

Proyecto hardware

Práctica 3: DISEÑO DE UNA PLATAFORMA INDEPENDIENTE

Turno de Mañanas. Grupo G

Hecho por

JAVIER JULVE YUBERO 840710
ALEJANDRO BENEDÍ ANDRÉS 843826

ÍNDICE

1-Resumen.....	3
2-Introducción.....	4
3-Objetivos.....	5
4-Metodología.....	6
4.1 Diseño y esquema del proyecto.....	6
Temporizadores y gestor de alarmas.....	6
Cola FIFO y planificador.....	7
Línea serie y gestión de la entrada/salida.....	8
Reducción del consumo de energía del sistema.....	8
Watchdog.....	8
Llamadas al sistema (SWI).....	9
Máquina de estados del juego.....	9
4.2 Código fuente.....	10
4.3 Horas dedicadas.....	10
5-Resultados.....	11
5.1 Resultados del juego final.....	11
5.2 Problemas encontrados.....	15
Envío de strings por la uart.....	15
Problemas en el intérprete de comandos.....	15
6-Conclusiones.....	16
7-Anexo.....	17
7.1 Código.....	17
7.2 Máquina de estados.....	26
7.3 Modificaciones realizadas respecto del código entregado anteriormente.....	27

1-Resumen

Este proyecto ha consistido en tomar el anterior del juego del conecta K y habilitarle un entorno de juego totalmente nuevo para que el usuario pueda hacer uso de periféricos y nuevas funcionalidades.

Los periféricos mencionados consisten en temporizadores que midan el tiempo, botones para desencadenar nuevos eventos como rendirse o cancelar, un intérprete que usará los comandos introducidos por el jugador y que se le especifican previamente y hasta una pantalla donde ver los caracteres introducidos y el tablero del juego de forma gráfica.

El código no mezcla los distintos módulos ni periféricos entre sí, lo cual hace muy sencilla la inclusión, modificación o eliminación de alguno de ellos. Para ello existe un planificador que se encarga de recibir las señales que mandan los distintos periféricos, que hemos denominado eventos para así realizar las gestiones correspondientes. Estos eventos se almacenan en una cola FIFO circular y el planificador siempre que no haya mandado gestionar algo y pueda, extraerá ese evento para así tratarlo como se deba. Digo que el planificador pueda, ya que habrá veces que no podrá ya que se ha incluido un modo de bajo consumo que se activa a los 12 segundos de inactividad y lo que hace es “apagar” el procesador hasta que llegue una interrupción, que en este caso será usando los botones. Pero estos botones no se limitan solo a eso, sino también a rendirse en el juego, ósea el EINT2 y pin 15 del GPIO o cancelar una jugada antes de los 3 segundos de haberla introducido, con el EINT1 y pin 14 del GPIO.

El control del tiempo a su vez se mide con los periféricos timer y además se han creado las alarmas, un concepto de este proyecto que permite establecer un tiempo límite para alguna acción y en caso de que se cumpla la acción en ese límite se desactive y se haga su gestión y en caso contrario, se hará otra cosa y se invalidará la acción que podía realizar el dispositivo que la ha generado, ya que ya no está en el tiempo establecido.

Por último el periférico que queda, es decir, la UART0 tiene dos funciones, por un lado sirve de pantalla para el tablero y los caracteres introducidos y por el otro, se encarga de recibir caracteres, analizarlos y transferirlos como comandos para realizar acciones. Los comandos que poseemos son: \$NEW! que inicia una partida solo si no hay ninguna en curso, \$END! que cumple la misma acción que el botón 2, ósea la de rendirse, y el comando \$#-#! siendo el primer parámetro la fila y el segundo la columna donde se va a colocar la ficha.

Una aclaración final es que el sistema trabaja en modo usuario y modo superusuario, con el fin de poder realizar llamadas al sistema implementadas en SWI para poder habilitar y deshabilitar interrupciones cuando sea necesario.

2-Introducción

Estas dos prácticas se encargan de crear un entorno totalmente nuevo para que la interacción del usuario con el conecta K de la práctica 1 sea más clara y más fácil, además de tener nuevas características como comandos y botones.

Lo primero es aclarar que el código se ha segmentado en distintos módulos con el fin de que sea más fácil de entender y de modificar ya sea para hacer cambios rápidos o para añadir más funciones como ha sido el caso. Todo esto además se junta gracias a funciones callback que se comunican con un planificador que es el que gestiona todo lo que ocurre a lo largo del juego.

Además de lo anterior hay módulos que trabajan con periféricos de todo tipo, ya sean temporizadores, botones, una entrada y salida de texto, etc...

Por último se ha creado un intérprete de comandos para que en base a un texto de 3 caracteres entre \$ y ! desencadene distintas acciones. Además de una gestión de errores visual por leds y un modo de ahorro de energía que se activa al cumplirse 12 segundos de inactividad.

3-Objetivos

En esta práctica además de los objetivos obligatorios, nosotros nos pusimos algunos personales, los cuales se han cumplido de forma exitosa y ahora pasamos a mostrar cuales han sido.

- Desarrollo de diferentes módulos que permitan un código final de tipo modular.
- Desarrollo rutinas de tratamiento de interrupción (RSI) en C.
- Diseño e implementación de periféricos, algunos que van por tiempo (Timer 0, Timer 1, Watchdog), otros que van por placa (GPIO, UART) y otros que sean de entrada salida (EINT).
- Gestión E/S de dispositivos en C.
- Uso y aprovechamiento de forma eficiente de los modos de ejecución.
- Creación de un planificador que se encargue de gestionar los eventos generados por cada módulo de manera distinta.
- Creación de una cola FIFO para poder transferir los eventos o demás información entre módulos.
- Implementación de llamadas al sistema para gestionar las interrupciones.
- Identificación y resolución de las condiciones de carrera.

4-Metodología

4.1 Diseño y esquema del proyecto

Usando la lógica del juego del conecta K se ha desarrollado un código modular para así separar el juego previamente programado de los nuevos periféricos y que no haya dependencia entre ellos.

Para poder realizar esto se creó una cola FIFO de tipo circular que se encarga de almacenar un tipo de datos creados llamados eventos, establecidos en una cabecera aparte llamada eventos.h. De esta forma cuando se realiza una acción que debe ser gestionada, se almacenará el evento y el planificador lo extraerá para así diferenciarlo y realizar ciertas acciones bien diferenciadas, además se permite guardar un dato auxiliar por si se necesitase información adicional como por ejemplo un comando concreto ya que el evento es el mismo.

Gracias a esto ya podemos desarrollar nuevos módulos que usen el hardware del periférico que use para poder intervenir en el juego, (timers, rtc, wd, gpio, eint1 y 2, uart0) y para su uso se han creado dos tipos de módulos, primero el de tipo hal, que se encarga de recibir o mandar los datos del hardware y el segundo es el de tipo drv o driver que se encarga de una gestión del periférico en base a los datos del hal a un nivel más alto. También hay un par de módulos que se encargan de gestionar el modo de ejecución del procesador para así poder usar modos de ahorro de energía.

Cada módulo está con un formato de nombre que deja claro que es aquello con lo que trabaja y además poseen cabecera propia, separándolos así en .c y .h

El módulo del conecta K también sigue esta metodología, aunque cabe aclarar que se ha hecho alguna modificación para poder adaptarlo a las nuevas funcionalidades.

Finalmente también se ha creado un módulo para crear alarmas que en resumen se encargan de establecer un tiempo límite para que en caso de que se active o no al cumplirse el tiempo establecido, lo cual nos da un control muy bueno sobre los módulos en base al tiempo.

Temporizadores y gestor de alarmas

Se hacen usos de 2 temporizadores diferentes. El primero es el encargado de obtener tiempos precisos que nos ayudan a implementar otros módulos. Por otro lado, el temporizador 1 se utiliza principalmente para comprobar qué alarmas hay que lanzar. Cada 1 us se produce una interrupción del temporizador 1 el cual encola el evento referente a comprobar las alarmas.

Las alarmas son utilizadas por varios módulos de forma simultánea. Es la única forma que se tiene para controlar intervalos de tiempo. Al iniciar una alarma es necesario introducir un evento y será este el que será encolado cuando la alarma salte. Las alarmas pueden ser

programadas para que se activen de forma periódica o que tan solo se ejecutan una vez. Si tan solo se ejecuta una vez, se desactiva dicha alarma y ese hueco queda libre para que se pueda introducir otra. Adicionalmente, cualquier alarma puede ser desactivada en cualquier momento.

Cola FIFO y planificador

La cola FIFO se ha creado con el fin de que haya comunicación entre los distintos módulos, de la forma en que funciona es que mediante funciones callback en los módulos se logra encolar el evento que se genere tras un suceso concreto, existe la opción de dar un dato auxiliar al encolar, como ya se ha indicado anteriormente. La cola es circular dado a que al extraer un evento no se elimina de la cola sino que avanza en el círculo y si se llega de nuevo a esa posición, se sustituirá por otro. La parte de extraer la usa sobretodo el planificador, ya que como siempre está en ejecución trata de extraer algún evento de la cola y en función de cual es se invoca a algún módulo, en caso de que la cola esté vacía, se invocará el modo IDLE hasta que vuelva a haber algo para extraer.

En la cola FIFO, al extraer o insertar eventos se podrían producir condiciones de carrera por eso mismo ha sido necesario, tanto al extraer como al insertar, desactivar las interrupciones, si estaban activas. Posteriormente se realiza la operación deseada para extraer o insertar y una vez terminado se restablece el I bit del CPSR a como estaba antes.

Interrupciones externas con botones

Se han creado 2 interrupciones externas de EINT 1 y EINT2 para simular el funcionamiento de 2 botones, estos botones se activan desde el GPIO y los pines 14 y 15.

Dado que las interrupciones son muy rápidas, a la hora de gestionar el botón se deshabilita su interrupción desde el VIC y se pone una alarma periódica para verificar si sigue pulsado, ya que el dejar de pulsarlo no genera interrupciones y es importante que al dejar de pulsarlo se vuelvan a activar las interrupciones ya que sino su función sólo podría ser usada una sola vez.

Por tanto cuando se deje de pulsar y se invoque la alarma verificará mediante unas flags de pulsado se cambiará su estado y al volverlo a pulsar podrá decir que ha vuelto a ser pulsado y que se debe tratar la interrupción.

En el ámbito del juego el botón 1 se encarga de cancelar el movimiento realizado antes de que se cumpla el tiempo de la alarma para poner la ficha, el botón 2 en cambio sirve para rendirse. Además estos botones permiten volver del Power Down y además no afecta a la partida actual hasta que se ponga todo en funcionamiento de nuevo.

Línea serie y gestión de la entrada/salida

Para las demás E/S se usa tanto el GPIO como la línea serie. Dentro del GPIO se usan 3 LEDs indicados en el io_reserva.h que sirve para indicar cuando algo va mal si se enciende ese (overflow en la FIFO, comando inválido o jugada inválida).

La línea serie hace uso de la UART0, en el modo de entrada simplemente capta el carácter introducido y luego si se cumplen ciertas condiciones dentro de su máquina de estados, se envía al planificador un evento junto al comando introducido. Por el lado de la salida, se imprimen los caracteres introducidos de la forma más rápida posible, además de servir como pantalla para mostrar el tablero, jugadas y normas del juego, además de los resultados de la partida.

Reducción del consumo de energía del sistema

Para reducir el consumo de energía se han utilizado principalmente 2 modos: modo “idle” y modo “Power Down” .

El modo “Idle”, reduce parcialmente el consumo de energía ya que aunque el procesador detenga la ejecución del programa, los “timers” y por tanto las interrupciones seguían funcionando. Para salir de este modo de ahorro de energía, es necesario realizar una interrupción.

Al modo “Idle” se entra cuando no hay nada que extraer de la cola de eventos. Además también se activará una alarma con el tiempo máximo en este modo, si esta alarma salta se encola un evento que activará el modo “Power Down”

El modo “Power Down”, es muy parecido al modo “Idle”, pero en este modo los periféricos también entran en modo de bajo consumo y dejan de funcionar. La única forma de despertar de este modo es al pulsar un botón. Por ello antes de entrar al modo es necesario configurar el “EXTWAKE” para que los 2 botones habilitados puedan despertar al procesador.

Al entrar a este modo, la frecuencia del reloj disminuye y al despertarse no se restablece la frecuencia estándar del reloj. Por eso mismo es necesario restablecer el PLL a como estaba antes de entrar al modo. Si no se hace esto, los “timers” dejan de funcionar correctamente ya que estos dependen del PLL.

Watchdog

Se tiene que alimentar al watchdog en el bucle infinito del planificador, ya que es este el que se encarga de extraer todos los eventos. De esta forma cada vez que extraiga se alimentara y no se resetee la ejecución. Si el programa entrará en cualquier otro bucle infinito el watchdog reseteara la ejecución.

Al alimentar el watchdog es necesario que se produzca sin que se produzca una interrupción en el proceso. Por eso mismo es necesario comprobar si las interrupciones están activadas o desactivadas. Si están activas se desactivan para poder realizar el “Feed” de una forma adecuada”. Una vez realizado es necesario restablecer el I bit (bit del CPSR que activa las interrupciones) a como estaba antes de de realizar el “Feed”

Llamadas al sistema (SWI)

Este módulo tiene el objetivo de interactuar con el sistemas.

Las operaciones relacionadas con las irq se han utilizado principalmente para eliminar las condiciones de carrera y para alimentar el watchdog. Mientras que la función clock_get_us nos permite leer el tiempo transcurrido por el temporizador 0.

Para poder implementar las funciones referentes a las IRQ y FIQ fue necesario implementarlo en ensamblador ya que desde C no se podía modificar el bit I del CPSR

Máquina de estados del juego

Para asegurarnos que el juego sigue su ejecución correcta, se ha creado e implementado una máquina de estados..

Esta máquina tiene en cuenta el estado en que se encuentra y la entrada que se le da en función del tipo de la misma, ya sea oprimiendo un botón o introduciendo un comando entre otras acciones, se tiene en cuenta además cada posible acción que puede tener el jugador para así poder saltar a otro estado o permanecer en el mismo.

Esta misma máquina puede verse en el ANEXO I

4.2 Código fuente

El código usado puede ser consultado en el ANEXO II de este documento.
Para verlo con más detalle acudir a los ficheros de código de C y ASM

4.3 Horas dedicadas

	INTEGRANTES	
TAREAS	Javier Julve	Alejandro Benedí
Diseño de la cola de eventos	4 horas	4 horas
Uso de temporizadores	1 hora	4 horas
Gestor Alarmas	30 minutos	3 horas
GPIO	1 hora y 30 minutos	45 minutos
Interrupciones externas	1 hora y 30 minutos	30 minutos
Modos de ahorro de energía	30 minutos	1 hora y 30 minutos
E/S en el juego	6 horas	2 hora
RTC y watdog	1 hora	30 minutos
Condiciones de carrera	1 hora y 30 minutos	3 horas
Línea serie	4 horas	1 hora y 30 minutos
Máquina de estados del juego	2 horas	1 hora
Depuración	10 horas	10 horas
Llamadas al sistema (SWI)	30 minutos	2 hora y 30 minutos
Memoria	5 horas	2 hora y 30 minutos
Total	39 horas	35 horas y 45 minutos

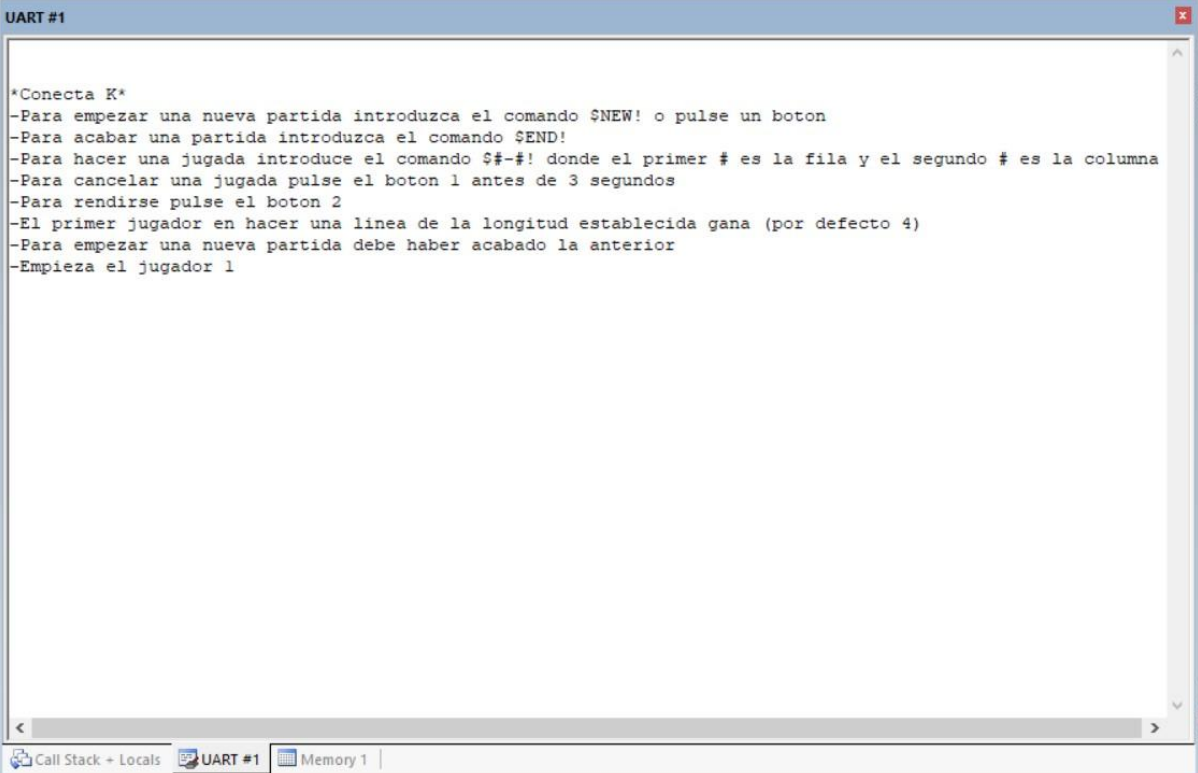
Figura 1. Horas dedicadas por integrante

5-Resultados

5.1 Resultados del juego final

A continuación presentamos algunas capturas realizadas a la uart0 (aunque en keil se llama UART #1) qué es la salida de nuestro juego, osea lo que actúa como pantalla del mismo.

Inicio de la ejecución del programa sin realizar ninguna acción:



```
UART #1

*Conecta K*
-Para empezar una nueva partida introduzca el comando $NEW! o pulse un boton
-Para acabar una partida introduzca el comando $END!
-Para hacer una jugada introduce el comando $#-#! donde el primer # es la fila y el segundo # es la columna
-Para cancelar una jugada pulse el boton 1 antes de 3 segundos
-Para rendirse pulse el boton 2
-El primer jugador en hacer una linea de la longitud establecida gana (por defecto 4)
-Para empezar una nueva partida debe haber acabado la anterior
-Empezar el jugador 1
```

Figura 2. Imagen con texto inicial

Tras pulsar uno de los botones o poner el comando \$NEW! se generará un nuevo tablero vacío, en el caso de ejemplo se ha usado el comando \$NEW!:

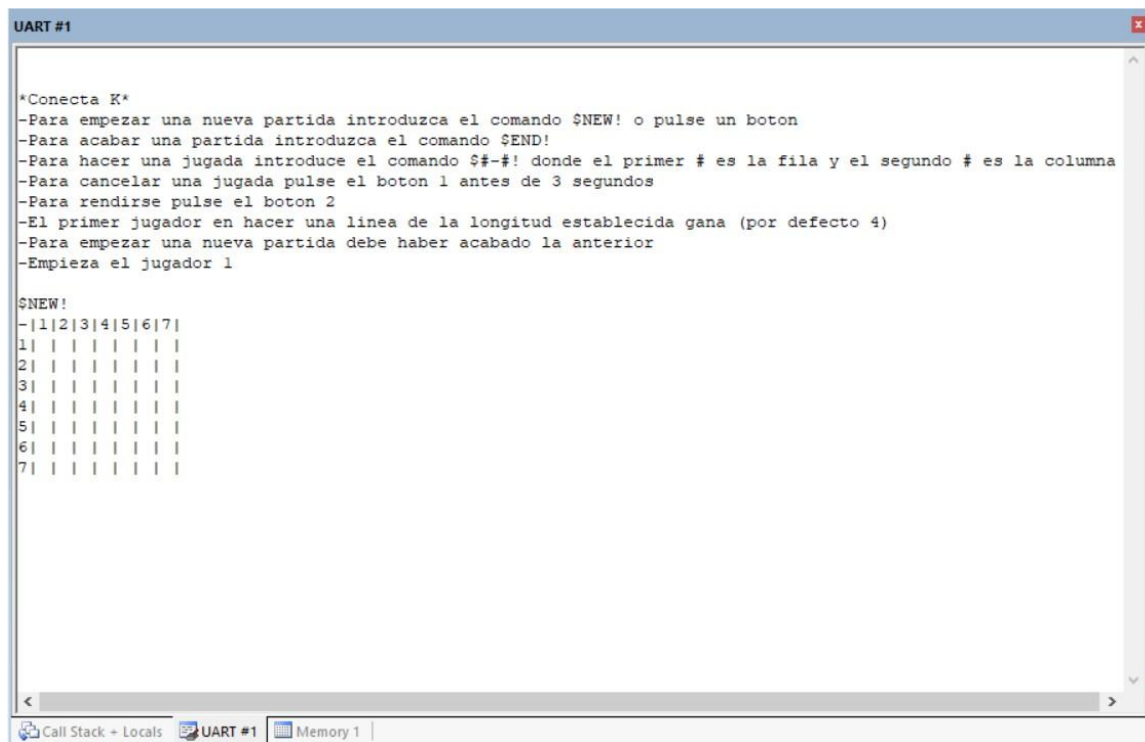


Figura 3: Imagen que muestra el tablero vacío

Tras esto es el turno del jugador 1 ya que como no ha habido ninguna partida anterior, empieza el por defecto, dicho jugador coloca una ficha en la posición 4-4, primero viendo la previsualización y posteriormente tras cumplirse los 3 segundos especificados en las instrucciones, se coloca la ficha realmente:

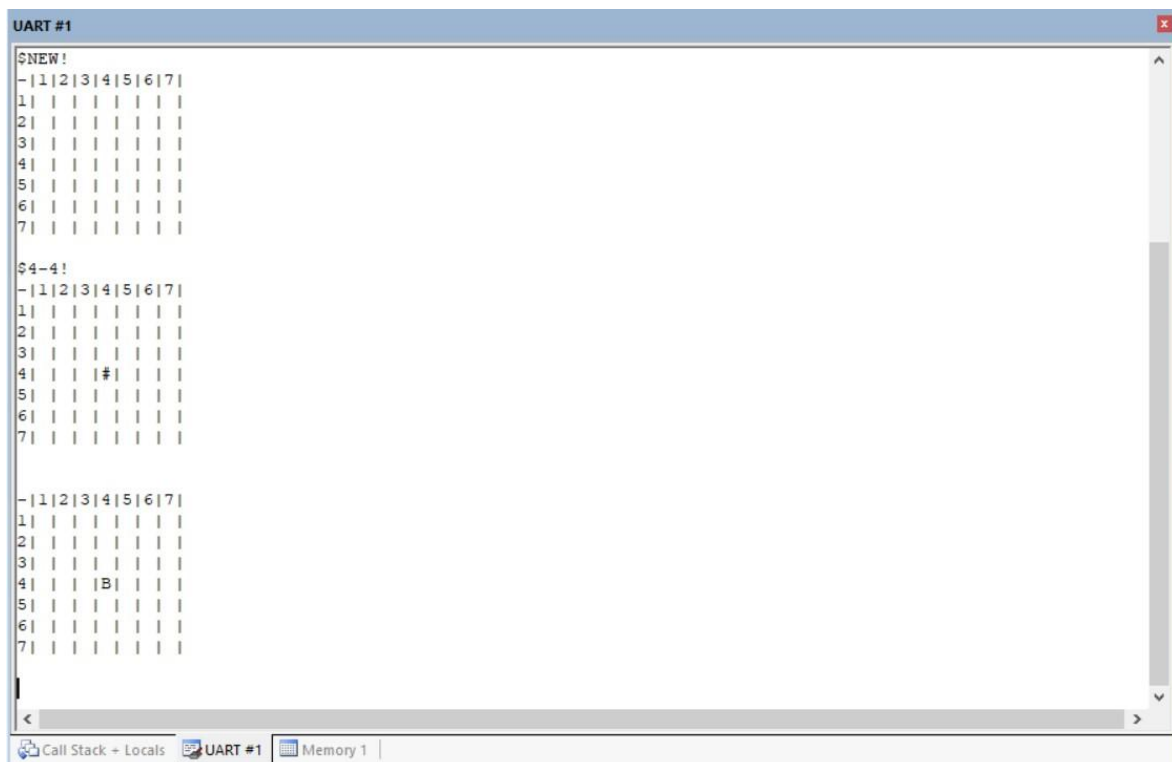


Figura 4: Imagen que muestra la colocación de una pieza

Ahora el otro jugador, intenta explotar el juego poniendo una ficha en la posición 8-8, sin embargo el juego lo impide y le indica el error:

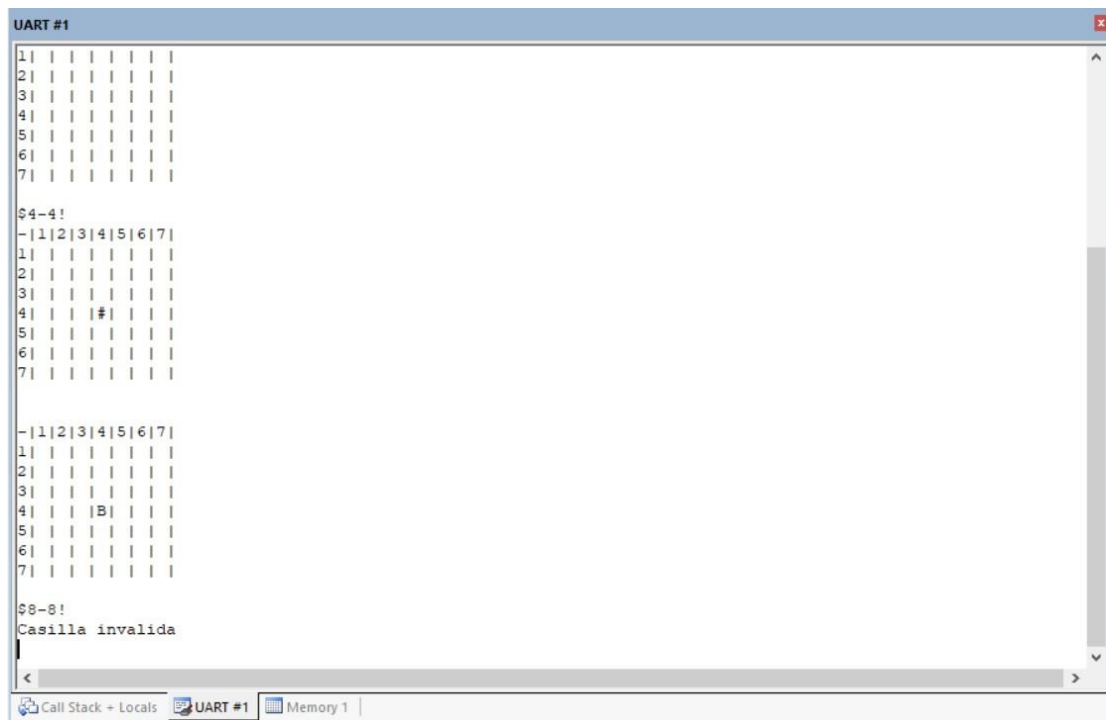


Figura 5: Imagen que muestra el error al introducir un ficha en una casilla donde hay otra.

Ahora el jugador 2 decide poner una ficha en una posición válida, sin embargo se arrepiente y por medio del botón 1 cancela su jugada tras ver la previsualización, mostrando al final el tablero como estaba pero sin pasar el turno, ya que el jugador aún no ha hecho su jugada :

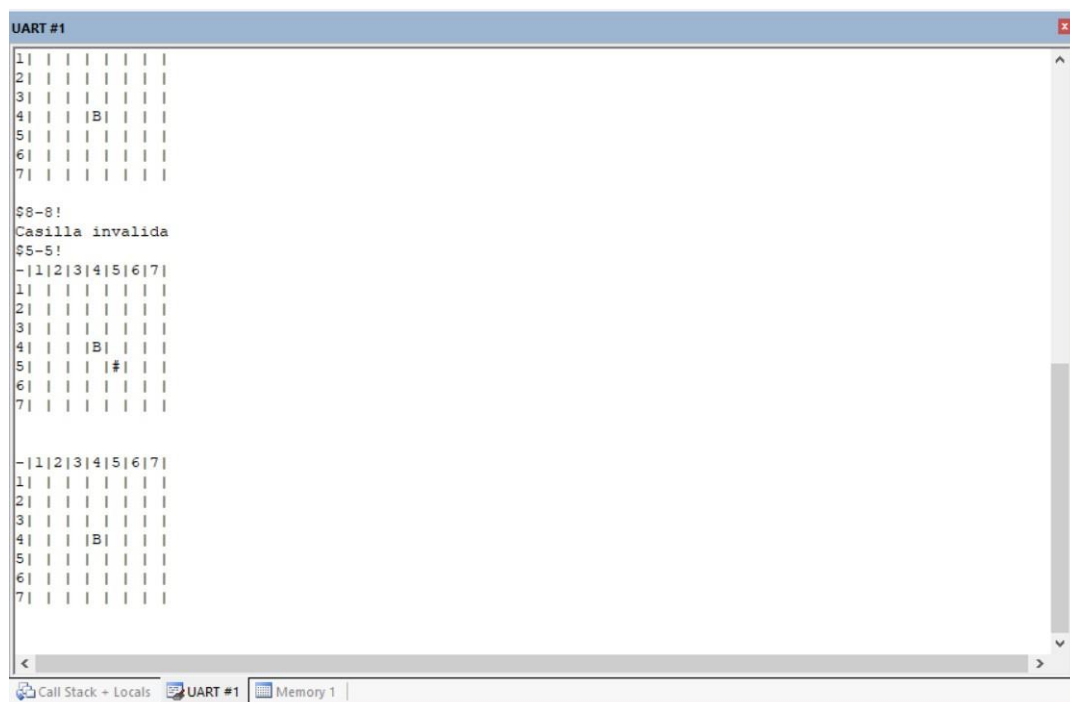
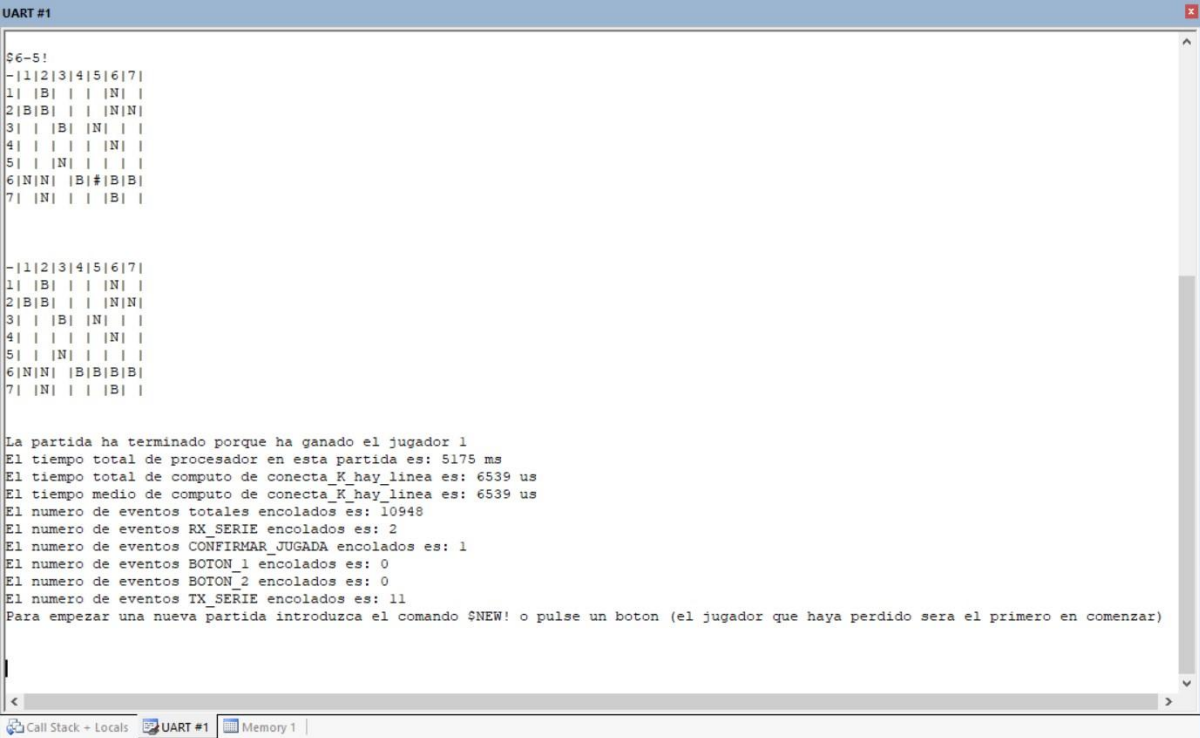


Figura 6: Imagen que muestra como deshacer un movimiento

Finalmente y tras una larga partida el jugador 1 logra ganar conectando sus fichas y salen los resultados, especificando que en este caso la partida finalizó por esta razón y especificando que el jugador que haya perdido es el que empieza (para realizar esta captura se ha usado un tablero de pruebas que teníamos, es por eso que los números de eventos son tan bajos, ya que este tablero estaba preparado para que el jugador 1 gane con un solo movimiento):



```
UART #1
$6-5!
-|1|2|3|4|5|6|7|
1| |B| | | |N| |
2|B|B| | | |N|N|
3| | |B| |N| | |
4| | | | | |N| |
5| | |N| | | | |
6|N|N| |B|#|B|B|
7| |N| | | |B| |

-|1|2|3|4|5|6|7|
1| |B| | | |N| |
2|B|B| | | |N|N|
3| | |B| |N| | |
4| | | | | |N| |
5| | |N| | | | |
6|N|N| |B|B|B|B|
7| |N| | | |B| |

La partida ha terminado porque ha ganado el jugador 1
El tiempo total de procesador en esta partida es: 5175 ms
El tiempo total de computo de conecta_K_hay_linea es: 6539 us
El tiempo medio de computo de conecta_K_hay_linea es: 6539 us
El numero de eventos totales encolados es: 10948
El numero de eventos RX_SERIE encolados es: 2
El numero de eventos CONFIRMAR JUGADA encolados es: 1
El numero de eventos BOTON_1 encolados es: 0
El numero de eventos BOTON_2 encolados es: 0
El numero de eventos TX_SERIE encolados es: 11
Para empezar una nueva partida introduzca el comando $NEW! o pulse un boton (el jugador que haya perdido sera el primero en comenzar)
```

Figura 7: Imagen que muestra las estadísticas finales al terminar un juego

5.2 Problemas encontrados

Envío de strings por la uart

Dado que se necesita mandar los comandos al planificador y a la vez imprimirlo por pantalla, era bastante complicado hacer ambas a la vez, con lo que simplemente optamos por ir almacenando todo en un buffer inventado mientras se iba imprimiendo las cosas y al llegar al final de la máquina es decir al mandar ! se verifica si el buffer de 3 está lleno y si lo está como cada letra se ha guardado en una posición, las almacenamos en el valor auxiliar desplazando la 1ª 16 bits, la 2ª 8 bits y la última sin desplazar, haciendo esto se lograba mandar todo a la vez y luego se volvía a trocear para interpretar el comando que era.

Problemas en el intérprete de comandos

Tras superar el problema anterior nos encontramos una dificultad que era interpretar los comandos que nos llegaban ya que habían pasado de char a uint32, con lo que la interpretación de c era distinta, para solucionar esto simplemente creamos una variable que junte un comando de la misma forma que el auxData enviado para así comprarlo fácilmente, en cuanto a un comando de una jugada, se sigue una estructura así: (uint8_t)((auxData >> 16) & 0xFF) - '0'; para guardarlo en una variable, es decir se establece una conversión forzada a uint8 del auxData desplazado tanto como se necesite para es caracter concreto y luego se hace una AND de los dos últimos bits y suprimiendo además el carácter de fin de línea.

6-Conclusiones

Tras haber expuesto todo en el documento y basándonos en la experiencia a la hora de trabajar creo que podemos tener claro que hemos cumplido todos los objetivos marcados a tiempo.

Además también ha servido bastante para familiarizarnos con conceptos como la modularidad en programación o el uso de funciones callback o el concepto de las condiciones de carrera, teniendo en cuenta también que hemos aprendido bastante sobre el funcionamiento de los diferentes periféricos y su comunicación con el procesador. La modularidad además ha hecho que entendamos ciertas decisiones de diseño a la hora de hacer código y del por qué se elige hacer esto ya que permite una abstracción menor del bajo nivel.

Por otro lado la idea del proyecto nos ha parecido bastante interesante ya que desde una idea muy básica como era el código de la práctica 1 hemos logrado hacer una mini consola funcional con este juego, lo cual nos parece una idea bastante innovadora e interesante frente al resto de proyectos que hemos realizado hasta la fecha.

Queremos destacar que a pesar de que sabemos que hay margen de mejora por ciertos problemas que hayamos podido tener o por lo pulido que esté el resultado final, estamos bastante contentos con el trabajo realizado ya que era un proyecto bastante complejo que requería mirar y entender la documentación y fijarse en los pequeños detalles ya que marcaban la diferencia. Pese a esto consideramos que el trabajo final está bastante completo y a la altura de lo que se pedía con lo que una nota global que creemos que deberíamos tener es un 7 o al menos con la que nos calificamos a nosotros mismos.

7-Anexo

7.1 Código

Juego (máquina de estados implementada):

```
void juego_tratar_evento(EVENTO_T ID_evento, uint32_t auxData){
    if (ID_evento == ev_RX_SERIE && continuable == 1){
        char buffer_comandos[3] = {'N','E','W'};
        uint32_t comando;
        comando = (buffer_comandos[0] << 16) | (buffer_comandos[1] << 8) | (buffer_comandos[2]);

        if(auxData == comando && partida_iniciada == 0){
            inicializar_partida();
            return;
        }
        if (partida_iniciada == 1){
            buffer_comandos[0] = 'E';
            buffer_comandos[1] = 'N';
            buffer_comandos[2] = 'D';
            comando = (buffer_comandos[0] << 16) | (buffer_comandos[1] << 8) |
(buffer_comandos[2]);
            tiempo_humano = tiempo_humano + (clock_getus() - intervalo);
            if (auxData == comando){
                gpio_hal_escribir(pin_serie_error, numero_pines_serie_error, 0);
                mostrar_estadisticas(END);
                return;
            }
            else{
                uint8_t guion = (uint8_t)((auxData >> 8) & 0xFF);

                if (guion == '-'){
                    entrada[1] = (uint8_t)((auxData >> 16) & 0xFF) - '0';
                    entrada[2] = (uint8_t)(auxData & 0xFF) - '0';
                    entrada[3] = turno;

                    entrada_leer(entrada, &fila, &columna, &color);
                    if(tablero_fila_valida(fila) && tablero_columna_valida(columna) &&
tablero_color_valido(color)){
                        if (celda_vacia(tablero_leer_celda(&cuadricula, fila,
columna))){
                            continuable = 0;
                            tiempo_excato = clock_getus();
                            conecta_K_previsualizar_tablero(&cuadricula,
vista_previa, fila, columna);

                            tiempo_excato = clock_getus();
                            tiempo_computo_conecta_K =
tiempo_computo_conecta_K + tiempo_excato;

                            linea_serie_drv_enviar_array(vista_previa);
                            cancelable = 1;
                        }
                    }
                }
            }
        }
    }
}
```

[illegible]

```

        conecta_K_visualizar_tablero_final(&cuadricula, buffer_pantalla);
        tiempo_computo_conecta_K = tiempo_computo_conecta_K + (clock_getus() -
tiempo_excato);
        linea_serie_drv_enviar_array(buffer_pantalla);
    }
    else if (ID_evento == ev_BOTON_1 && partida_iniciada == 0){
        inicializar_partida();
    }
    else if (ID_evento == ev_BOTON_2 && continuable == 1){
        if (partida_iniciada == 0){
            inicializar_partida();
            return;
        }

        if (auxData == 1){
            return;
        }
        tiempo_humano = tiempo_humano + (clock_getus() - intervalo);
        mostrar_estadisticas(BOTON_RENDIRSE);
        //Sacar stats
    }
    else if (ID_evento == ev_TX_SERIE){
        continuable = 1;
    }
}

```

Gestor alarmas:

```

static Alarma lista_alarmas[ALARMAS_MAX];
static void (*FIFO_encolar)(); // Funcion callback de FIFO_encolar
static GPIO_HAL_PIN_T overflow;

void alarma_inicializar(GPIO_HAL_PIN_T pin_overflow, void (*funcion_callback_1)()){
    uint8_t i;
    num_activas = 0;
    for (i = 0; i < ALARMAS_MAX; i++){
        lista_alarmas[i].activa = 0;
    }
    FIFO_encolar = funcion_callback_1;
    overflow = pin_overflow;
}

void alarma_activar(EVENTO_T ID_evento, uint32_t retardo, uint32_t auxData){

    uint32_t mascara = 0x80000000;
    Alarma nueva_alarma;
    uint8_t i, activada;

    if (retardo == 0){ // Eliminar alarma
        desactivar_alarma(ID_evento);
    }
}

```

```

else if (num_activas >= ALARMAS_MAX) {      // Desbordamiento de alarmas
    FIFO_encolar(ALARMA_OVERFLOW, 0);
}
else { // Activar alarma
    nueva_alarma.ID_evento = ID_evento;
    nueva_alarma.auxData = auxData;
    nueva_alarma.activa = 1;

    // Comprobamos si es periodica
    if ((retardo & mascara) == mascara){
        nueva_alarma.periodica = 1;
        nueva_alarma.periodo = retardo ^ mascara;
    }
    else {
        nueva_alarma.periodica = 0;
        nueva_alarma.periodo = retardo;
    }

    // Añadimos a la nueva alarma a la lista de alarmas
    i = 0;
    activada = 0;
    while ( i < ALARMAS_MAX && activada == 0) {
        if (lista_alarmas[i].activa == 0){
            lista_alarmas[i] = nueva_alarma;
            num_activas++;
            activada = 1;
        }
        i++;
    }
}
}
}

```

```

void comprobar_alarmas(unsigned int tiempo){
    uint8_t i, esPeriodica, estaActiva;
    uint32_t retardo = tiempo;
    for (i = 0; i < ALARMAS_MAX; i++){
        estaActiva = lista_alarmas[i].activa;
        esPeriodica = lista_alarmas[i].periodica;
        retardo = lista_alarmas[i].periodo;
        if (estaActiva && ((tiempo % retardo) == 0)){
            FIFO_encolar(lista_alarmas[i].ID_evento, 0);
            lista_alarmas[i].periodo = retardo;
            if (esPeriodica == 0){
                lista_alarmas[i].activa = 0;
                num_activas--;
            }
        }
    }
}
}

```

```

void desactivar_alarma(EVENTO_T ID_evento){
    uint8_t i = 0, encontrado = 0;

```

```

while (i < ALARMAS_MAX && encontrado == 0){
    if (lista_alarmas[i].ID_evento == ID_evento){
        lista_alarmas[i].activa = 0;
        num_activas--;
        encontrado = 1;
    }
    i++;
}

}

void alarma_overflow(void){
    gpio_hal_escribir(overflow, 1, 1);
    while(1);
}

void alarma_tratar_evento(unsigned int tiempo){
    comprobar_alarmas(tiempo);
}

```

Módulos driver:

Temporizador:

```
static void (*FIFO_encolar)();
static EVENTO_T evento;

void temporizador_drv_iniciar(void){
    temporizador_hal_iniciar();
}

void temporizador_drv_empezar(void){
    temporizador_hal_empezar();
}

uint64_t temporizador_drv_leer(void){
    return temporizador_hal_leer() * temporizador_hal_ticks2us;
}

uint64_t temporizador_drv_parar(void){
    return temporizador_hal_parar() * temporizador_hal_ticks2us;
}

// Funcion para crear una alarma.
void temporizador_drv_reloj (uint32_t periodo, void
(*funcion_encolar_evento)(), EVENTO_T ID_evento){
    FIFO_encolar = funcion_encolar_evento;
    evento = ID_evento;
    temporizador_hal_reloj (periodo, funcion_callback_temporizador);
}

void funcion_callback_temporizador(uint32_t tiempo) {
    FIFO_encolar(evento, tiempo);
}

uint64_t __swi(0) clock_getus(void);
uint64_t __SWI_0 (void) {
    return temporizador_hal_leer() / temporizador_hal_ticks2us;
}
```

Botones:

```
static unsigned int estado_pulsacion_1 = NO_PULSADO;
static unsigned int estado_pulsacion_2 = NO_PULSADO;

static void (*FIFO_encolar)();

void inicializar_botones(void (*funcion_encolar_evento)(), uint32_t pr){
    eint1_init();
    eint2_init();
    alarma_activar(ALARMA_BOTONES, 0x80000064, 0);
    FIFO_encolar = funcion_encolar_evento;
}
```

```

}

void clear_nueva_pulsacion_1(void){
    eint1_clear_nueva_pulsacion();
}

unsigned int nueva_pulsacion_1(void){
    unsigned int new;
    new = eint1_read_nueva_pulsacion();
    eint1_clear_nueva_pulsacion(); // Las pulsaciones sólo deben procesarse una vez. Por tanto
    se pone a 0 después de leerlo
    return new;
}

void actualizar_estado_1(void){
    EXTINT = EXTINT | (1<<1);    // clear interrupt flag de EINT1
    if ((EXTINT & (1<<1)) != 0){ // si el botón está pulsado, la instrucción de arriba no hará nada y
EXTINT valdrá 1. Si el botón no está pulsado valdrá 0
        estado_pulsacion_1 = PULSADO;
    }
    else{
        estado_pulsacion_1 = NO_PULSADO;
        // si no está pulsado se habilitan las interrupciones (antes ya se ha limpiado el de
EXTINT)
        VICIntEnable = VICIntEnable | (1 << 15); // Enable EXTINT1 Interrupt (la interrupción
del botón se deshabilita a si misma, al terminar la pulsación hay ue volver a habilitarla)
    }
}

unsigned int leer_estado_1(void){
    return estado_pulsacion_1;
}

//EINT 2
void clear_nueva_pulsacion_2(void){
    eint2_clear_nueva_pulsacion();
}

unsigned int nueva_pulsacion_2(void){
    unsigned int new;
    new = eint2_read_nueva_pulsacion();
    eint2_clear_nueva_pulsacion(); // Las pulsaciones sólo deben procesarse una vez. Por tanto
    se pone a 0 después de leerlo
    return new;
}

void actualizar_estado_2(void){
    EXTINT = EXTINT | (1<<2);    // clear interrupt flag de EINT1
    if ((EXTINT & (1<<2)) != 0){ // si el botón está pulsado, la instrucción de arriba no hará nada y
EXTINT valdrá 1. Si el botón no está pulsado valdrá 0
        estado_pulsacion_2 = PULSADO;

```

```

    }
    else{
        estado_pulsacion_2 = NO_PULSADO;
        // si no está pulsado se habilitan las interrupciones (antes ya se ha limpiado el de
EXTINT)
        VICIntEnable = VICIntEnable | (1 << 16); // Enable EXTINT1 Interrupt (la interrupción
del botón se deshabilita a si misma, al terminar la pulsación hay ue volver a habilitarla)
    }
}

unsigned int leer_estado_2(void){
    return estado_pulsacion_2;
}

void gestion_boton(void){
    if(nueva_pulsacion_1() == 1){
        FIFO_encolar(ev_BOTON_1, 0);
    }
    if(nueva_pulsacion_2() == 1){
        FIFO_encolar(ev_BOTON_2, 0);
    }
}

void comprobar_botones(){
    actualizar_estado_1();
    if(leer_estado_1() == PULSADO){
        gestion_boton();
    }
    actualizar_estado_2();
    if(leer_estado_2() == PULSADO){
        gestion_boton();
    }
}

```

Leer línea:

```

static void (*FIFO_encolar)();
    // Callback para la funcion FIFO_encolar.h
static EVENTO_T evento;
static char buffer_comandos[MAX_BUFFER_SIZE]; // Buffer para almacenar los caracteres
recibidos
static int UART0_counter;
static int comando_valido;
static GPIO_HAL_PIN_T pin_serie_error;
static GPIO_HAL_PIN_T numero_pines_serie_error;

void linea_serie_drv_inicializar(GPIO_HAL_PIN_T pin_s, GPIO_HAL_PIN_T numero_pines_s, void
(*funcion_encolar_evento)(), EVENTO_T ev) {
    pin_serie_error = pin_s;
    UART0_counter = 0;
    comando_valido = 0;
}

```



```

    numero_pines_serie_error = numero_pines_s;
    FIFO_encolar = funcion_encolar_evento;
    evento = ev;
    gpio_hal_sentido(pin_s, numero_pines_s, GPIO_HAL_PIN_DIR_OUTPUT);
    linea_serie_hal_init(&funcion_callback, &funcion_callback_fifo_encolar);
}

void linea_serie_recibir_char(char c){
    if (comando_valido == 1){
        if(c == '!'){
            if (UART0_counter == 3) {
                uint32_t aux;
                aux = (buffer_comandos[0] << 16) | (buffer_comandos[1] <<
8) | (buffer_comandos[2]);
                FIFO_encolar(evento, aux);
            }
            else{
                gpio_hal_escribir(pin_serie_error, numero_pines_serie_error,
1);
            }
            UART0_counter = 0;
            buffer_comandos[0] = buffer_comandos[1] = buffer_comandos[2] = 0;
            comando_valido = 0;
            linea_serie_drv_enviar_array("!\\n");
            return;
        }
        else if (c != '$'){
            if(UART0_counter < 3){
                buffer_comandos[UART0_counter] = c;
                UART0_counter++;
            }
            else{
                gpio_hal_escribir(pin_serie_error, numero_pines_serie_error,
1);
                comando_valido = 0;
            }
        }
        else{
            linea_serie_drv_enviar_array("\\n");
        }
    }
    if (c == '$'){
        UART0_counter = 0;
        comando_valido = 1;
        gpio_hal_escribir(pin_serie_error, numero_pines_serie_error, 0);
    }
    linea_serie_drv_enviar_array(&c);
    if(comando_valido == 0){
        gpio_hal_escribir(pin_serie_error, numero_pines_serie_error, 1);
        linea_serie_drv_enviar_array("\\n");
    }
}

```

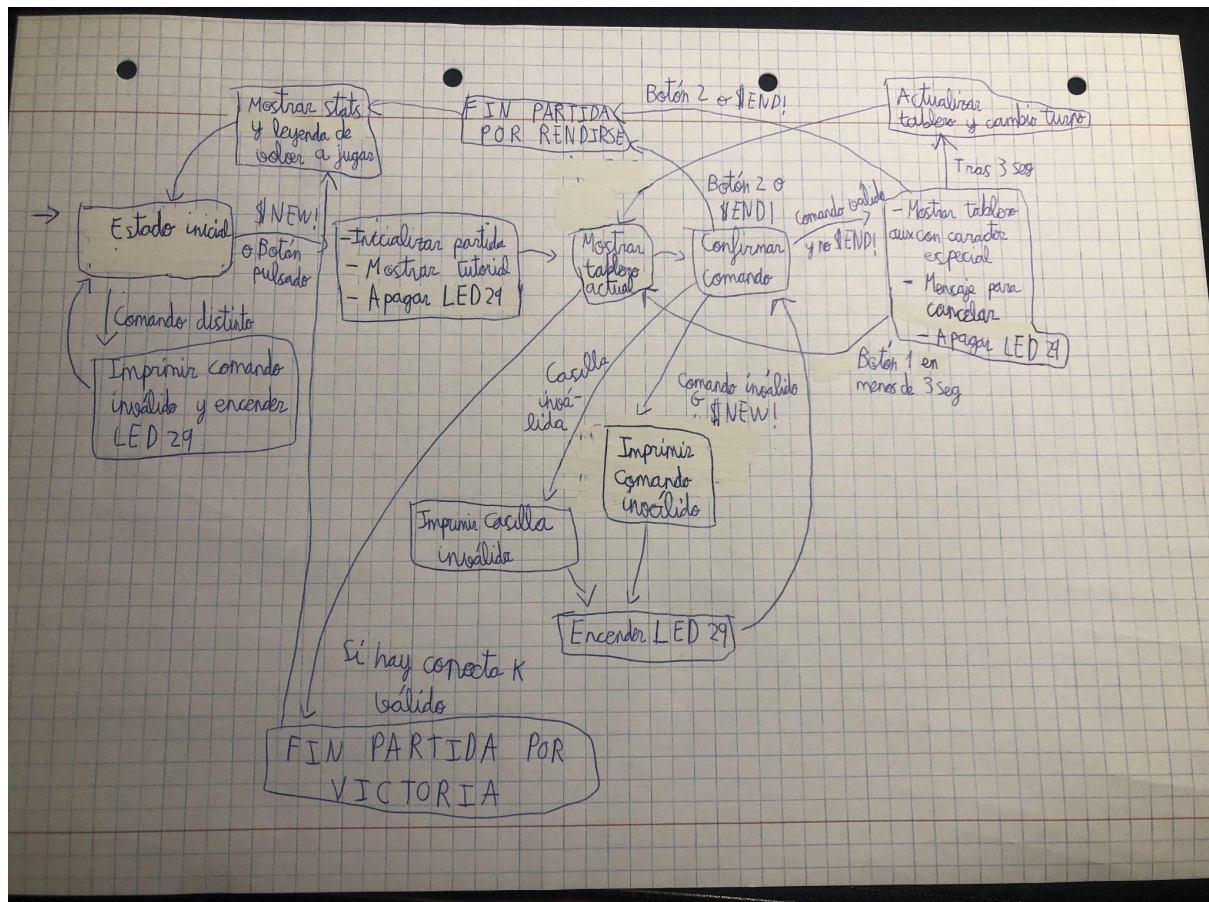
```
void linea_serie_drv_enviar_array(char *buffer){
    linea_serie_hal_enviar_array(buffer);
    linea_serie_hal_continuar_envio();
}
```

```
void linea_serie_drv_continuar_envio(char *buffer){
    linea_serie_hal_continuar_envio();
}
```

```
void funcion_callback(char letra){
    linea_serie_recibir_char(letra);
}
```

```
void funcion_callback_fifo_encolar(void) {
    FIFO_encolar(ev_TX_SERIE, 0);
}
```

7.2 Máquina de estados



7.3 Modificaciones realizadas respecto del código entregado anteriormente.

Se han realizado pequeñas modificaciones en el código, respecto al código entregado hace un mes

Anteriormente, al iniciar el juego, se carga automáticamente un tablero preestablecido de test. Ahora al iniciar se crea un tablero vacío sin ninguna pieza colocada. Sin embargo si se necesita volver a este tablero de test basta con descomentar la línea 273 del fichero “juego.c”

Además se han añadido comentarios para explicar de mejor forma que hace cada función.

No se ha modificado nada más, por lo tanto el código es muy parecido y los cambios son mínimos.