[Nombre de la compañía]

Práctica 2 y 3

Reversi

Martin
[Fecha]

Guion

Resumen

Introducción
Objetivos
Metodología

Durante el desarrollo de las prácticas 2 y 3 se han seguido una serie de pasos, incorporado una serie de componentes y se han diseñado una serie de funcionalidades nuevas que a continuación se explican.

Primeros pasos

En primer lugar, realizamos un estudio de la documentación necesaria a la hora de diseñar la gestión de excepciones y periféricos, a continuación, estudiamos también los ficheros fuente de ejemplo correspondientes a los periféricos de entrada y salida con el objetivo de aprender el funcionamiento de estos.

En último lugar se estudió la forma de ejecución parcial y compilación condicional con el objetivo final de ejecutar el proyecto sin la necesidad del uso de la placa, esto se logra mediante las directivas *#ifdef*, *#else* y *#endif.*

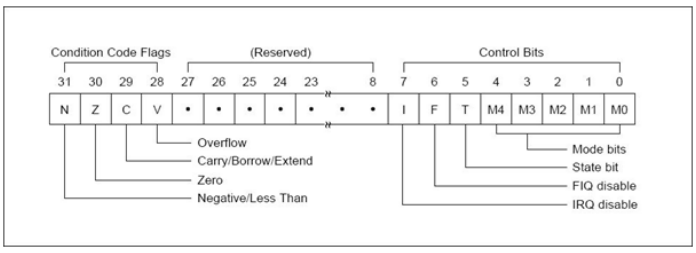
Control de excepciones

La primera cuestión a solucionar es por tanto la creación de un sistema capaz de capturar las excepciones de los tipos abort, software y undefined, para ello se diseño inicialmente un fichero fuente llamado "*controlador_excepciones.c"* que inicializa, captura y valida las excepciones antes mencionadas.

Para capturarlas y, mediante el procedimiento *"void ERROR_ISR()"* que en primer lugar captura la palabra de estado con la siguiente instrucción:

```
asm("ldr r0, =estado\n\t"
    "MRS r1, SPSR\n\t"
    "str r1,[r0]");
```

Una vez se ha obtenido la palabra de estado se aplica un filtro para obtener el contenido de los 5 bits menos significativos que indican el modo en el que se está ejecutando el procesador que determinará el tipo de excepción producida.



En función de estos bits mostraremos por el display 7 segmentos una A para las excepciones del tipo swi, B para las de tipo abort y C para las de tipo undefined.

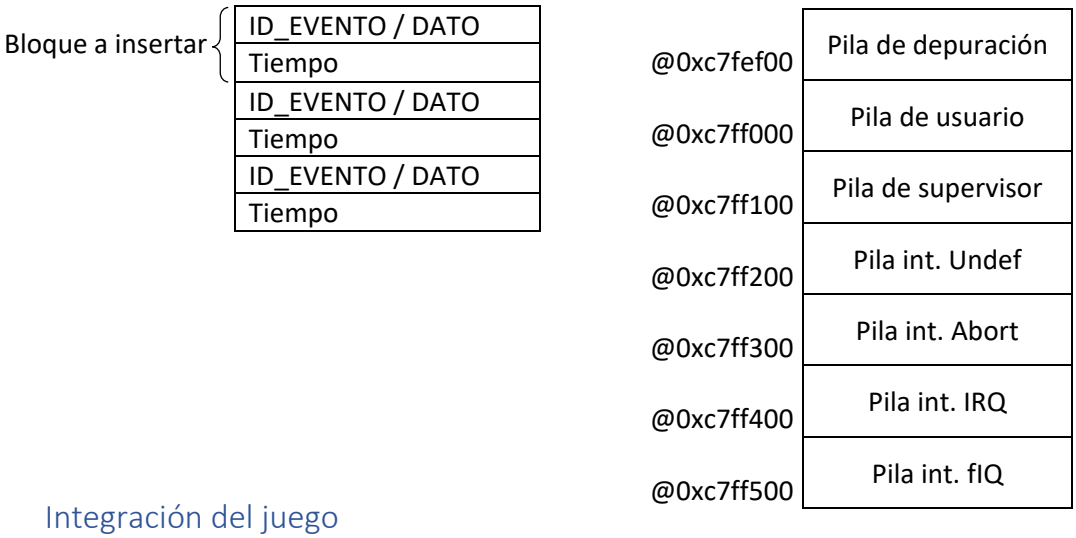
Por ultimo, existe la función test_excepciones() que con ayuda del fichero fuente *"generardor_interrupciones.asm"* que provoca las excepciones antes mencionadas con el objetivo de verificar el correcto funcionamiento de nuestro capturador de excepciones.

Pila de depuración

Otra nueva funcionalidad nueva es la correspondiente a la pila de depuración que, además de servir para la depuración, será una pieza fundamental en nuestro programa ya que todas las comunicaciones entre periféricos de entrada y salida se comunicarán con nuestro programa a través de esta.

La pila actúa como una pila circular actuará como una pila circular de un tamaño de 256 bits que estará en una dirección de memoria inmediatamente anterior a las pilas del sistema, definidas en el fichero *"44binit.asm".*

La pila (además de la inicialización con *init_stack()*) permite la inserción de datos y el instante en el que se ha insertado, el dato será la concatenación de un identificador ID_EVENTO y la información a guardar DATO.



Integración del juego

Latido

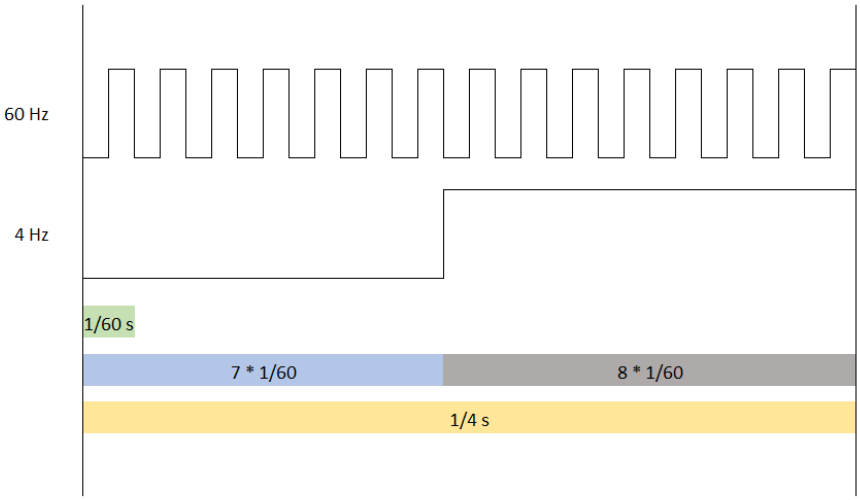
Con el objetivo de mostrar al usuario que la ejecución se está llevando con normalidad aparece el latido, que simula el pulso constante que enciende y apaga un led. Esto se logra gracias a la implementación de un timer (*Timer.c*) que genera sesenta eventos por segundo y los apila en la pila de depuración. Posteriormente se desapilará ese evento y se procesará en el fichero fuente "*controlador_latido.c".*

El latido se muestra a una frecuencia de 4 hercios, para conseguir esto se ha transformado la frecuencia de 60Hz hasta reducirla a cuatro de la siguiente forma:

Puesto que en un segundo se deben encender 4 veces el led (4Hz) permanecerá la mitad de este tiempo apagado y la otra mitad encendido por lo que

Para reducir la frecuencia inicial se ha dividido esta (60Hz) por la frecuencia de parpadeo (4Hz), de este modo tenemos que cada 15 eventos el led se ha encendido y apagado una vez.

Sin embargo, es evidente que no se puede dividir 15 eventos en dos partes obteniendo un numero entero, por lo que uno de los dos estados deberá durar más, en este caso se ha optado por que el estado encendido del led dure más.



Botones

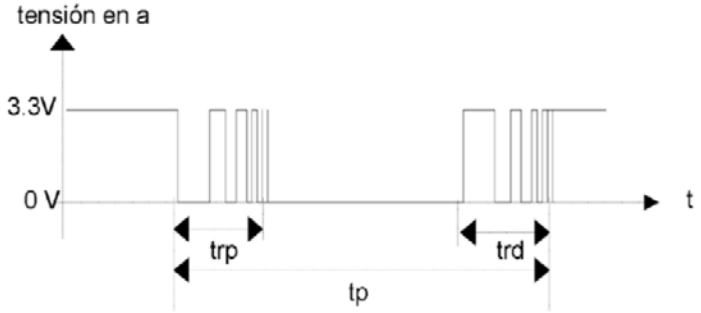
- Funcionamiento:

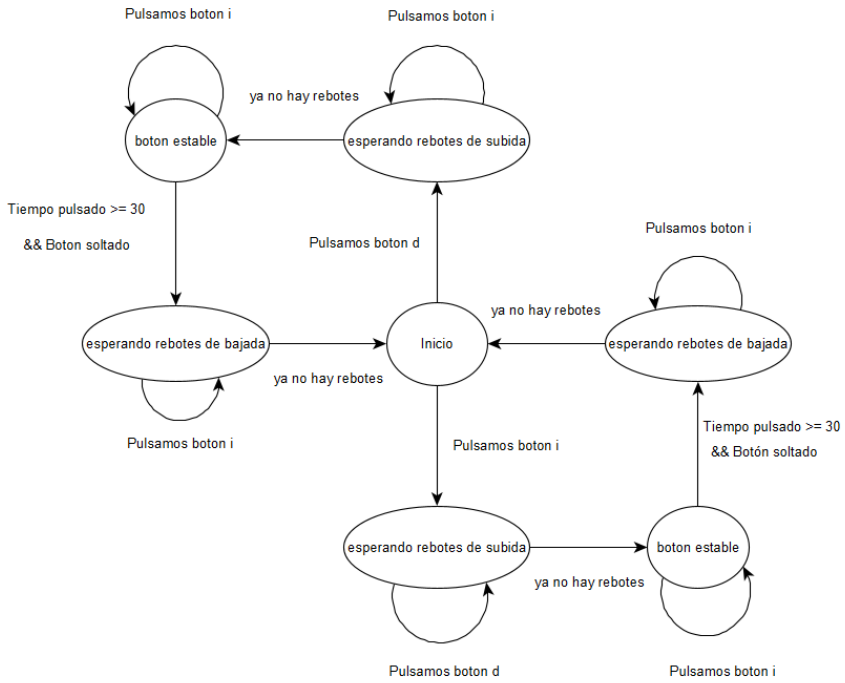
  El funcionamiento de los botones se define en el fichero *button.c* donde se especifican una serie funcionalidades primitivas básicas correspondientes a los botones, tales como la inicialización, reinicio, su deshabilitación y las interrupciones que genera.

Botones antirebotes.

Cada vez que se pulsa uno de los botones se genera una interrupción, sin embargo, no solo se produce esta, sino que se producen una serie de interrupciones indeseadas que llamaremos rebotes.

Estos rebotes se producen al presionar y al soltar el botón como se muestra en la figura x, por lo que se ha diseñado un autómata que representa el filtrado de estos rebotes (Figura x).



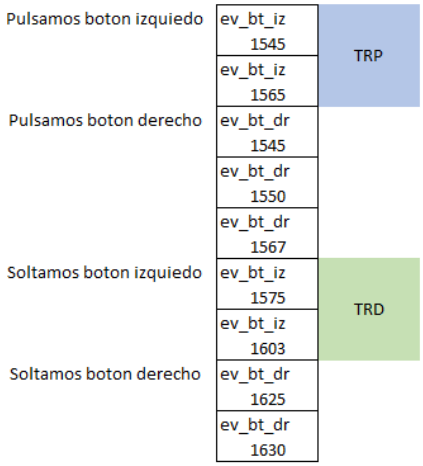


Se puede apreciar en el autómata que disponemos de dos botones (i y d) que parten de un estado depresionado al que llamamos inicial que cambiará en el momento en que pulsemos uno de estos. Una vez hemos pulsado el botón y ha llegado la primera interrupción del botón procedemos a deshabilitar las interrupciones y esperar un tiempo a que terminen los rebotes producidos al pulsar (TRP), este estado se corresponde con el estado "*esperando rebotes de subida*" en el autómata.
Cuando haya transcurrido el TRP asumimos que el botón se mantiene pulsado por lo que, mediante una encuesta, iremos verificando periódicamente si el botón se ha soltado, en este caso terminaría el estado "*botón estable*" pasando al estado "*esperando rebotes de bajada*".
El estado "*esperando rebotes de bajada*" es similar al de los rebotes de subida ya que el proceso es el mismo, en este caso esperaremos un tiempo de depresión (TRD) para después volver a habilitar las interrupciones del botón y poder iniciar de nuevo este proceso. Puesto que las interrupciones permanecen deshabilitadas, no tiene ningún efecto pulsar el segundo botón como se muestra en el autómata.

El tiempo TRP y TRD puede cambiar dependiendo de la placa en la que se ejecute el código, en este caso se ha decidido que ambos tiempos serán de 30ms calculados a través una serie de mediciones realizadas siguiendo los siguientes pasos:

- No deshabilitamos las interrupciones al pulsar el botón.
- Apilamos eventos diferentes al pulsar cada uno de los botones.
- Pulsamos el botón izquierdo y se apila ese evento junto al instante en el que se ha pulsado
- Sin soltar el botón izquierdo pulsamos el derecho
- Soltamos el botón izquierdo
- Soltamos el botón derecho
- Observamos la pila que quedaría de manera similar a la figura x
- Medimos el tiempo registrado entre el primer evento del botón izquierdo y el ultimo (TRP).

- Medimos el tiempo registrado entre el primer evento del botón izquierdo tras soltarlo y el ultimo (TRD).



El proceso anterior se implementa en un fichero fuente llamado "*botón_antirebotes.c*" que actúa como una librería de alto nivel que gestiona los botones, además, para la gestión de los rebotes necesitamos un cronometro con la que gestionar el tiempo de TRP y TRD. Este timer se comunica con nuestra librería a través de la pila y el programa principal de forma que botón antirebotes inicia el timer, este apila periódicamente (cada 10ms) eventos, el programa principal los procesa y botón antirebotes los trata.

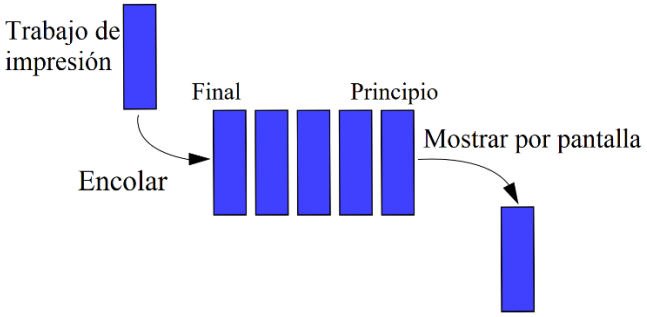
Panel táctil

Pantalla LCD

En la práctica 3, se exige que se juegue a través del panel LCD de la placa, para ello se ha estudiado la documentación de pantalla y el proyecto de ejemplo para desarrollar una librería que controle la impresión por pantalla de la interfaz del juego.

La librería de impresión por pantalla se divide en dos ficheros fuente, "*pantalla_revesi.c*" que contiene una serie de funciones de alto nivel para controlar la pantalla y "*lcd.c*" de forma que al igual que sucede con la clase "*button.c*" contiene una serie de funciones primitivas orientadas al control hardware.

- Funcionamiento LCD: RELLENAR ESTO
- Arquitectura de la librería: Durante la ejecución del juego se producen una serie de eventos que necesitan modificar el contenido de la pantalla, solicitan la impresión al módulo *pantalla_reversi.c* que puede estar ocupado, en ese caso se encola en la estructura de datos FIFO "*cola_pantalla.c*" (figura x)



- Pintado del tablero
- Pintado de info
- Pintado de fichas

Modo supervisor

Hasta ahora y durante toda la ejecución del programa se ha hecho con el procesador en modo supervisor, sin embargo, en la práctica 3 se nos exige la ejecución en modo usuario, para ello se ha diseñado un módulo llamado "*control_modo_procesador.c*" desde el cual se controla el cambio y obtención del estado del procesador. En este módulo se encuentra la función "*cambiar_modo_usuario()*" que ejecuta el siguiente código ensamblador:

```
mrs r0, cpsr
bic r0, r0, #0x1f
orr r1, r0, #0x10
msr CPSR_c, r1
mov sp, user_stack
```

En este código guardamos en el registro r0 el registro cpsr (que contiene el modo del procesador), a continuación, limpiamos los cinco bits menos significativos (bits de modo) para mas adelante escribir el numero #0x10 que corresponde con el modo de usuario.

Utilizamos el comando msr acompañado del parámetro CPSR con el flag c que indica que se van a almacenar los bits de control (0-7) en la palabra de estado que hemos guardado en el registro r1.

En ultimo lugar hay que tener en cuenta que se puede haber cambiado el puntero a pila a alguna de las pilas de los otros modos de procesador, por ello hay que restaurarlo de nuevo a la ubicación de la pila de usuario (user_stack).

FIQ

El programa hasta ahora únicamente ha contado con interrupciones IRQ para el control de los dispositivos de entrada y salida, en la práctica 3 hemos modificado esto para incorporar también interrupciones FIQ en el módulo "*timer2.c*".

Para que el timer 2 capture las interrupciones FIQ se ha añadido la instrucción "void timer2_ISR(void) __attribute__((interrupt("FIQ")));" en la que asignamos a la función timer2_ISR(void) las interrupciones FIQ. Esta función será integrada en el vector de interrupciones mediante la variable pISR_FIQ, que será una rutina de servicio genérica de las FIQ ya que estas no soportan el modo autovectorizado (Si hubiera varias fuentes de interrupción FIQ deberíamos encuestar a la hora de tratar las interrupciones).

Plataforma autónoma

En la práctica 1 y 2 se ejecutaba el programa a través del pc, sin embargo, en la última práctica se solicita que el programa sea capaz de ejecutarse en la placa sin la necesidad de conectarla con el pc. Esto implica que nuestro programa, antes ubicado exclusivamente en la memoria RAM, se almacene en la memoria flash de forma que se pueda ejecutar siempre de forma independiente al pc.

Para volcar el programa en la memoria flash se ha utilizado el siguiente comando para generar el archivo binario a partir del archivo reversi.elf:

"`arm-none-eabi-objcopy -O binary reversi.elf reversi.bin`"

El siguiente comando volcaría en la memoria flash el programa:

"`openocd-0.7.0.exe -f test/arm-fdi-ucm.cfg -c "program D:/reversi.bin 0x00000000`"

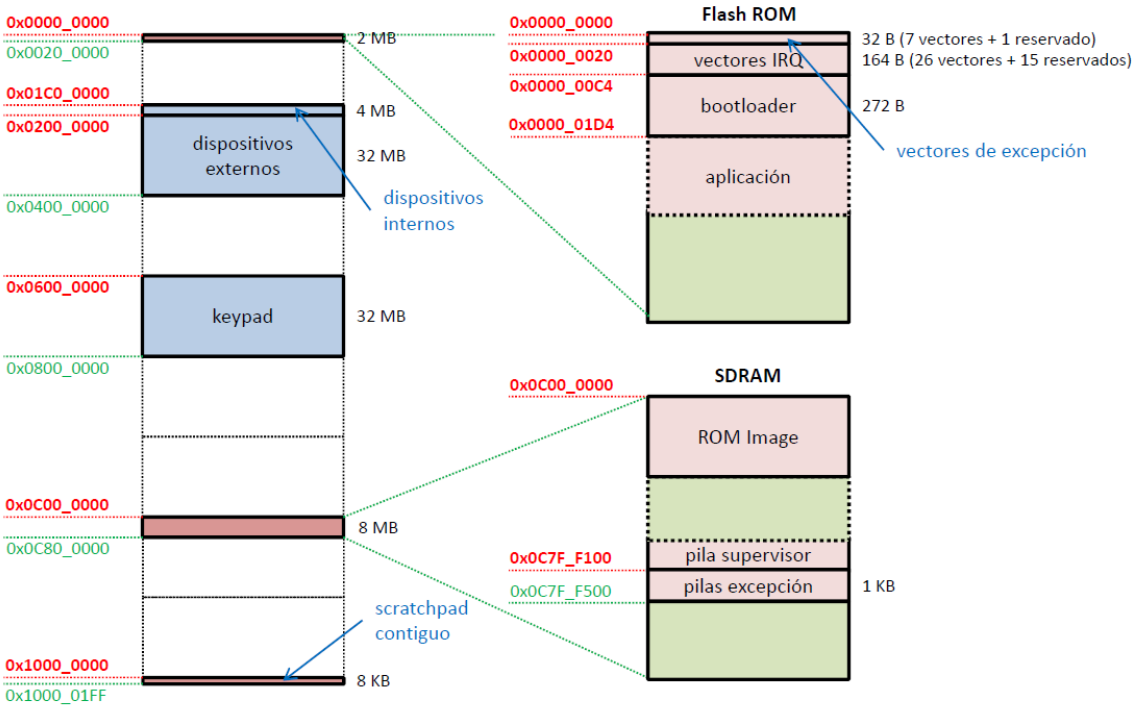
Sin embargo, nuestro linker está configurado para ejecutar en la memoria RAM, por lo que debemos copiar el contenido de la memoria flash en la memoria RAM cada vez que se arranque la placa. Por estos motivos nos hemos visto en la necesidad de modificar el bootloader 44binit.asm.

La figura x se corresponde con el framgento de código encargado de copiar de la memoria flash a la memoria RAM, sin embargo, nuestro linker asigna la misma dirección a la memoria al final de la memoria ROM y el inicio de la memoria RAM (Image_RO_Limit y Image_RW_Base respectivamente) por lo que no copia nada.

```
    LDR     r0, =Image_RO_Limit     /* Get pointer to ROM data */
```

```
    LDR     r0, =0x0
    LDR     r1, =Image_RO_Base
    LDR     r3, =Image_ZI_Base

    CMP     r0, r1          /* Check that they are different */
    BEQ     F1
F0:
    CMP     r1, r3          /* Copy init data*/
    LDRCC   r2, [r0], #4    /* --> LDRCC r2, [r0] + ADD r0, r0, #4 */
    STRCC   r2, [r1], #4    /* --> STRCC r2, [r1] + ADD r1, r1, #4 */
    BCC     F0
```

Se ha cambiado el contenido de r0 (dirección de inicio del contenido a copiar) asignándole la dirección de memoria donde comienza la memoria flash (0x0) el registro r1 (dirección de inicio donde se va a copiar, la RAM) como se muestra en la figura x. La etiqueta Image_RO_Limit representa el final de la ROM que como se ha explicado ya es el comienzo de la memoria RAM según el linker.



Por último, hay que modificar la ubicación del registro de control de memoria que está ubicado en SMRDATA, sin embargo, esta etiqueta apunta a un registro en memoria RAM y nuestro registro con la configuración está volcado en la memoria flash.

Modificamos esta ubicación para que la encuentre en la flash restándole a la etiqueta SMRDATA la dirección de inicio de la memoria RAM (0xc000000) por lo que la etiqueta se ubicará misma posición relativa a la RAM pero en la ROM.

Código antiguo: `ldr r0, =SMRDATA`
Código nuevo: `ldr r0, =(SMRDATA-0xc000000)`

Arquitectura de la aplicación y jugabilidad

-Jugabilidad por botones:

En la práctica 2 la jugabilidad por botones consiste en que al pulsar el botón izquierdo se incrementará el valor de fila y columna mostrándola por pantalla y a continuación pulsando el botón derecho para confirmar. Ejemplo de una jugada:

Inicialmente el 7 segmentos muestra una letra F, a continuación, pulsamos repetidamente el botón izquierdo.

Se ha implementado una mejora y es que si se mantiene el botón pulsado durante un tercio de segundo se incrementará automáticamente sin la necesidad de pulsar el botón varias veces para incrementar.

Como se ve en la figura X cuando el contador llega al numero 8 se reinicia a uno puesto que nuestro tablero únicamente contiene ocho filas y columnas. Al pulsar el botón derecho se mostrará la letra C de columna y se repetirá el proceso descrito en la figura x para seleccionar la columna donde se ubicará la jugada.

Por último, al pulsar el botón derecho y confirmar la columna se procederá a realizar la jugada y volver a empezar.

Este proceso se implementa en el módulo *jugada_por_botones.c principalmente compuesto por el método evento_botones_producido(int evento),* que actuaría como la implementación del autómata descrito anteriormente.

En la figura x se puede observar como tenemos cuatro estados:

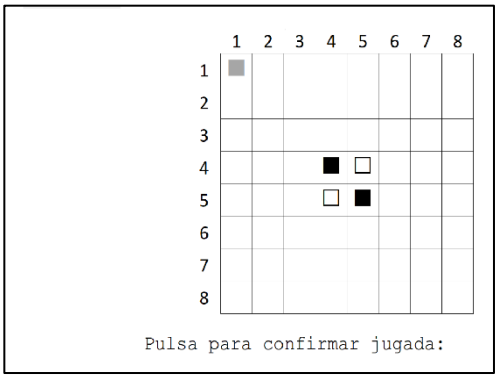
1. INICIO_FILA: Indica que a continuación seleccionaremos una fila ya que en el 7 segmentos se muestra la letra F, al pulsar el botón derecho no se realizará ninguna acción y al pulsar el botón izquierdo se pasará al estado SELECCIONAR_FILA.
2. SELECCIONAR_FILA: En este estado mostraremos el número de fila que, al pulsar el botón derecho, confirmaremos para la jugada. Al pulsar el botón izquierdo incrementaremos en una unidad el valor del 7 segmentos y si lo mantenemos pulsado se incrementará automáticamente.
3. INICIO_COLUMNA: Similar al estado INICIO_FILA salvo por el hecho que en el 7 segmentos se muestra una F en lugar de una C y al pulsar el botón derecho nos lleva al estado SELECCINAR_COLUMNA.
4. SELECCIONAR_COLUMNA: Se muestra el numero de columna que tendrá nuestra jugada y lo confirmaremos cuando pulsemos el botón derecho, al pulsarlo se realizará la jugada con la fila y columna seleccionada y volveremos al estado INICIO_FILA para realizar una nueva jugada.

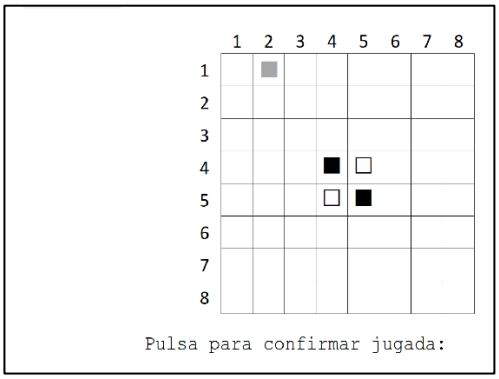
```
void evento_botones_producido(int evento){
  //Esperando a seleccionar fila
  switch(estado_jugada_botones){
    case INICIO_FILA:
      D8Led_symbol(15);
      if(evento == EVENTO_IZ_BOTON_SOLTADO){
        estado_jugada_botones = SELECCIONAR_FILA;
        contador = 1;
        D8Led_symbol(contador);
      }
      break;
    case SELECCIONAR_FILA:
      if(evento == EVENTO_IZ_BOTON_SOLTADO){
        contador ++;
        if(contador == 9)
          contador = 1;
        D8Led_symbol(contador);
      }else if(evento == EVENTO_DR_BOTON_SOLTADO){
        fila_botones = contador;
        contador = 1;
        estado_jugada_botones = INICIO_COLUMNA;
        D8Led_symbol(12);
      }
      break;
    //Esperando a seleccionar columna
    case INICIO_COLUMNA:
      if(evento == EVENTO_IZ_BOTON_SOLTADO){
        estado_jugada_botones = SELECCIONAR_COLUMNA;
        contador = 1;
        D8Led_symbol(contador);
      }
      break;
    case SELECCIONAR_COLUMNA:
      if(evento == EVENTO_IZ_BOTON_SOLTADO){
        contador ++;
        if(contador == 9)
          contador = 1;
        D8Led_symbol(contador);
      }else if(evento == EVENTO_DR_BOTON_SOLTADO){
        columna_botones = contador;
        contador = 1;

        estado_jugada_botones = INICIO_FILA;
        jugar(fila_botones - 1,columna_botones - 1);
        D8Led_symbol(15);
      }
      break;
  }
```

-Jugabilidad por pantalla

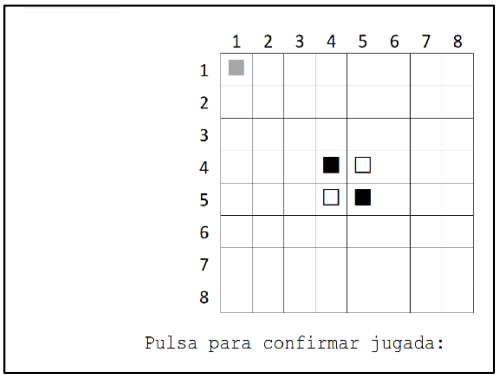
Ya en la práctica 3 se introduce una nueva forma de interacción, la pantalla y el touchpad, lo que cambia la forma en la que introducimos fila y columna. Se ha diseñado un nuevo autómata que muestra las fichas en el tablero como se ha mostrado en el apartado X por lo tanto la jugabilidad se basa en la pulsación de la tecla izquierda para desplazarnos horizontalmente y la tecla derecha para desplazarnos.

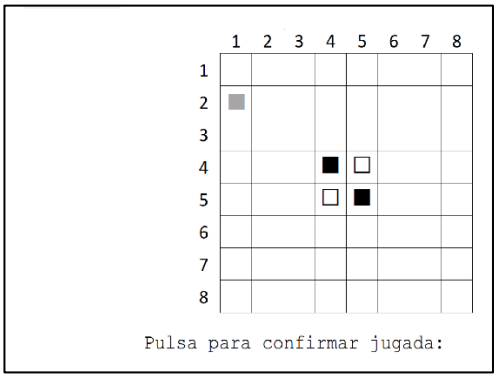
Al pulsar el botón izquierdo:



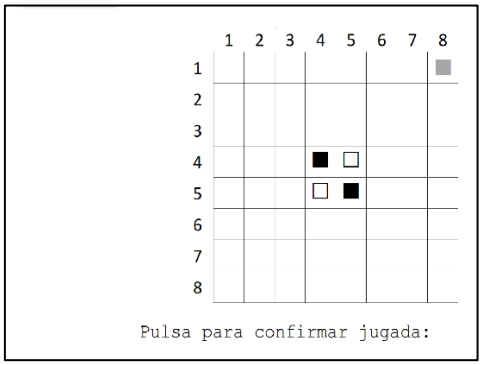


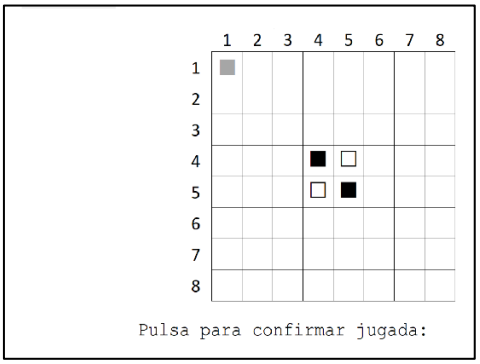
Al pulsar el botón derecho:





En caso de encontrarnos en el limite derecho o inferior del tablero saltaríamos al inicio de la fila o columna:





Mejoras y correcciones de la práctica 1

Resultados
Conclusiones