function)(void)) ;

// Threads

#define PI_THREAD(X)  void *X (UNU void *dummy)

// Failure modes

#define WPI_FATAL      (1==1)
#define WPI_ALMOST     (1==2)

extern int  piThreadCreate      (void *(*fn)(void *)) ;
```

```
extern void piLock          (int key) ;
extern void piUnlock        (int key) ;

extern void      delay      (unsigned int howLong) ;
extern unsigned int millis  (void) ;

extern void pseudoWiringPiEnableDisplay(int estado);

#endif /* _PSEUDO_WIRINGPI_H_ */
```

## systemLib.h

```
#ifndef
_SYSTEMLIB_H_

#define _SYSTEMLIB_H_

#include <time.h>
#include <signal.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
```

```
#include <unistd.h>
#include <sys/time.h>
//#include <wiringPi.h>
#include "pseudowiringPi.h"

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

#define CLK_MS 10 //periodo de actualizacion de la maquina de estados

// ATENCION: Valores a modificar por el alumno
// INTERVALO DE GUARDA ANTI-REBOTES
#define DEBOUNCE_TIME 100
#define TIMEOUT_ACTUALIZA_JUEGO 2000

// 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.

// CLAVES PARA MUTEX
// ATENCION: Valores a modificar por el alumno
#define KEYBOARD_KEY 0 //Cuando utilizo teclado matricial de la raspberry
#define SYSTEM_FLAGS_KEY 1
#define MATRIX_KEY 2 //Matriz de ladrillos
#define STD_IO_BUFFER_KEY 3 //Para el uso mas concreto de la matriz

// Distribucion de pines GPIO empleada para el teclado y el display
// ATENCION: Valores a modificar por el alumno
#define GPIO_KEYBOARD_COL_1 0
```

```
#define GPIO_KEYBOARD_COL_2  1
#define GPIO_KEYBOARD_COL_3  2
#define GPIO_KEYBOARD_COL_4  3
#define GPIO_KEYBOARD_ROW_1  5
#define GPIO_KEYBOARD_ROW_2  6
#define GPIO_KEYBOARD_ROW_3  12
#define GPIO_KEYBOARD_ROW_4  13
```

```
#define GPIO_LED_DISPLAY_COL_1    11
#define GPIO_LED_DISPLAY_COL_2    14
#define GPIO_LED_DISPLAY_COL_3    17
#define GPIO_LED_DISPLAY_COL_4    18
#define GPIO_LED_DISPLAY_ROW_1     4
#define GPIO_LED_DISPLAY_ROW_2     7
#define GPIO_LED_DISPLAY_ROW_3     8
#define GPIO_LED_DISPLAY_ROW_4    10
#define GPIO_LED_DISPLAY_ROW_5    22
#define GPIO_LED_DISPLAY_ROW_6    23
#define GPIO_LED_DISPLAY_ROW_7    24
```

```
// FLAGS FSM CONTROL DE SERPIENTE Y GESTION JUEGO
```

```
// ATENCION: Valores a modificar por el alumno
```

```
#define FLAG_MOV_ARRIBA      0x00
#define FLAG_MOV_ABAJO       0x00
#define FLAG_MOV_DERECHA     0x01
#define FLAG_MOV_IZQUIERDA   0x02
#define FLAG_TIMER_JUEGO     0x04
#define FLAG_BOTON           0x08
#define FLAG_FIN_JUEGO       0x10
#define FLAG_PAUSA           0x20
```

```
#define FLAG_FINAL_PAUSA    0x40
```

```
enum fsm_state {  
    WAIT_START,  
    WAIT_PUSH,  
    WAIT_END};
```

```
extern int flags;  
extern int speed;  
extern int bestscore;  
extern int lifes;  
extern int nivel;  
extern char *info;
```

```
#endif /* SYSTEMLIB_H_ */
```

## 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 }, };

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

void InicializaTeclado(TipoTeclado *p_teclado) {
    // A completar por el alumno...
    // ...
    p_teclado->columna_actual = COLUMNA_1;
    p_teclado->teclaPulsada.col = -1;
    p_teclado->teclaPulsada.row = -1;

    for (int i = 0; i < NUM_COLUMNAS_TECLADO; i++) {
        pinMode(teclado.columnas[i], OUTPUT);
        digitalWrite(teclado.columnas[i], LOW);
    }

    for (int i = 0; i < NUM_FILAS_TECLADO; i++) {
        pinMode(teclado.filas[i], INPUT);
        pullUpDnControl(teclado.filas[i], PUD_DOWN);
        wiringPiISR(teclado.filas[i], INT_EDGE_RISING, teclado.rutinas_ISR[i]);
    }

    teclado.tmr_duracion_columna = tmr_new(timer_duracion_columna_isr);
}
```

```
        tmr_startms((tmr_t*) (teclado.tmr_duracion_columna),
                    TIMEOUT_COLUMNNA_TECLADO);
    }

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

void ActualizaExcitacionTecladoGPIO(int columna) {
    // A completar por el alumno
    // ...
    switch (columna) {
    case COLUMNA_1:
        digitalWrite(GPIO_KEYBOARD_COL_1, HIGH);
        digitalWrite(GPIO_KEYBOARD_COL_2, LOW);
        digitalWrite(GPIO_KEYBOARD_COL_3, LOW);
        digitalWrite(GPIO_KEYBOARD_COL_4, LOW);
        break;
    case COLUMNA_2:
        digitalWrite(GPIO_KEYBOARD_COL_1, LOW);
        digitalWrite(GPIO_KEYBOARD_COL_2, HIGH);
        digitalWrite(GPIO_KEYBOARD_COL_3, LOW);
        digitalWrite(GPIO_KEYBOARD_COL_4, LOW);
        break;
    case COLUMNA_3:
        digitalWrite(GPIO_KEYBOARD_COL_1, LOW);
        digitalWrite(GPIO_KEYBOARD_COL_2, LOW);
        digitalWrite(GPIO_KEYBOARD_COL_3, HIGH);
        digitalWrite(GPIO_KEYBOARD_COL_4, LOW);
        break;
    }
```

```
        case COLUMNA_4:
            digitalWrite(GPIO_KEYBOARD_COL_1, LOW);
            digitalWrite(GPIO_KEYBOARD_COL_2, LOW);
            digitalWrite(GPIO_KEYBOARD_COL_3, LOW);
            digitalWrite(GPIO_KEYBOARD_COL_4, HIGH);
            break;
    }

    tmr_startms((tmr_t*) (teclado.tmr_duracion_columna),
                TIMEOUT_COLUMNA_TECLADO);
    //Se inicializa al ActualizaExcitacionTecladoGPIO cada vez que se actualiza
}

//-----
// 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);
    // A completar por el alumno...
    // ...

    piLock(KEYBOARD_KEY);
    result = (p_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);
    // A completar por el alumno
    // ...

    piLock(KEYBOARD_KEY);
    result = (p_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);
    // A completar por el alumno
    // ...
    piLock(KEYBOARD_KEY);
    teclado.flags &= (~FLAG_TIMEOUT_COLUMNA_TECLADO);
    piUnlock(KEYBOARD_KEY);

    p_teclado->columna_actual++;
    int col_act = p_teclado->columna_actual;
```

```
    if (p_teclado->columna_actual < 4) {
        p_teclado->columna_actual = col_act;
    } else {
        p_teclado->columna_actual = 0;
    }

    // Llamada a ActualizaExcitacionTecladoGPIO con columna a activar como argumento
    ActualizaExcitacionTecladoGPIO(p_teclado->columna_actual);
    tmr_startms((tmr_t*) (teclado.tmr_duracion_columna),
                TIMEOUT_COLUMNNA_TECLADO);
}

void ProcesaTeclaPulsada(fsm_t *this) {
    TipoTeclado *p_teclado;
    p_teclado = (TipoTeclado*) (this->user_data);
    // A completar por el alumno
    // ...

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

    switch (p_teclado->teclaPulsada.col) {
    case COLUMNA_1:
        //IZQUIERDA
        if (p_teclado->teclaPulsada.row == FILA_2) {
            piLock(SYSTEM_FLAGS_KEY);
            flags |= FLAG_MOV_IZQUIERDA;
            piUnlock(SYSTEM_FLAGS_KEY);
        }
    }
}
```

```
        p_teclado->teclaPulsada.row = -1;
        p_teclado->teclaPulsada.col = -1;
//ENCENDIO O RESETEO
    } else if (p_teclado->teclaPulsada.row == FILA_1) {
        piLock(SYSTEM_FLAGS_KEY);
        flags |= FLAG_BOTON;
        piUnlock(SYSTEM_FLAGS_KEY);

        p_teclado->teclaPulsada.row = -1;
        p_teclado->teclaPulsada.col = -1;
    }
    break;

case COLUMNA_2:

    break;

case COLUMNA_3:
    //DERECHA
    if (p_teclado->teclaPulsada.row == FILA_2) {

        piLock(SYSTEM_FLAGS_KEY);
        flags |= FLAG_MOV_DERECHA;
        piUnlock(SYSTEM_FLAGS_KEY);

        p_teclado->teclaPulsada.row = -1;
        p_teclado->teclaPulsada.col = -1;
    }
    break;
```

```
case COLUMNA_4:
    //FIN DE JUEGO
    if (p_teclado->teclaPulsada.row == FILA_1) {
        piLock (SYSTEM_FLAGS_KEY);
        flags |=FLAG_FIN_JUEGO;
        piUnlock (SYSTEM_FLAGS_KEY);
        p_teclado->teclaPulsada.row = -1;
        p_teclado->teclaPulsada.col = -1;
    }
    //PAUSA
    if (p_teclado->teclaPulsada.row == FILA_4) {
        //Si el timeout no es cero, el sistema no esta en pausa,
        //por lo tanto se puede pausar
        if(speed!=0){
            piLock (SYSTEM_FLAGS_KEY);
            flags |=FLAG_PAUSA;
            piUnlock (SYSTEM_FLAGS_KEY);
        }
        //Si el timeout del sistema está en cero, es que está pausado,
        //por lo que se pondrá a jugar de nuevo
        if(speed==0){
            piLock (SYSTEM_FLAGS_KEY);
            flags |=FLAG_FINAL_PAUSA;
            piUnlock (SYSTEM_FLAGS_KEY);
        }

        p_teclado->teclaPulsada.row = -1;
        p_teclado->teclaPulsada.col = -1;
    }
}
```

```
        break;

    default:
        break;
}

}

//-----
// SUBROUTINAS DE ATENCION A LAS INTERRUPCIONES
//-----

void teclado_fila_1_isr(void) {
    // A completar por el alumno
    // ...
    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);

    teclado.debounceTime[FILA_1] = millis() + DEBOUNCE_TIME;
}

void teclado_fila_2_isr(void) {
    // A completar por el alumno
```



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

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

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

void teclado_fila_3_isr(void) {
    // A completar por el alumno
    // ...
    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;
```

```
}

void teclado_fila_4_isr(void) {
    // A completar por el alumno
    // ...
    if (millis() < teclado.debounceTime[FILA_4]) {
        teclado.debounceTime[FILA_4] = millis() + DEBOUNCE_TIME;
        return;
    }

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

    teclado.debounceTime[FILA_4] = millis() + DEBOUNCE_TIME;
}

void timer_duracion_columna_isr(union sigval value) {
    // A completar por el alumno
    // ...
    piLock(KEYBOARD_KEY);
    teclado.flags |= FLAG_TIMEOUT_COLUMNA_TECLADO;
    piUnlock(KEYBOARD_KEY);
}
```

## teclado\_TL04.h

```
#ifndef
_TECLADO_TL04_H_

#define _TECLADO_TL04_H_

#include "systemLib.h"

// REFresco TECLADO
#define TIMEOUT_COLUMNNA_TECLADO    25

#define NUM_COLUMNAS_TECLADO        4
#define NUM_FILAS_TECLADO           4

// FLAGS FSM CONTROL DE EXCITACION TECLADO Y FSM GESTION TECLAS PULSADAS
// ATENCION: Valores a modificar por el alumno
#define FLAG_TIMEOUT_COLUMNNA_TECLADO    0x01    //?
#define FLAG_TECLA_PULSADA                0x02

enum columns_values {
    COLUMNA_1,
    COLUMNA_2,
    COLUMNA_3,
    COLUMNA_4,
};

enum rows_values {
    FILA_1,
    FILA_2,
```

```
FILA_3,
FILA_4
};

enum estados_excitacion_teclado_fsm {
    TECLADO_ESPERA_COLUMNNA
};

enum estados_deteccion_pulsaciones_teclado_fsm {
    TECLADO_ESPERA_TECLA
};

typedef struct {
    int col;
    int row;
} TipoTecla;

typedef struct {
    int columnas[NUM_COLUMNAS_TECLADO]; // Array con los valores BCM de los pines GPIO empleados para cada columna
    int filas[NUM_FILAS_TECLADO]; // Array con los valores BCM de los pines GPIO empleados para cada fila
    int debounceTime[NUM_FILAS_TECLADO]; // // Array de variables auxiliares para la implementacion de mecanismos anti-
rebotes para cada entrada de interrupcion
    void (*rutinas_ISR[NUM_FILAS_TECLADO]) (void); // Array de punteros a procedimientos de atencion a las interrupciones
ligados al teclado
    int columna_actual; // Variable que almacena el valor de la columna que esta activa
    TipoTecla teclaPulsada; // Variable que almacena la ultima tecla pulsada
    tmr_t* tmr_duracion_columna; // Temporizador responsable de medir el tiempo de activacion de cada columna
    int flags; // Variable para gestion de flags especificamente ligados a la gestion del teclado
} TipoTeclado;
```

```
extern TipoTeclado teclado;
extern fsm_trans_t fsm_trans_excitacion_columnas[];
extern fsm_trans_t fsm_trans_deteccion_pulsaciones[];
extern int flags; // Flags generales de sistema (necesario para comunicacion inter-FMs)

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

void InicializaTeclado(TipoTeclado *p_teclado);

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

void ActualizaExcitacionTecladoGPIO (int columna);

//-----
// FUNCIONES DE ENTRADA O DE TRANSICION DE LAS MAQUINAS DE ESTADOS
//-----

int CompruebaTimeoutColumna (fsm_t* this);
int CompruebaTeclaPulsada (fsm_t* this);

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

void TecladoExcitaColumna (fsm_t* this);
void ProcesaTeclaPulsada (fsm_t* this);
```

```
//-----  
// SUBROUTINAS DE ATENCION A LAS INTERRUPCIONES  
//-----  
  
void teclado_fila_1_isr (void);  
void teclado_fila_2_isr (void);  
void teclado_fila_3_isr (void);  
void teclado_fila_4_isr (void);  
void timer_duracion_columna_isr (union sigval value);  
  
#endif /* _TECLADO_TL04_H_ */
```

## tmr.c

```
/*  
 * tmr.c  
 *  
 * Created on: 1 de mar. de 2016  
 * Author: Administrador  
 */  
  
#include "tmr.h"  
#include <stdlib.h>  
#include <signal.h>
```

```
#include <time.h>

tmr_t*
tmr_new (notify_func_t isr)
{
    tmr_t* this = (tmr_t*) malloc (sizeof (tmr_t));
    tmr_init (this, isr);
    return this;
}

void
tmr_init (tmr_t* this, notify_func_t isr) {
    this->se.sigev_notify = SIGEV_THREAD;
    this->se.sigev_value.sival_ptr = &(this->timerid);
    this->se.sigev_notify_function = isr;
    this->se.sigev_notify_attributes = NULL;
    timer_create (CLOCK_REALTIME, &(this->se), &(this->timerid)); /* o CLOCK_MONOTONIC si se soporta */
}

void
tmr_destroy(tmr_t* this)
{
    tmr_stop (this);
    free(this);
}

void
tmr_startms(tmr_t* this, int ms) {
    this->spec.it_value.tv_sec = ms / 1000;
```

```
    this->spec.it_value.tv_nsec = (ms % 1000) * 1000000;  
    this->spec.it_interval.tv_sec = 0;  
    this->spec.it_interval.tv_nsec = 0;  
    timer_settime (this->timerid, 0, &(this->spec), NULL);  
}
```

```
void  
tmr_stop (tmr_t* this) {  
    timer_delete (this->timerid);  
}
```

## tmr.h

```
/*  
 * tmr.h  
 *  
 * Created on: 1 de mar. de 2016  
 * Author: Administrador  
 */  
  
#ifndef TMR_H_  
#define TMR_H_  
  
#include <signal.h>  
#include <time.h>
```



```
struct tmr_t {
    timer_t timerid;
    struct itimerspec spec;
    struct sigevent se;
};
typedef struct tmr_t tmr_t;

typedef void (*notify_func_t) (union sigval);

tmr_t* tmr_new (notify_func_t isr);
void tmr_init (tmr_t* this, notify_func_t isr);
void tmr_destroy(tmr_t* this);
void tmr_startms(tmr_t* this, int ms);
void tmr_stop (tmr_t* this);

#endif /* TMR_H_ */
```