DISEÑO DE UN JUEGO PARA PANTALLA LCD GRÁFICA Y CONTROLADO POR EL MICROPROCESADOR HCS12

Memòria del Projecte Fi de Carrera
d'Enginyeria en Informàtica
realitzat per
Manuel Vázquez Ortega
i dirigit per
Vicente José Ivars Camáñez
Bellaterra 16 de Septiembre de 2009

Universitat Autònoma de Barcelona

etse
Escola Tècnica Superior d'Enginyeria

El sotasignat, Vicente-José Ivars Camañez Professor/a de l'Escola Tècnica Superior d'Enginyeria de la UAB,

CERTIFICA:

Que el treball a què correspon aquesta memòria ha estat realitzat sota la seva direcció per en Manuel Vázquez Ortega

I per tal que consti firma la present.

Signat: Vicente-José Ivars Camáñez

Bellaterra, 16 de setembre de 2009

Agradecimientos

A mis padres, a mis hermanas, a Maria Jesús, por estar ahí en todo momento y saber darme ánimos.

ÍNDICE

Indice de figuras e imágenes

Figuras **Página**

Imágenes **Página**

CAPÍTULO 1: Introducción

En el año 1971 nacía el primer microprocesador fabricado en un simple chip, de la mano de *Intel* con unas especificaciones muy sencillas en comparación de los de nuestros días. Esta *CPU* de 4 bits contaba con 46 instrucciones, 16 registros, un reloj de 740 *KHz* y era capaz de direccionar 5 *MB* de memoria *RAM*. Este encapsulado de 16 *pins* estaba formado en total por 2300 transistores.

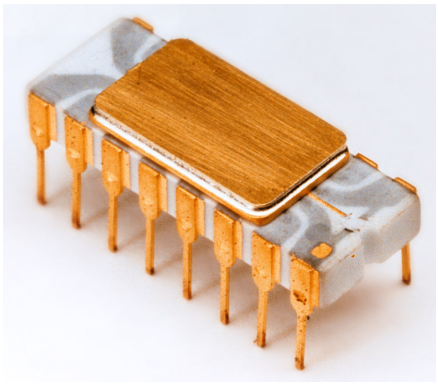


Imagen 1.1 - Microprocesador *Intel 4004*

Imagen 1.2. - Calculadora *Busicom 141-PF*

Intel recibió en 1969 el encargo de diseñar un conjunto de chips de propósito específico por parte del japonés *Calculating Machine Corporation* para su calculadora *Busicom 141-PF*. Diseñaron un conjunto de 4 chips; 3 de ellos de propósito específico y el *4004*[1][2] programable que ejecutaría el resto de tareas. Tras darse cuenta de las posibilidades de estos chips *Intel* recuperó los derechos sobre el encapsulado y desde entonces se ha convertido en una marca puntera en el diseño y comercialización, entre otras cosas, de microprocesadores hasta nuestros días.

Durante estos cuarenta años aparecieron también los microcontroladores, encapsulados que concentran tanto la *CPU* así como los diferentes unidades y periféricos que estas necesitan: memoria, buses, entradas y salidas, oscilador y conversor analógico/digital entre otros. El responsable del primer microcontrolador, en 1974, fue *Texas Instruments* que siguiendo la línea del *4004* creó el *TMS 1000* incluyendo un microprocesador, 1 *KB* de *RAM* y 256 *bits* de memoria *ROM* integrados en el mismo encapsulado.

Desde estos años, lejanos ya, los microcontroladores se han utilizado para diferentes metas y en muchos objetos y periféricos. Un ejemplo de ello, muy gráfico para este proyecto, es el microcontrolador 8084 de *Intel.* Que aparte de ser integrado en sintetizadores de música y teclados de IBM también fue utilizado en la consola de juegos *Magnavox Odissey²* de 1978.

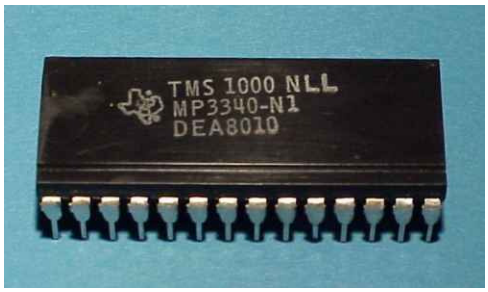


Imagen 1.3 - Microprocesador *Texas Instruments TMS 1000 NLL*

Imagen 1.4 - Consola *Magnavox Odissey²*

En cuanto a juegos se refiere debemos destacar el papel del microprocesador *Z-80*. Tras cuatro años como líder de proyectos en *Intel* diseñando entre otros el *4004* y el *8080*, *Federico Faggin* funda la compañía *Zilog e*n 1976, donde diseñará y comercializará este microprocesador. El *Z-80*[3] siendo compatible con el 8080, sigue siendo hoy día fabricado y utilizado en diversos dispositivos con ligeras modificaciones y adaptaciones. Este microprocesador fue utilizado en diversas consolas de juegos como la *Sega Master System* y *Sega Game Gear,* así como la *Nintendo Game Boy* y su primera versión en color que integraban una variante fabricada por *Sharp*.

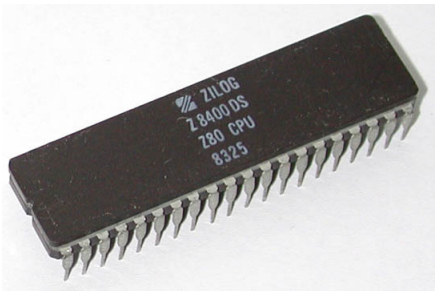


Imagen 1.5 - Microprocesador *Zilog Z80*

Imagen 1.6 - Consola *Gameboy* de *Nintendo*

Este proyecto consiste en diseñar y desarrollar un juego sencillo que se ejecutará en un microprocesador de la familia *HCS12*[4]. El juego se visualizará en una pantalla *LCD* táctil de 5,7 pulgadas. Los dos elementos, microprocesador y pantalla táctil, deberán conectarse y comunicarse en ambas direcciones. Aparte de la lógica microprocesador/pantalla para la visualización, necesitaremos comunicación pantalla/microprocesador para aprovechar la característica táctil de la pantalla para interactuar con el jugador.

El objetivo del proyecto no es otro que el diseño correcto de la configuración, interconexión y programación de código necesarios para ejecutar el juego de manera completamente funcional.

Se pretende también crear referencias para posteriores reutilizaciones y adaptaciones de todo ello, tanto de forma parcial como total, en futuras prácticas.

La memoria comienza con una pequeña introducción seguida del estado del arte. Después se verá en el capítulo tres el análisis que se hizo para poder concluir la viabilidad del proyecto así como para conocer el entorno. El capítulo cuatro describe el diseño realizado, las soluciones implementadas; también describe las pruebas realizadas al producto final y los problemas encontrados hasta llegar a este. La memoria finaliza con la conclusión seguida de varios anexos.

CAPÍTULO 2: Estado del arte

En la actualidad contamos con procesadores muy superiores a los anteriormente descritos. El desarrollo experimentado en el campo de las tarjetas gráficas ha facilitado, en gran medida, la proliferación de una industria de gran importancia: la de los videojuegos. Esta no solo se conforma del software en sí, sino también de las propias consolas, facturando durante el 2008 solo en España 1.432 millones de Euros según *aDeSe*[5], la *Asociación Española de Distribuidores y Editores de Software de Entretenimiento*.

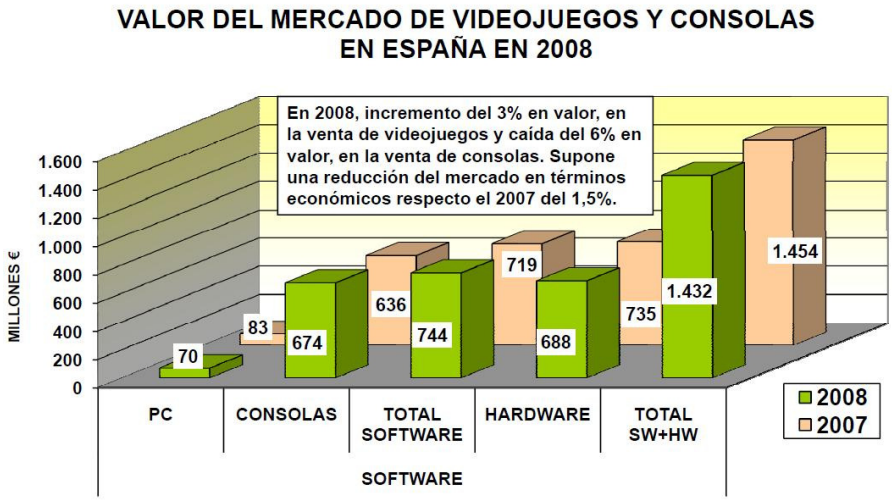


Imagen 2.1 – Valor del maercado de videojuegos y consolas en España en 2008

La arquitectura y tecnología utilizada en la fabricación de los microprocesadores ha evolucionado desde los tiempos del 4004 y como muestra podemos ver los *i7 Extreme Edition* de *Intel* y los *Phenom II X4 965* de *AMD*. Ambos procesadores están formados por 4 núcleos y su diseño enfocado directamente a la ejecución de tareas multimedia. Ambos están fabricados en 45 *nm*. Fuera del ámbito del sobremesa hemos visto como durante el pasado mes de Marzo la compañía *Fujitsu* ha anunciado, en el *Fujitsu Forum 2009* celebrado en Tokio, el desarrollo de un procesador de 8 núcleos capaz de alcanzar un pico máximo de 128 *Gigaflops* llamado *Venus* y la intención de abrir esta tecnología a la fabricación de procesadores de consumo general.

También debemos tener en cuenta la evolución de las tarjetas gráficas. Actualmente ofrecen, gracias a la tecnología *SLI* de *Nvidia* y *Crossfire* de *ATI,* la posibilidad de utilizar dos de ellas en paralelo.

Los microcontroladores son hoy día usados para muchos fines, y nos acompañan en el día a día en diferentes situaciones: aparatos electrodomésticos, automoción, domótica, periféricos informáticos o juguetería. También son utilizados en otros campos, quizás no tan próximos, como pueden serlo la navegación espacial, la electromedicina o el control industrial y la robótica. Diferentes modelos comerciales difieren en función de la finalidad de cada uno de ellos disponiendo de diseños específicos para su propio campo. Así tenemos, por ejemplo, la familia *MPC56XX* de *Freescale* enfocado a la industria automotriz, el *PIC18F87J90* de *Microchip Technology* para el control de pantallas *LCD* o bien la familia *AT32UC3A3* de *Atmel* para la reproducción de medios digitales. Podemos encontrar modelos de 8, 16 o 32 *bits* de diferentes fabricantes (*ARM,* Atmel, *Microchip Technology, Freescale...*) y con diferentes especificaciones asegurando siempre las suficientes opciones para encontrar el más adecuado para el proyecto que queramos desarrollar.

En el campo de la educación se suelen utilizar modelos de gama baja integrados en placas de desarrollo que, como es el caso del que se ha utilizado en este proyecto, incluyen el puerto *USB* necesario para su programación así como un oscilador y en alguna ocasión leds conectados a *pins* de entrada/salida, un botón de *reset* y una zona donde soldar nuestro circuito que conectaremos al MCU (Micro Controller Unit). En nuestra facultad utilizamos la placa *PK-HCS12E128* de *Softech,* que integra el microcontrolador *MC9S12E128* de *FreeScale*. En otras universidades se utilizan placas con microcontroladores de la misma gama y características como pueden ser el *PIC16F84A* en la Universidad de Vigo, o el *ARM CPC2378* en la universidad de Cádiz.

Basándose en estos chips y otros de diseño específico existe también un mercado de videojuegos portátiles de gama baja. Estos se venden integrados en un soporte que incluye tanto la pantalla como, normalmente, los mandos que permitirán jugar. Destacan por su bajo precio, sencillez y su gran presencia en diferentes tipos de comercio. Este proyecto se podría enmarcar dentro de este grupo de juegos dadas las características de nuestros dispositivos.

CAPÍTULO 3: Análisis y viabilidad del proyecto

Para la realización de este proyecto dispongo de un módulo fabricado por *Amulet*[6] *Technologies* con una pantalla táctil de 5,7 pulgadas y 320x240 píxeles controlada por el MCU de propósito específico *AGB64LV01-QC*[7] de la misma empresa. Gracias a este módulo se evita tener que programar a nivel de píxel, ya que implementa funciones gráficas tales como dibujar líneas o rectángulos. Además permite desarrollar: interfaces gráficas basadas en *HTML* restringido; utilizar imágenes en formatos *GIF* y *JPG*; y nos brinda cierto número de componentes, llamados *widgets* que pueden ser tanto de control como visualización, tales como botones, barras de progreso o checkboxes. El módulo dispone de comunicación sincrónica *SPI* (*Serial Peripheral* Interface) para poder ser programada y de comunicación asíncrona a través de una *UART* (*Universal Assynchronous Receiver-Transmitter*) para la interacción con el microcontrolador HCS12. El departamento dispone también de un placa de desarrollo de *Softech*[8] modelo *PK-HCS12E128*[9] con un microcontrolador *MC9S12E128*[10] de *Freescale*[11] que integra una CPU *HCS12* con una velocidad de bus a 25MHz, una memoria flash de 128*KB* y 8*KB* de memoria *RAM*. También dispone de 4 puertos de comunicación, siendo 3 de ellos asíncronos *SCI* (*Serial Communication Interface*) y uno sincrónico *SPI*.

Veamos a continuación estos dos elementos en detalle.

3.1. Placa de desarrollo *PK-HCS12E128*

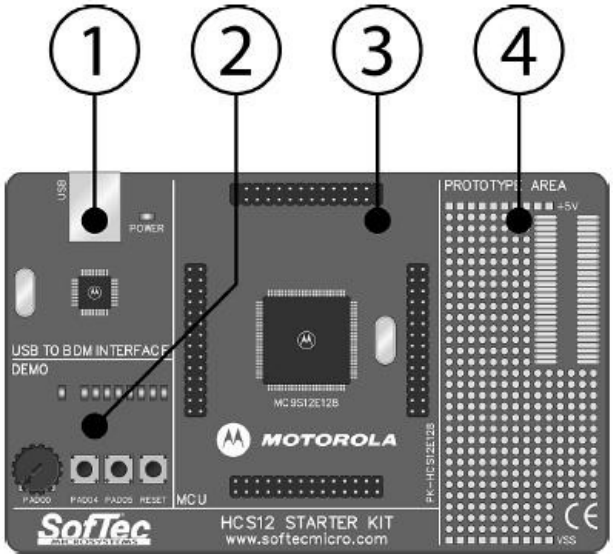


Imagen 3.1.1 - Placa de desarrollo ***Softech PK-HCS12E128***

Esta placa integra diferentes elementos separados en distintas zonas:

- 1 – Interfaz *USB-BDM*: Nuestro microcontrolador nos ofrece características de depuración como ejecución en tiempo real, *stepping* o *breakpoints,* gracias al módulo de depuración en segundo plano *BDM (Background Debug Module*). Esta sección contiene la circuitería necesaria para traducir, tanto eléctrica como lógicamente, los comandos BDM *d*el ordenador desde el que será programado. La alimentación de la placa se hará a través de la conexión USB aquí incluida.
- 2 – Sección de pruebas: Aquí encontramos diferentes elementos para que el usuario pueda realizar pruebas con el microcontrolador. Tenemos un potenciómetro; dos botones normales y uno de reset*; y* 9 leds. Todo ello está conectado a diferentes pins del *MCU.*
- 3 – Sección del microcontrolador: Nuestro HCS12 con la circuitería necesaria para su correcto funcionamiento y un oscilador de 16 *MHz* conectado a él. Los pins del MCU está conectados a los elementos de la sección de pruebas así como a los 4 conectores visibles en la imagen 3.1.1.

- 4 – Área de prototipos: Aquí se pueden montar o soldar los componentes que sean necesarios.

Pasemos a ver en detalle las características principales de nuestro HCS12:

- Núcleo HCS12 de 16 bits.
- 128 *KB* de memoria *flash*.
- 8 *KB* de memoria *RAM*.
- 3 interfaces de comunicación *SCI*.
- 1 interfaz de comunicación *SPI*.
- *3* timers de 16 bits de 4 canales.
- 25 *MHz* de velocidad de bus.
- Rango de voltaje de entrada desde 3.135V hasta 5.5V.
- Módulo de depuración en segundo plano (BDM).

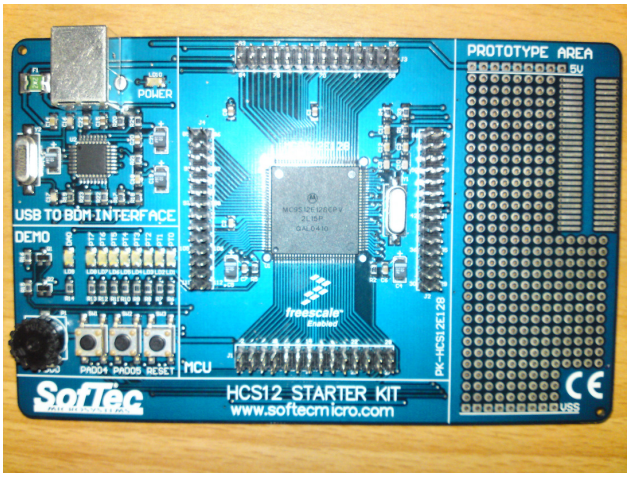


Imagen 3.1.2 - Foto de la placa en el laboratorio.

3.2 Módulo *STK-AOB3202405*

Este módulo está formado por la pantalla en sí, el microcontrolador *AGB64LV01-OC*, la pantalla táctil, una placa de interfaz de programación *PIB*[12] (*Program Interface Board*) que se explica a continuación y accesorios, como un lápiz especial y un cable serie. Veamos en detalle estos elementos.

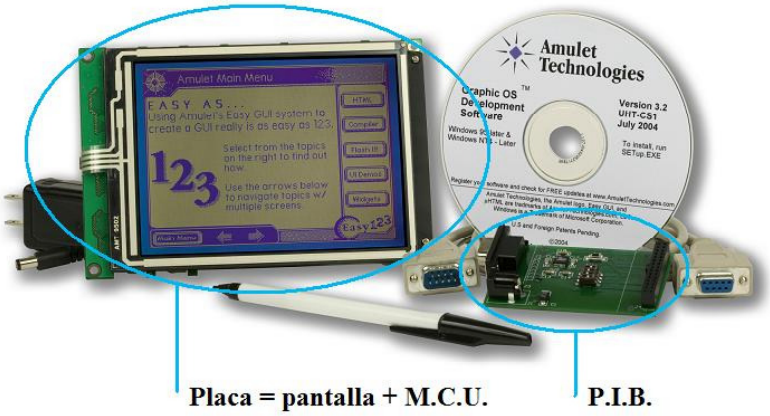


Imagen 3.2.1 - Módulo *STK-AOB3203405* de *Amulet*

- Pantalla táctil: Se trata de una pantalla táctil de 320x240 píxeles monocromo.
- Microcontrolador *AGB64LV01-QC:* Es un microcontrolador de propósito específico de 8 bits diseñado para el control de pantallas LCD, como la incluida en este módulo. El microcontrolador viene integrado en una placa, donde viene fijada también la pantalla. La placa contiene además todas las conexiones entre estos dos elementos y un conector donde irá el tercero de ellos: el PIB. El MCU gestiona la interfaz gráfica de usuario (*GUI, Grafic Interface User*) y permite su programación por medio de lenguaje HTML restringido, dando la opción a utilizar widgets predefinidos que veremos más adelante. Integra tanto el microprocesador de 8 bits así como una UART, un puerto SPI, un controlador de LCD y timers.
- PIB: Esta placa, la *IB*-RS232, integra el conector complementario al de la placa del *MCU* y 4 *switchs.* Deberemos modificar estos 4 *switchs* en función de si

queremos programar el módulo, devolverlo a sus valores de fábrica o calibrar nuestra pantalla. Controlaremos desde aquí la velocidad de transmisión de datos pudiendo variar esta entre 19.200 y 115.200 baudios. También incluye un transceptor *RS232* de *Texas Instruments*. Este es el encargado de traducir los voltajes de las señales que se enviarán y recibirán por protocolo *RS-232*, pasándolos de 3,3*V* a 5*V* y viceversa. También disponemos de un botón de *reset* y de un conector serie *DB9* con la opción de conectar los 3 hilos necesarios para esta comunicación directamente en el *PCB*. Finalmente será aquí donde se conecte la alimentación de todo el módulo con una fuente de 5V.

Las características de comunicación del módulo son las siguientes: velocidades de 9.600, 19.200, 57.600 o 115,200 baudios, sin paridad, 8 *bits* de datos y un *bit* de *stop*. La capacidad de almacenamiento para los archivos µHtml es de 8*Mbits*. La memoria interna consta de 199 variables de tipo string de 18 caracteres de longitud, 256 de tipo byte y 256 de 16 bits que podrán ser leídos y escritos tanto de forma interna como externa por medio de su UART. A pesar de contar con 3 hilos de comunicación nuestra pantalla operará en modo *Half Dúplex* y no en el esperado *Full Dúplex*.
Cada mensaje enviado y recibido contendrá una cabecera *MSOM* (*Master Start Of Message*) que lo definirá de manera unívoca y vendrá seguido de los datos correspondientes. En el anexo número uno se puede observar la relación de cada una de estas cabeceras así como los mensajes de eco que el receptor debe enviar a su vez para confirmar la recepción. Otra característica de la comunicación de nuestra pantalla es el protocolo de codificación de datos a enviar a través de su puerto asíncrono. Para ver en detalle el funcionamiento de esta, incluyendo diferentes ejemplos, pueden ver el anexo número dos de esta memoria. Finalmente, en cuanto a comunicación cabe destacar también la posibilidad de enviar a la pantalla imágenes gif animadas, en tiempo de ejecución, a través de un protocolo específico llamado *Xmodem*. Dicho protocolo será explicado más adelante.

Imagen 3.2.2 - Microcontrolador *AGB64LV01-QC* de *Amulet*

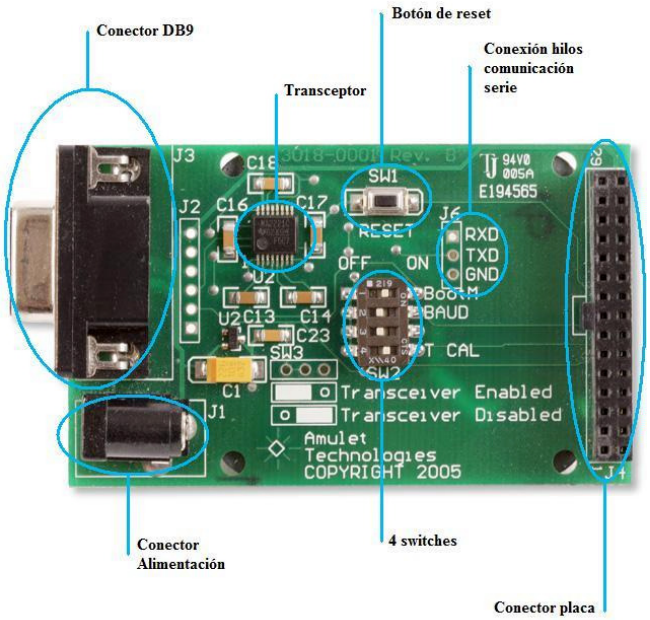


Imagen 3..2.3 - *PIB* modelo *IB-RS232*

Gracias a los widgets de Amulet ya citados se pueden introducir objetos de interfaz de usuario fácilmente. Estos se añaden dentro del código *HTML* a modo de applets de *java* pudiendo configurar sus diferentes parámetros. Existen 14 diferentes separados en dos grupos: los widgets de visualización y los widgets de control. Los primeros no son interactivos y se limitan a mostrar imágenes o a variar en función del valor de una variable asociada a ellos en su definición. Entre ellos contamos con barras de progreso, líneas gráficas y marcadores, tanto numéricos como alfanuméricos, entre otros. Los widgets de control sin embargo serán los responsables del cambio de valor de una variable en función del uso que les de el usuario. Entre ellos contamos con

check boxes, sliders y listas de objetos seleccionables. Las variables en ambos casos pueden ser tanto internas de la pantalla como externas accesibles a través de la *UART*.

3.3 Comunicación

Tal y como se ha visto anteriormente ambos elementos disponen de puertos de comunicación asíncrona. El microcontrolador *MC9S12E128* puede trabajar con el protocolo RS-232 de 3 hilos y la pantalla puede por igual hacerlo a través de su *UART* utilizando la conexión disponible en su PIB. Configurando el puerto serie 0 del HCS12 como SPI los pines necesarios serían el *PS0/RXD0* (número 57), *PS0/TXD0* (número 58) y la señal VSS2 como tierra (número 70). La conexión entre ambos elementos la podemos observar en la siguiente imagen:

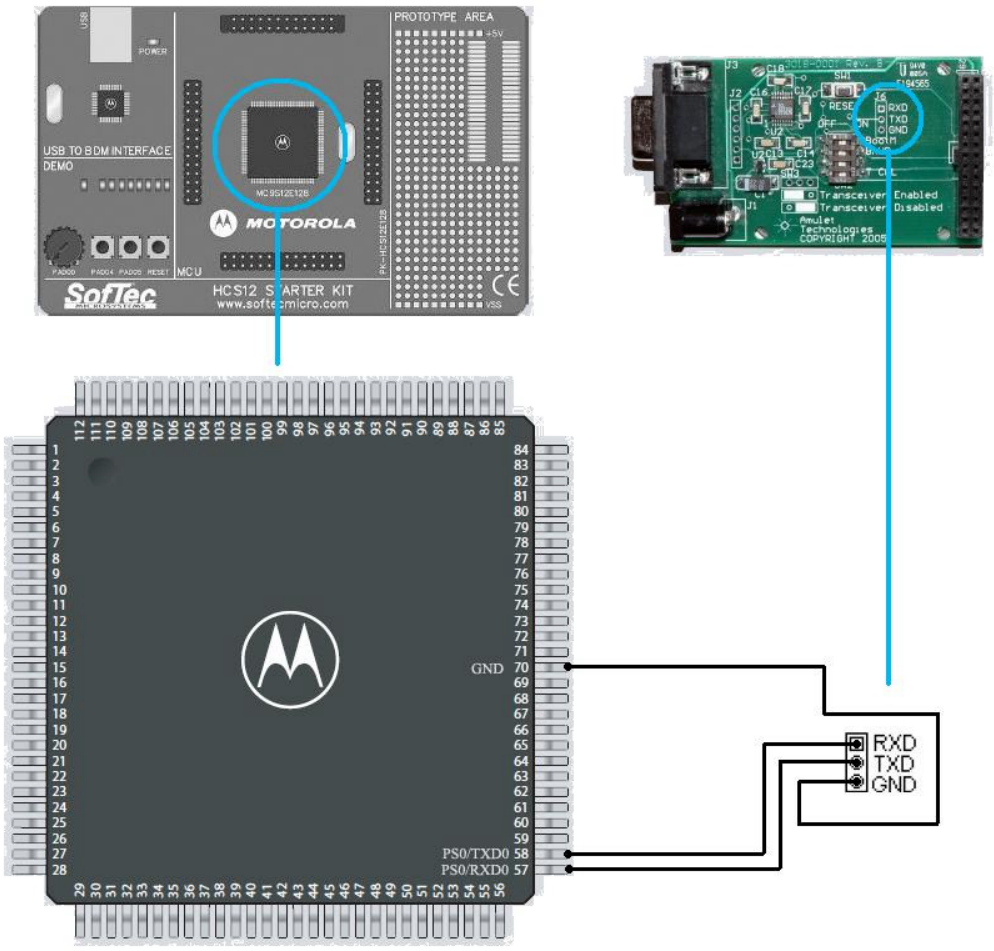


Figura 3.3.1 - Esquema de interconexión de los dos dispositivos.

3.4 Descripción del juego

Se trata de un juego basado en el conocido *'BeJeweled'* de *'PopCap Games*[13]' y es de tipo puzzle. Tenemos un tablero de 8x8 casillas con gemas de 7 tipos básicos diferentes. El objetivo consistirá en cambiar la posición de una gema con otra adyacente para formar cadenas de al menos 3 gemas del mismo tipo, ya sean estas verticales u horizontales. Al conseguir esto las gemas de dicha cadena desaparecerán y las gemas situadas sobre su posición caerán sobre el vacío dejado, entrando en juego nuevas gemas por las posiciones superiores de la pantalla en su vertical. En caso de conseguir una cadena de 4 gemas del mismo tipo desaparecerán 3 de ellas y la cuarta se transformará en una gema del mismo tipo que la original pero a su vez especial y diferenciada por un brillo u otra característica gráfica. Al incluir esta nueva gema en otra cadena de al menos 3 gemas iguales, esta gema especial explotará destruyendo las gemas de su contorno además de desaparecer la cadena formada. Por otro lado si conseguimos que la cadena sea de 5 gemas del mismo tipo, o superior, ocurrirá lo mismo que ocurre con una cadena de 4 con la diferencia de que la gema creada será completamente diferente de los 7 tipos iniciales. Este nuevo tipo de gema servirá de comodín y mediante su uso se podrán destruir todas las gemas que tengamos en pantalla del mismo tipo de la gema con la que se intercambie.

Al destruir gemas cada una de ellas se traducirá en puntos para el jugador siendo la puntuación de cada gema un número fijo. Esta puntuación se verá incrementada en lo que llamaremos reacciones en cadena. Llamaremos así al caso en que cuando las gemas superiores de las gemas ya destruidas formen nuevas cadenas al caer y colocarse en su nueva posición. Cuando estas reacciones en cadena sean sucesivas los puntos ganados por gema destruida se incrementarán con cada una de ellas. Al incrementar los puntos también se conseguirá incrementar la barra de progreso de juego. Cuando esta llegue a su límite se incrementará, a su vez, el nivel de juego.

Como característica principal hay que destacar que siempre deberá haber, como mínimo, un movimiento correcto posible, dependiendo así la duración de la partida únicamente del criterio del usuario.

La pantalla de juego dispondrá de diferentes elementos. En primer lugar, y más importante, de un tablero de 8x8 casillas siendo estas en total 64. Dos marcadores nos

indicarán tanto el nivel actual de la partida como las puntuación conseguida hasta el momento. Una barra de progreso mostrará hasta que punto se ha avanzado dentro del nivel actual de juego. Un botón de ayuda nos indicará, al ser pulsado, una (o la única) de las piezas que podrán formar una cadena correcta al ser intercambiada con una de sus vecinas. Al utilizar este botón de ayuda se descontarán puntos del marcador. Finalmente la pantalla contará con otro botón que permitirá el abandono de la partida tras una confirmación. Mientras se esté a la espera de esta confirmación el tablero de juego será borrado redibujandose en caso de cacenlar el abandono.

A continuación tenemos el diagrama de flujo del funcionamiento del juego.

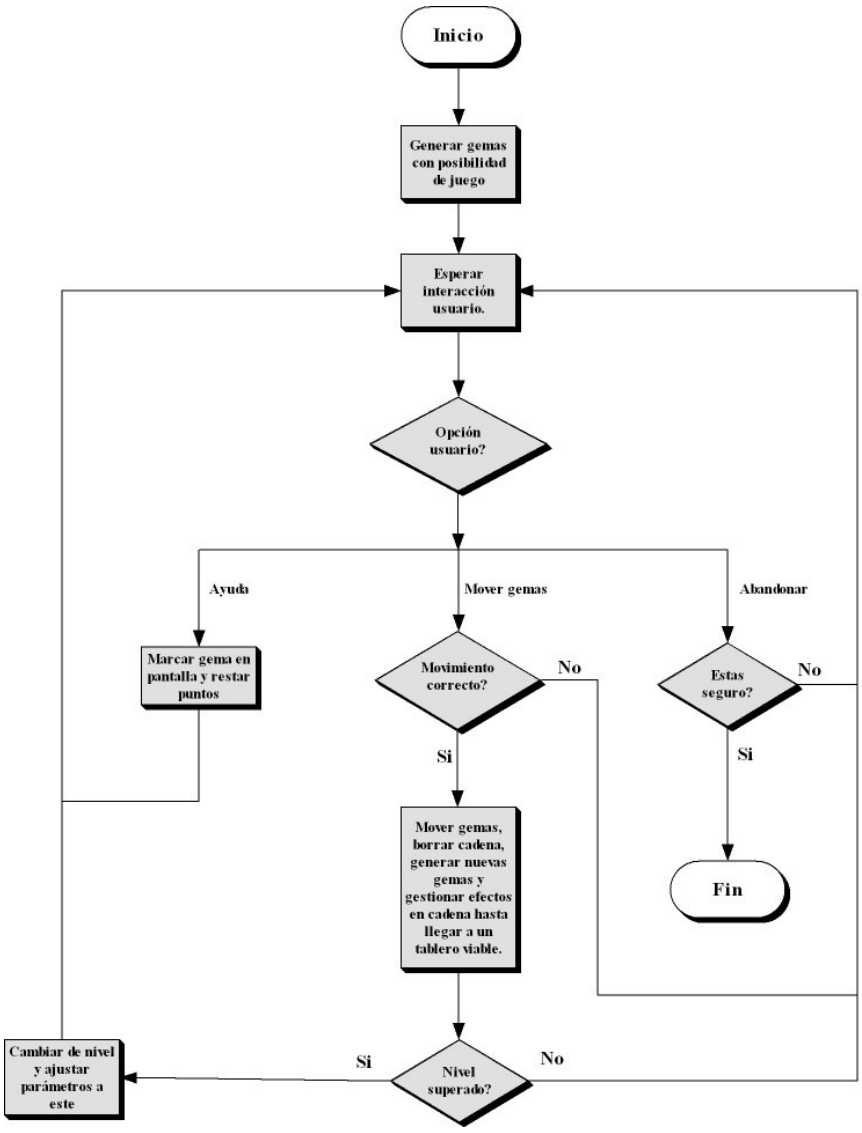


Figura 3.4.1 - Diagrama de flujo del juego.

3.5 Requerimientos

Al tratarse de un juego nuestra aplicación deberá constar de diferentes pantallas. Entre ellas debemos tener como mínimo la de inicio, siendo deseable tener otra que detalle las instrucciones de juego para los usuarios no familiarizados con este. Deberamos tener también una pantalla de menú que permita acceder tanto al juego como a las instrucciones. Veamos las diferentes pantallas de las que constará el juego así como los elementos de cada una de ellas.

3.5.1 Pantalla de inicio

Esta será la pantalla más sencilla de la aplicación y no constará más que de alguna imagen de fondo acompañada de un botón que permitirá llegar a la pantalla de menú.

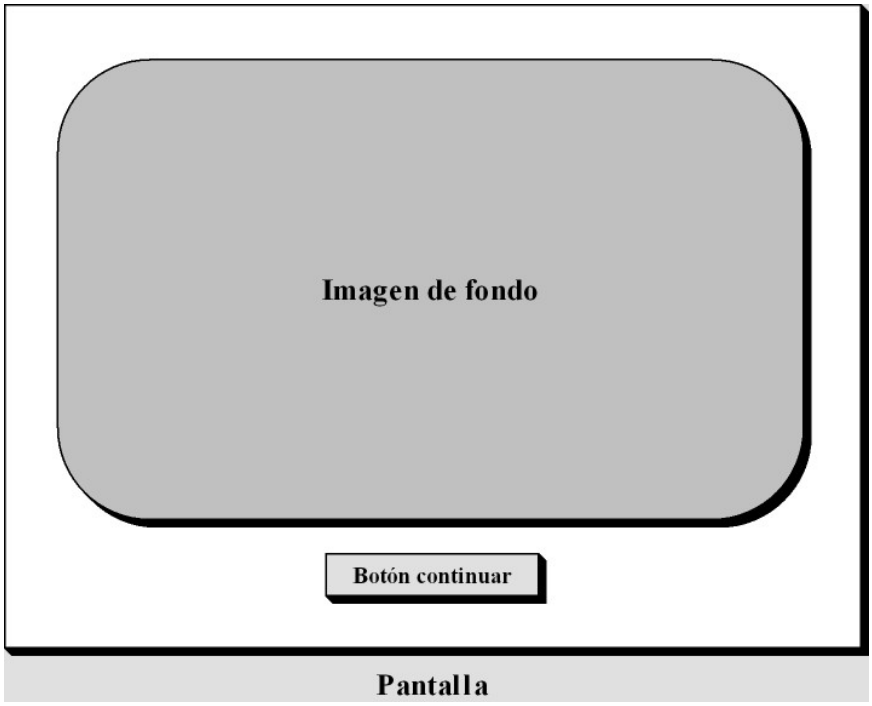


Figura 3.5.1.1 - Esquema de la pantalla de inicio de la aplicación

3.5.2 Pantalla de menú

Aquí podremos escoger si comenzar una partida o bien si visualizar las instrucciones del juego. Ambas opciones se ofrecerán a través de sendos botones donde se podrá leer su funcionalidad. La pantalla dispondrá de una imagen de fondo para hacerla estéticamente atractiva.

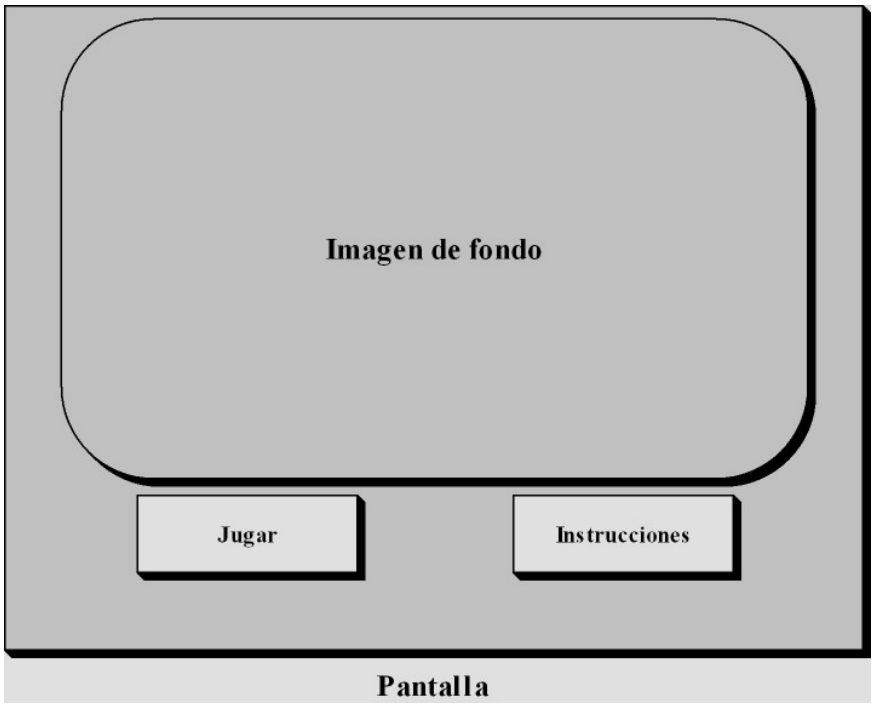


Figura 3.5.2.1 - Esquema de la pantalla de inicio de menú.

3.5.3 Pantalla de instrucciones

Esta parte de la aplicación podrá contar con una o más pantallas que detallen el funcionamiento del juego. Cada una de ellas constará de imágenes ilustrativas para facilitar la comprensión así como de cuadros de texto donde se describirán las diferentes situaciones posibles durante la partida. Constarán también de botones que permitirán pasar hacia delante y hacia atrás en el conjunto de pantallas. Asimismo cada una de ellas tendrá un botón por medio del cual volver al menú en cualquier momento.

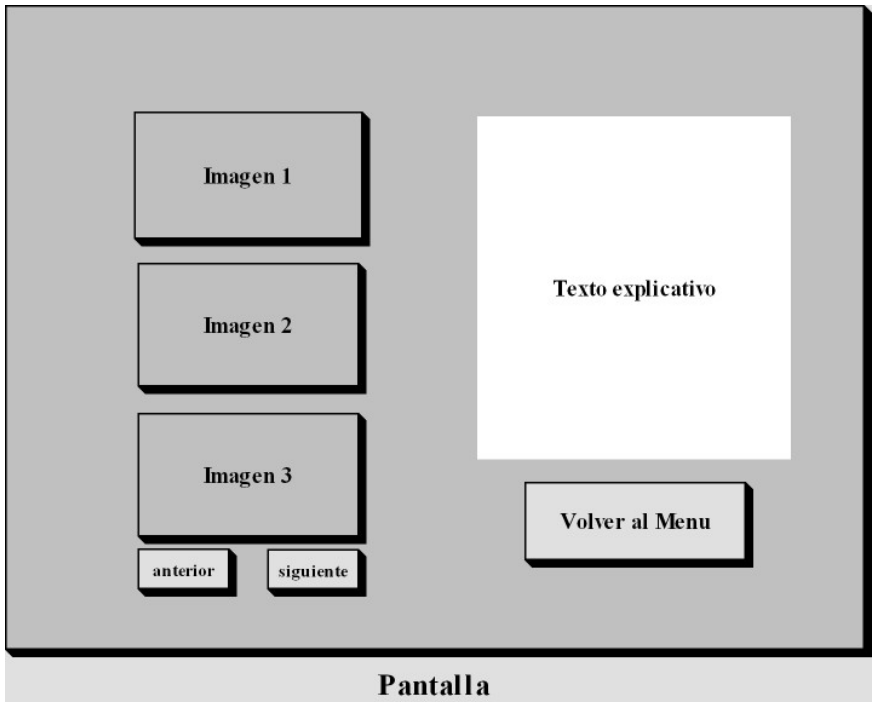


Figura 3.5.3.1 - Esquema de la pantalla de instrucciones de juego.

3.5.4 Pantalla de juego

Esta será la pantalla más importante y compleja de la aplicación. Tal y como se ha explicado en el capítulo anterior constará de: tablero de 8x8 casillas, marcador de nivel y marcador de puntuación, barra de progreso así como de dos botones, uno para abandonar la partida y otro para pedir ayuda, siendo ambos visibles desde el principio de la partida. También dispondrá de otros dos botones, que solo serán visibles en caso de haber sido pulsado previamente el botón de abandono de partida, que servirán para confirmar o cancelar dicho abandono. Ambos marcadores deberán actualizar su valor periódicamente para mostrar en tiempo real la puntuación y progreso reales. En la siguiente imagen se puede apreciar un esquema de esta pantalla.

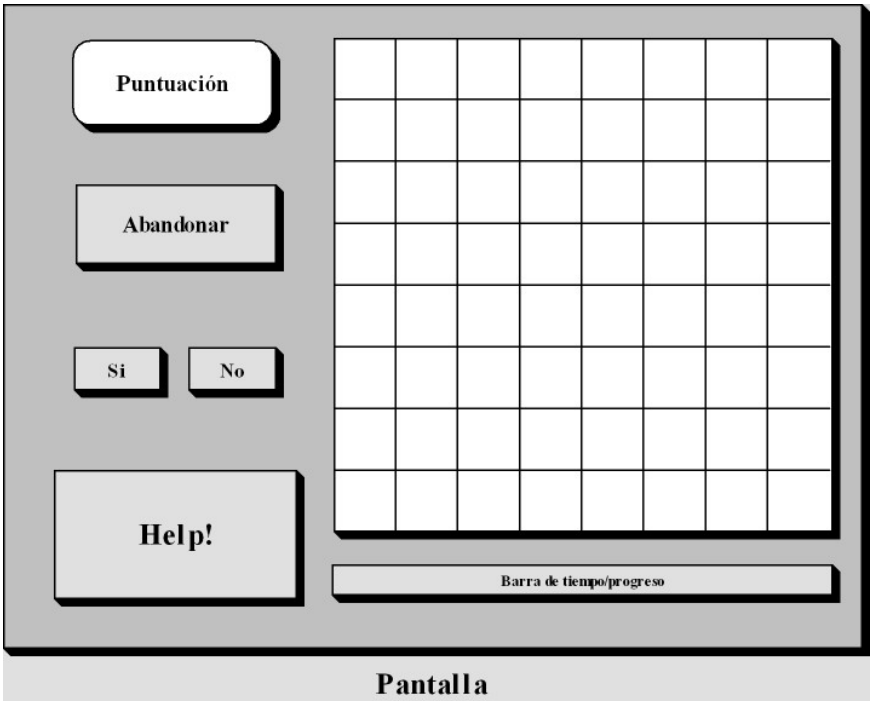


Figura 3.5.4.1 - Esquema de la pantalla de juego.

Como síntesis de lo anteriormente expuesto en la siguiente imagen se puede observar un esquema general del juego con los enlaces entre cada una de las pantallas.

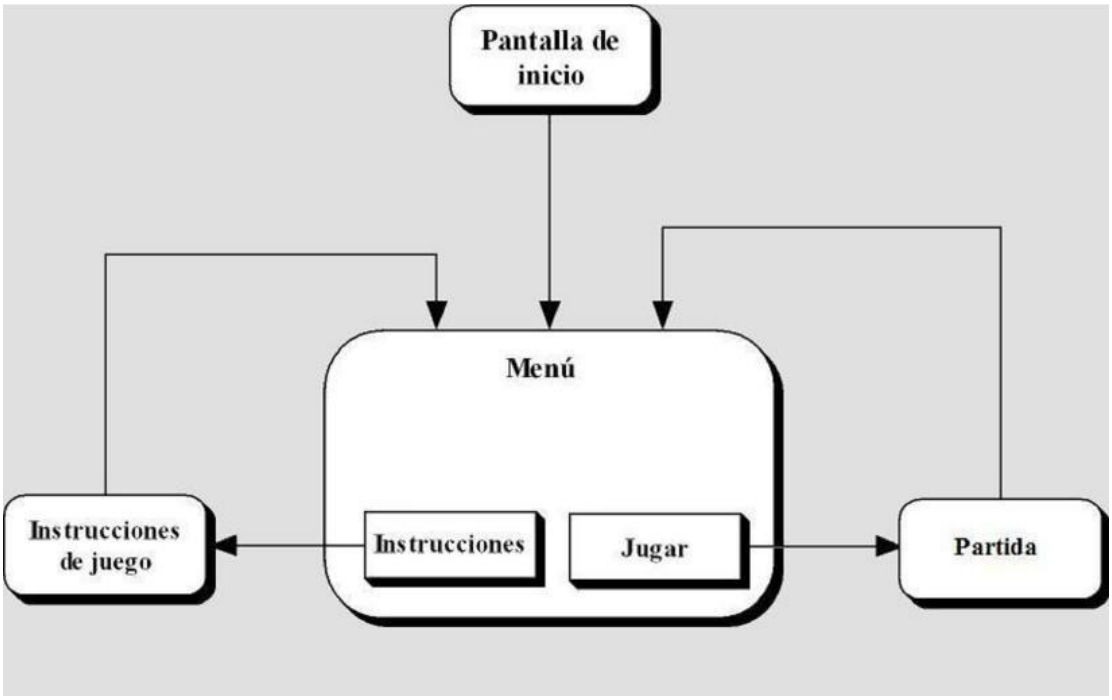


Figura 3.5.3.2 - Esquema general de las diferentes pantallas de la aplicación.

En referencia al funcionamiento del juego se deben tener en cuenta ciertos detalles en el orden de acontecimientos de la formación de cadenas de piezas. A continuación se describen las diferentes situaciones que el jugador puede encontrar a lo largo de una partida.

Cuando el jugador selecciona la primera de las gemas, alrededor de esta se formará un cuadro de selección que la marcará. Al seleccionar la segunda pieza pueden ocurrir dos cosas: que el movimiento no permita crear una sola cadena o que permita crear una o más. En el primero de los casos la primera pieza seleccionada perderá el cuadro que la estaba marcando. Si el movimiento es correcto pueden haberse creado en un primer momento una o dos cadenas de gemas. En tal caso cada una de las piezas correctas se marcará de color oscuro en pantalla y solo cuando todas las gemas correctas estén marcadas estas podrán iniciar el proceso de desaparición.

Este proceso consiste en borrar de la pantalla todas las gemas implicadas en las cadenas formadas a excepción de las que sean especiales en caso de haber alguna. Estas piezas, tal y como queda explicado en la descripción del juego destruyen todas sus gemas circundantes. Estas serán marcadas y posteriormente borradas siendo siempre la última en desaparecer la gema especial. Cabe el caso de que una pieza especial destruya otra gema especial creando un efecto de explosión en cadena. En tal caso la última gema especial en desaparecer será siempre la primera en explotar; y viceversa, la primera pieza especial en desaparecer de pantalla será la última en haber explotado.

Todas las gemas que desaparecen dejan huecos libres que serán ocupados por las superiores a su posición. La caída de estas se realizará de forma secuencial cayendo todas las gemas implicadas de forma simultánea mientras entran en juego las nuevas gemas por la vertical superior. Al terminar la caída pueden haber de nuevo cadenas de gemas formadas. En ese caso el proceso será el mismo que el descrito para las gemas intercambiadas por el jugador.

En caso de que una de las gemas sea un comodín, las piezas del mismo tipo que la segunda gema implicada se marcarán todas por pantalla y tras esto desaparecerán. Mientras dure este proceso la gema comodín parpadeará de forma intermitente y al finalizar desaparecerá también.

Cuando el jugador pulse el botón de ayuda, el juego le mostrará la gema que deberá mover aplicando un parpadeo sobre ella. Parpadeará también el marcador de nivel cuando este se incremente.

3.6 Viabilidad del proyecto

Una vez analizadas las características de los dispositivos disponibles así como el funcionamiento del juego y requisitos de cada una de las pantallas de este, queda por decidir si su implementación resulta viable o no.

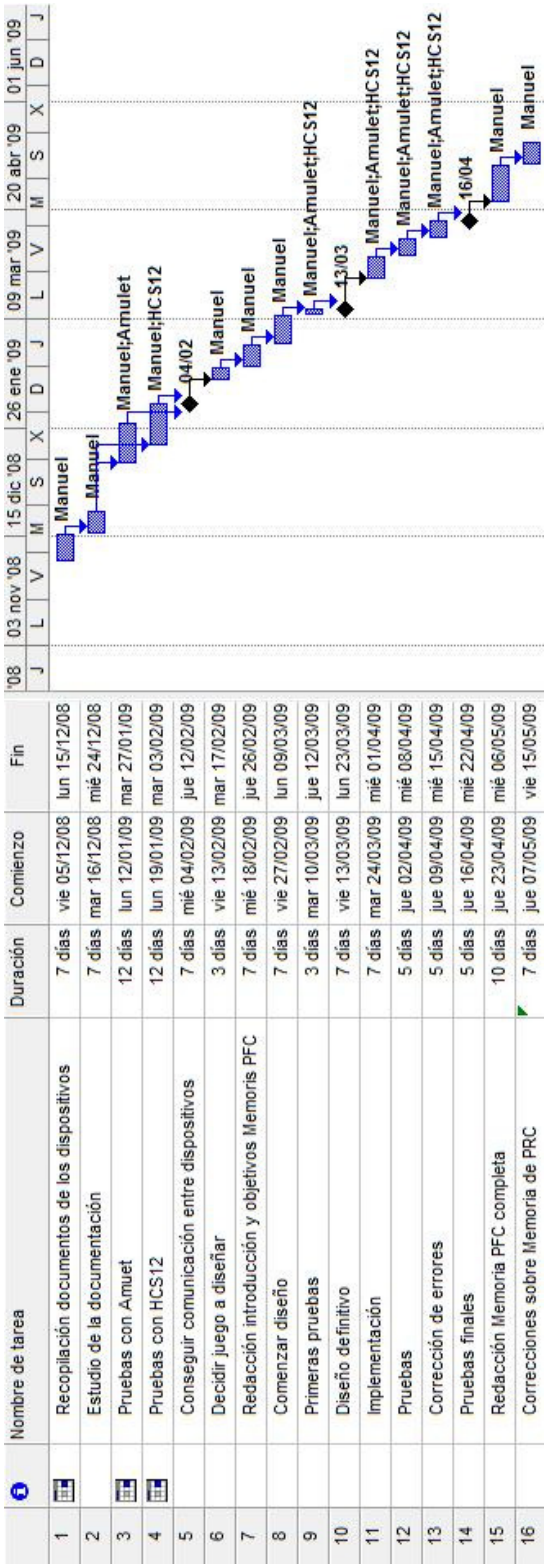
Se dispone de la tecnología necesaria para poder comunicar los dos elementos disponibles así como el software correspondiente para poder compilar y cargar en memoria los programas que se implementen. Por un lado la herramienta *Code Warrior* y por el otro el compilador de código Html contenido en el kit de Amulet.

La herramienta Code Warrior de *Metrowerks* permite programación en *C* y *ensamblador*. El modo *processor expert* permite añadir módulos, llamados *beans*, tales como timers o comunicaciones asíncronas muy fáciles de configurar mediante el tablero de especificaciones de cada uno de ellos. Por otro lado el compilador que distribuye la casa Amulet permite generar código y cargarlo en la memoria flash del dispositivo de forma muy rápida.

La funcionalidad del proyecto no implica la necesidad de gran cantidad de recursos, dada la sencillez del juego, y su diseño no acarrea la implementación de algoritmos complejos.

Visto esto se puede afirmar que el proyecto es viable.

3.7 Planificación temporal



	Nombre de tarea	Duración	Comienzo	Fin
1	Recopilación documentos de los dispositivos	7 días	vie 05/12/08	lun 15/12/08
2	Estudio de la documentación	7 días	mar 16/12/08	mié 24/12/08
3	Pruebas con Amuet	12 días	lun 12/01/09	mar 27/01/09
4	Pruebas con HCS12	12 días	lun 19/01/09	mar 03/02/09
5	Conseguir comunicación entre dispositivos	7 días	mié 04/02/09	jue 12/02/09
6	Decidir juego a diseñar	3 días	vie 13/02/09	mar 17/02/09
7	Redacción introducción y objetivos Memoris PFC	7 días	mié 18/02/09	jue 26/02/09
8	Comenzar diseño	7 días	vie 27/02/09	lun 09/03/09
9	Primeras pruebas	3 días	mar 10/03/09	jue 12/03/09
10	Diseño definitivo	7 días	vie 13/03/09	lun 23/03/09
11	Implementación	7 días	mar 24/03/09	mié 01/04/09
12	Pruebas	5 días	jue 02/04/09	mié 08/04/09
13	Corrección de errores	5 días	jue 09/04/09	mié 15/04/09
14	Pruebas finales	5 días	jue 16/04/09	mié 22/04/09
15	Redacción Memoria PFC completa	10 días	jue 23/04/09	mié 06/05/09
16	Correcciones sobre Memoria de PRC	7 días	jue 07/05/09	vie 15/05/09

Figura3.7.1 – Diagranade Gant previsto

CAPÍTULO 4: Diseño

A continuación se detallará el diseño modular de algunas de las funciones más importantes que necesitará realizar nuestro juego para garantizar un perfecto funcionamiento. Solo veremos el diseño de los más importantes de ellos, comentando superficialmente la funcionalidad de otros que harán tareas secundarias. Después de ello veremos las soluciones propuestas para las diferentes pantallas, los problemas encontrados durante todo el desarrollo del proyecto. El capítulo finaliza con las pruebas realizadas al producto.

Cabe destacar que en esta fase de diseño hay que tener presentes los requerimientos anteriormente expuestos, ya que alguno de ellos será decisivo a la hora de describir el esquema modular de la aplicación. Otros obligarán a crear en diferentes módulos funcionalidades que a priori se podrían hacer descrito dentro de uno solo.

4.1 Módulo principal

Este módulo gestionará casi todos los mensajes enviados por el Amulet a nuestro HCS12. Al iniciar la aplicación se deberá esperar en primer lugar la recepción del mensaje de inicio de partida. Al recibirlo otro módulo deberá inicializar puntuación, progreso, nivel y crear un tablero viable. Después de esto, ya el usuario en la pantalla juego, existirán tres tipos de mensajes a recibir: coordenadas de gemas a intercambiar, demanda de ayuda, o abandono de partida. En cada uno de los tres casos se llamarán módulos diferentes que serán los encargados de ejecutar las acciones pertinentes.
Se deberán recibir dos coordenadas antes de llegar a analizar el juego, pudiendo recibir entre estas cualquier señal de los otros dos tipos. Este módulo se ejecutará de manera constante en forma de bucle infinito mientras el dispositivo se mantenga en funcionamiento.

A continuación se puede observar el diagrama de flujo de este módulo.

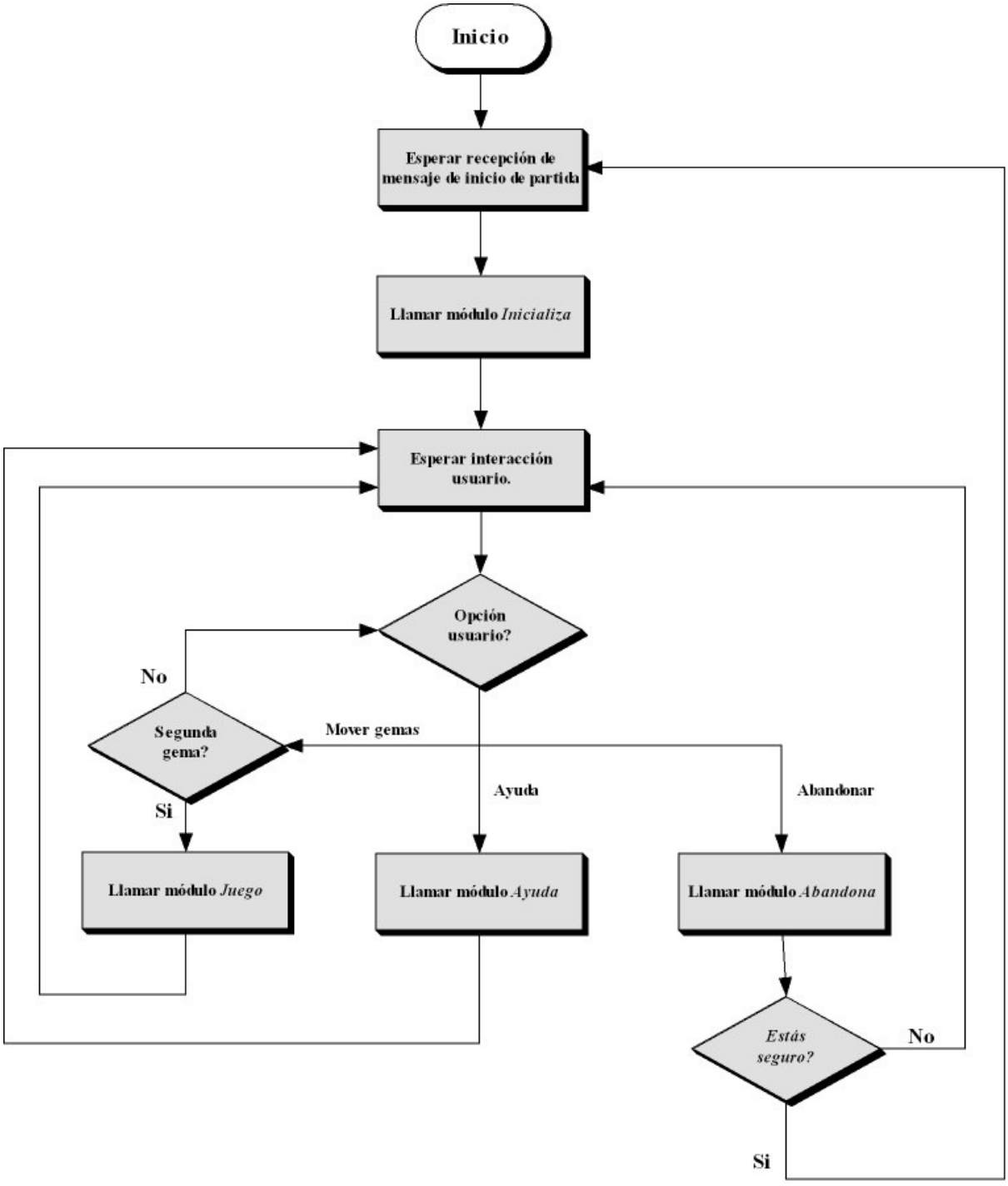


Figura 4.1.1 - Diagrama de flujo del módulo principal

4.2 Módulo *Juego*

Con las dos coordenadas seleccionadas por el usuario, lo primero que se hará será comprobar si las posiciones son vecinas vertical u horizontalmente. En caso negativo saldremos del módulo. En caso afirmativo distinguiremos dos casos. El primero será cuando una de las dos coordenadas contenga un comodín, llamando al módulo del mismo nombre. Para el segundo de los casos cabe destacar uno de los requerimientos del capítulo anterior. Las gemas deben primero marcarse como correctas en su totalidad para más tarde desaparecer simultáneamente; tanto las gemas de la posible cadena formada por la primera coordenada como las de la segunda. Por lo tanto habrá que crear módulos diferentes: uno que verifique la existencia de cadenas para una sola gema y las marque como correctas por pantalla, y otro que se ocupe de borrar las gemas de este caso general. Los llamaremos *Verifica* y *BorraGemas* respectivamente. En el caso particular de *Comodín* las gemas podrán ser borradas de forma interna. No se debe olvidar que los marcadores deben de ser actualizados y por tanto uno de estos dos módulos deberá contablilizar los puntos a sumar al marcador general. Tras ser borradas las piezas de la/s cadena/s se deberán crear nuevas gemas en los huecos dejados por estas, formando un tablero viable; el módulo será llamado *Caída*. Cabe la posibilidad de existir nuevas cadenas tras todo ello, es decir reacciones en cadena. Se creará otro módulo encargado de ello, llamado *ReaccionEnCadena*.

En la imagen 4.2.1 se puede observar el diagrama de flujo del módulo.

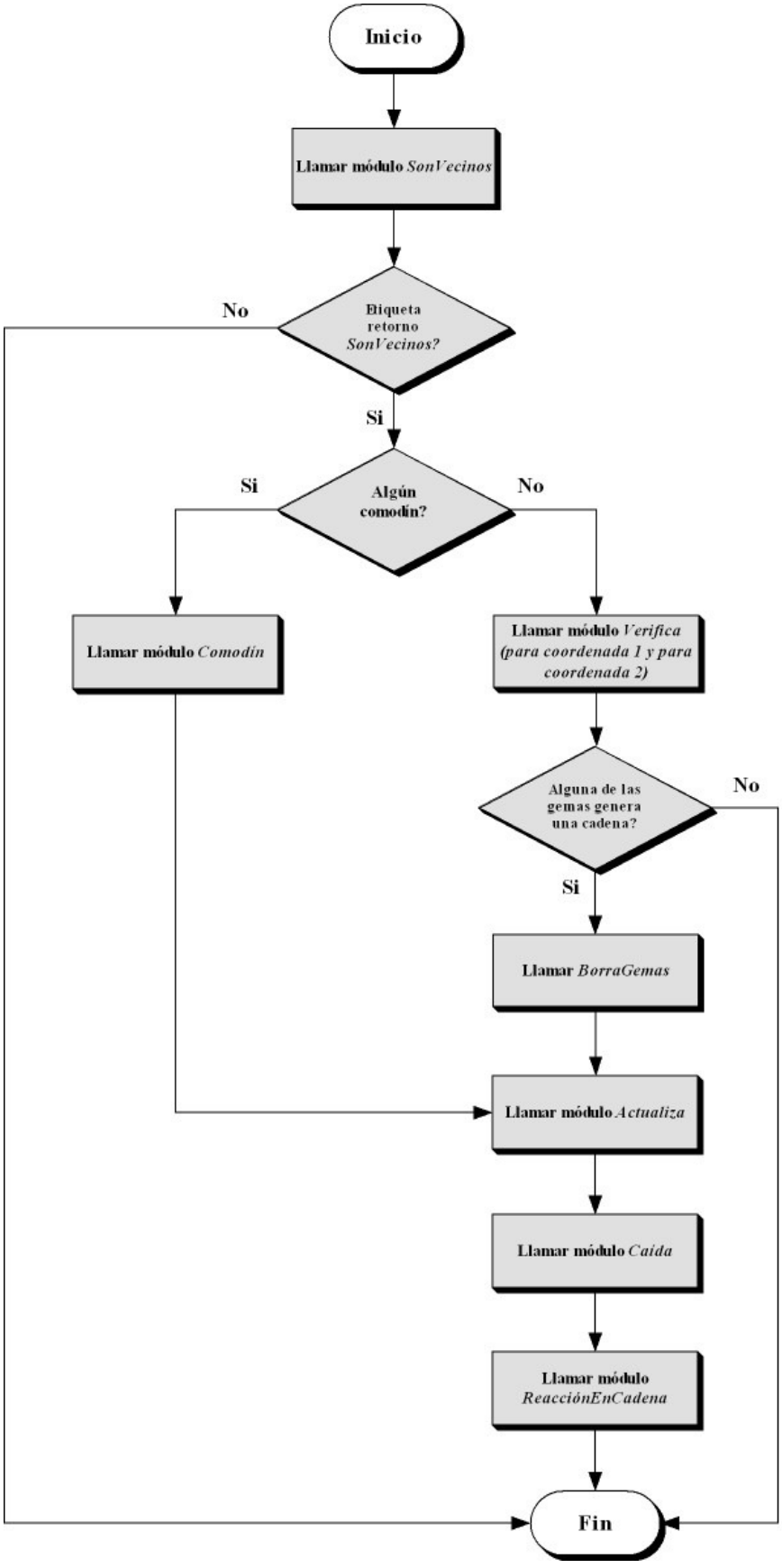


Figura 4.2.1 - Diagrama de flujo del módulo *Juego*.

4.3 Módulo *Verifica*

Este módulo será llamado en dos casos muy diferentes. Se ejecutará cuando el usuario pulse dos gemas y se ejecutará cuando se necesite comprobar la viabilidad del estado del tablero de juego. En ambos casos el módulo analizará si la gema, que se encuentra en la coordenada que se recibe como parámetro, forma una cadena. El módulo deberá devolver una variable de control que diga si la coordenada ha formado una cadena: para continuar ejecutando la secuencia de módulos del proceso, en el primer caso, y para saber si hemos generado un tablero viable, en el segundo. Aparte de esto en el primero de los casos se marcarán de manera interna las gemas que hayan formado una cadena para más delante poder borrarlas sin tener que recorrer el tablero en su busca. El encargado de borrarlas será, como ya se ha dicho anteriormente, el módulo *BorraGemas*.

La imagen 4.3.1 muestra el diagrama de flujo de este módulo.

4.4 Módulo *Borragemas*

En una cadena de gemas sin comodines este será el módulo será el encargado de borrar las gemas, gráficamente, del tablero, así como de la estructura de datos que se utilice para representar las gemas que lo conforman. En primer lugar se señalarán en la pantalla las gemas normales marcadas anteriormente como correctas. Después se borrarán de dicha pantalla y de la estructura de datos simultáneamente. En tercer lugar dibujaremos de nuevo por pantalla las gemas especiales para asegurar la visualización de las creadas a causa de el movimiento que se está procesando. Finalmente se procesan todas las piezas especiales que tengan que explotar, se borran de pantalla y se borran del tablero. Por cada pieza que se borre del tablero se contabilizará un número de puntos que se devolverá como parámetro a la función que la hubiera llamado.

El diagrama de flujo de este módulo se puede observar en la imagen 4.4.1.

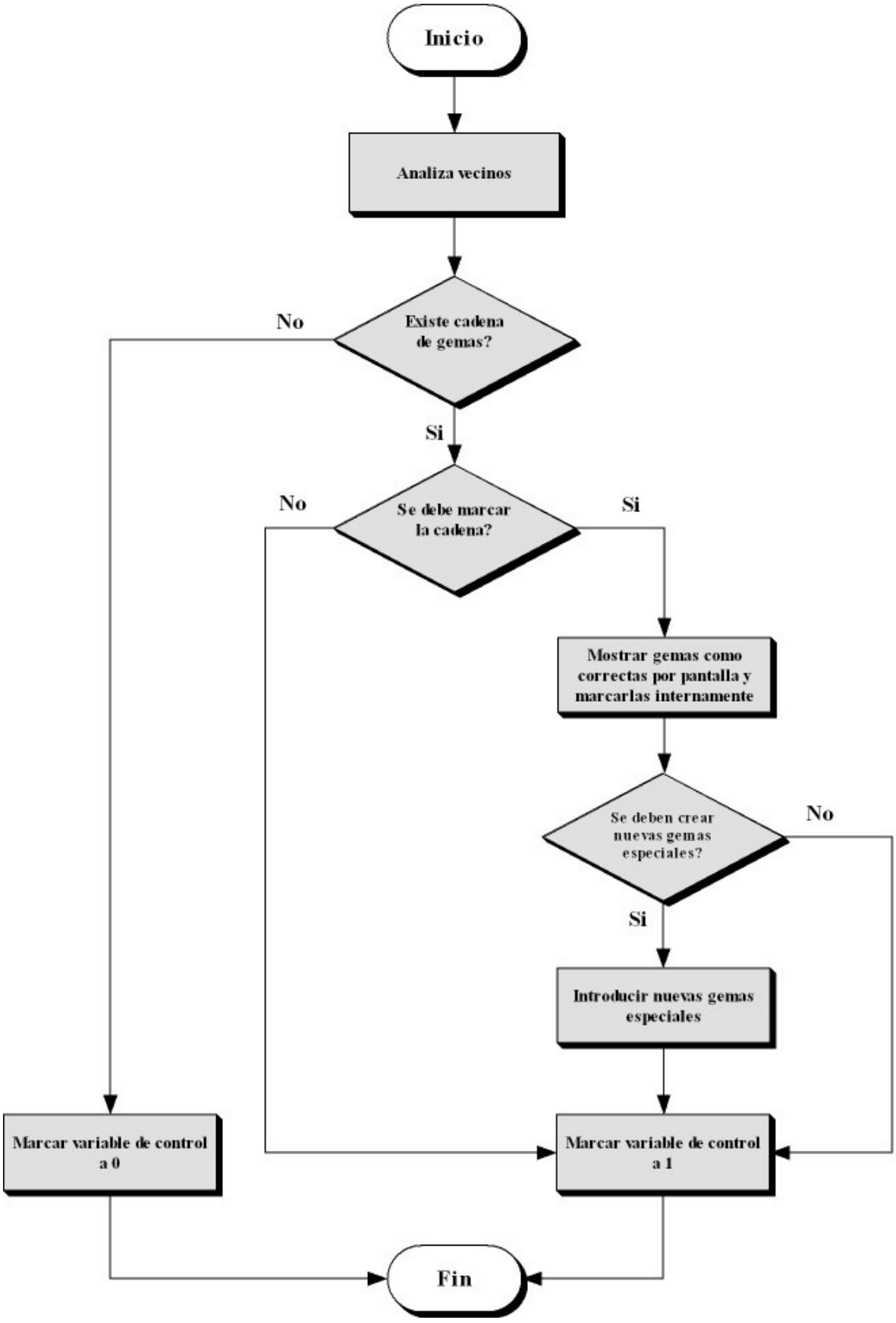


Figura 4.3.1 - diagrama de flujo del módulo *Verifica*.

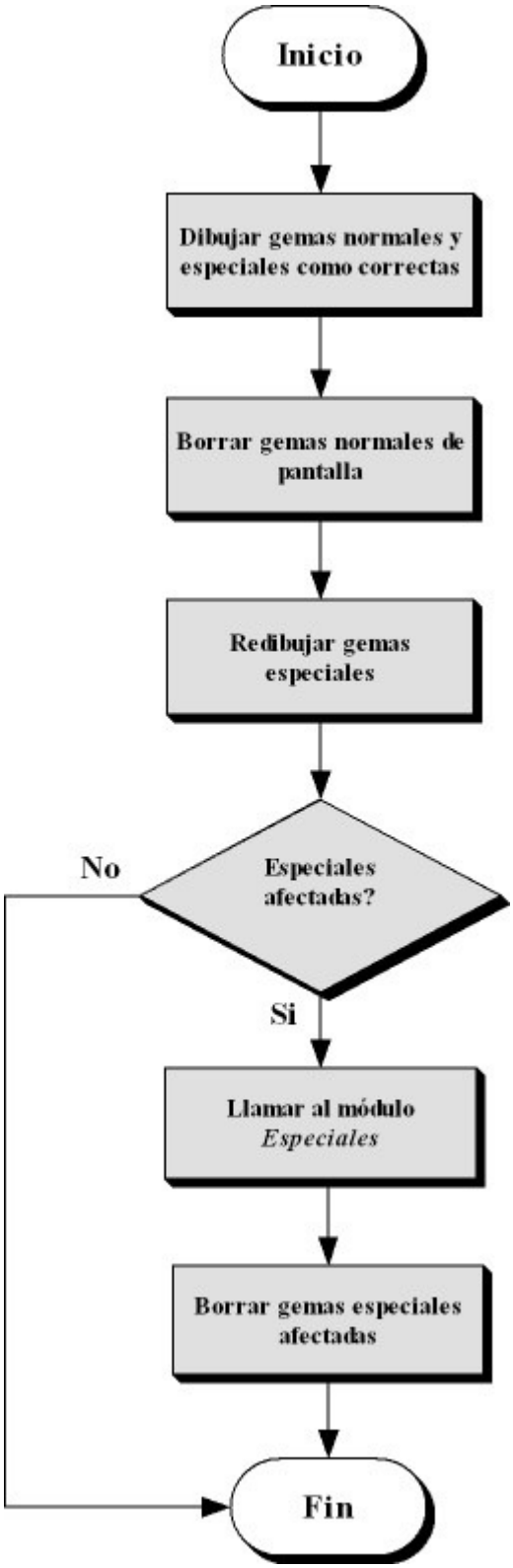


Figura 4.4.1 – Diagrama de flujo del módulo *Borragemas*.

4.5 Módulo *Caída*

Se llegará a este módulo siempre con huecos dentro del tablero de juego. Este se encargará de generar nuevas gemas y comprobar la viabilidad de este a través del módulo *Viable*. Después de esto pasaremos a dibujar la caída de las gemas, tanto antiguas como recién generadas. Con el fín de dibujar la caída de las gemas nuevas y las antiguas deberemos haber generado el nuevo tablero viable en una estructura de datos temporal desde donde se irán leyendo mientras se introducen en el tablero real.

En la imagen 4.5.1 se puede observar el diagrama de flujo del módulo.

4.6 Módulo *Viable*

Este módulo se encargará de comprobar la viabilidad del tablero. Será llamado en dos situaciones diferentes. La primera al iniciar el juego y crearse el tablero desde cero. En segundo lugar cuando se generan nuevas gemas y se rellenan con ellas los huecos de las gemas recién desaparecidas. Por cada casilla del tablero se llamará al módulo *Verifica*, hasta que la respuesta de este sea positiva. En ese caso se interrumpe el recorrido y se devuelve un valor positivo. En caso contrario se informará al módulo que le hubiera llamado de forma negativa. Se aprovechará que este módulo comprueba las gemas del tablero en busca de posibles cadenas para almacenar en una variable global la posición de la primera de ellas que se encuentre. Gracias a ello se tendrá siempre localizada un gema con posible movimiento correcto para cuando el usuario pulse el botón de ayuda.

El diagrama de flujo de este módulo queda representado en la imagen 4.6.1.

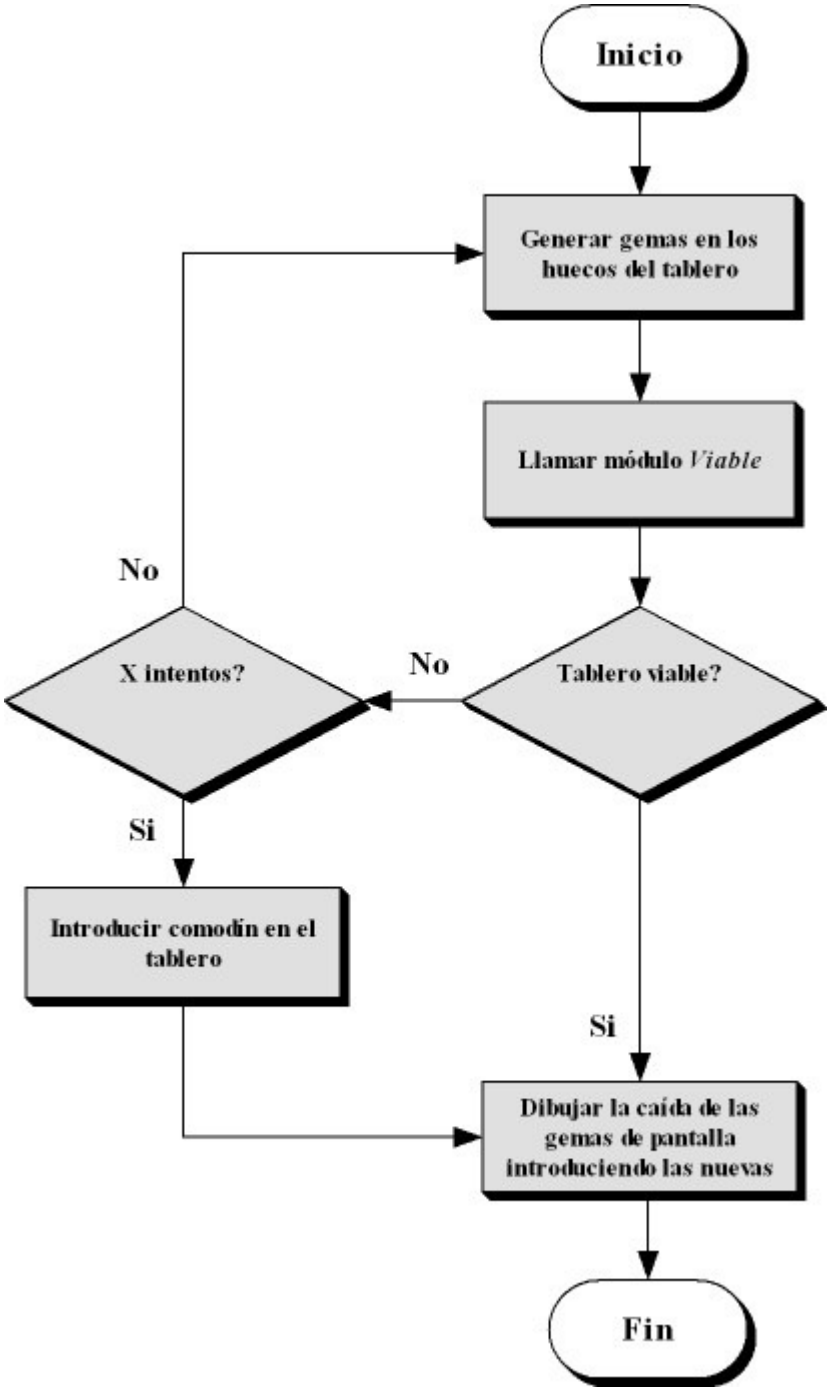


Figura 4.5.1 - Diagrama de flujo del módulo *Caída*.

4.7 Módulos generales

A continuación se explciarán algunos de los módulos menos importantes de la aplicación; otros serán omitidos dada la simplicidad de estos. Al final del apartado se podrán observar los diagramas de flujo de algunos de ellos.

- *ReaccionEnCadena*: tras la caída de las gemas en los huecos dejados por las desaparecidas, ejecutará un bucle que analizará en cada iteración si se han generado nuevas cadenas llamando a *Verifica* para cada posición del tablero. Si encuentra una sola cadena, se llamarán los módulos *Borragemas*, *Actualiza* y *Caida*. Tras esto comenzará de nuevo otra iteración. Cuando termine una sin encontrar cadenas finalizará la ejecución. Su diagrama de flujo esta representado por la imagen CCC.
- *Abandonar*: después de el usuario haya pulsado el botón de abandonar partida la aplicación deberá esperar una confirmación o una cancelación borrando el tablero de juego de la pantalla de forma provisional. En caso de cancelación este se redibujará volviendo al módulo principal y continuar el juego. En caso de confirmación se cambiará el valor de una variable de control que indicará al módulo principal que debe dejar de esperar pulsaciones de juego por parte del usuario. Se volverá entonces a esperar una señal de inicio de partida. Su diagrama de flujo de puede observar en la imagen DDD.
- *Especial*: este módulo será ejecutado cuando se encuentre una gema especial marcada como parte de una cadena. Comprobaremos las 8 gemas circundantes a la coordenada recibida por parámetro marcándola como correcta en caso de ser esta de tipo normal. En caso de ser otra gema especial, el módulo se llamará a sí mismo de forma recursiva con la coordenada de esta como parámetro. Por cada gema que desaparezca así se contabilizarán un número de puntos que al final la ejecución se devolverán a la función que la hubiera llamado.
- *Comodín*: En primer lugar se comprobará cual de las dos casillas seleccionadas contiene el comodín. Entonces comenzará el recorrido por el tablero en busca de todas las gemas que existan en juego del mismo tipo que la contenida en la

segunda casilla. Una vez localizadas primero se marcarán todas ellas como correctas por pantalla para seguidamente borrarlas tanto de la estructura de datos como de la misma pantalla. Este módulo también contabilizará cierto número de puntos por cada pieza así borrada, y los devolverá como parámetro.

- *GeneraTablero*: Por cada posición del tablero se genera una gema de forma aleatoria. Por cada una que se genera se comprueba que las dos precedentes, tanto en vertical como en horizontal, no sean del mismo tipo. Este módulo será llamado por *Inicializa* al comienzo del juego y debe configurar el tablero del tal forma que no hayan cadenas de gemas ya formadas en su inicio.

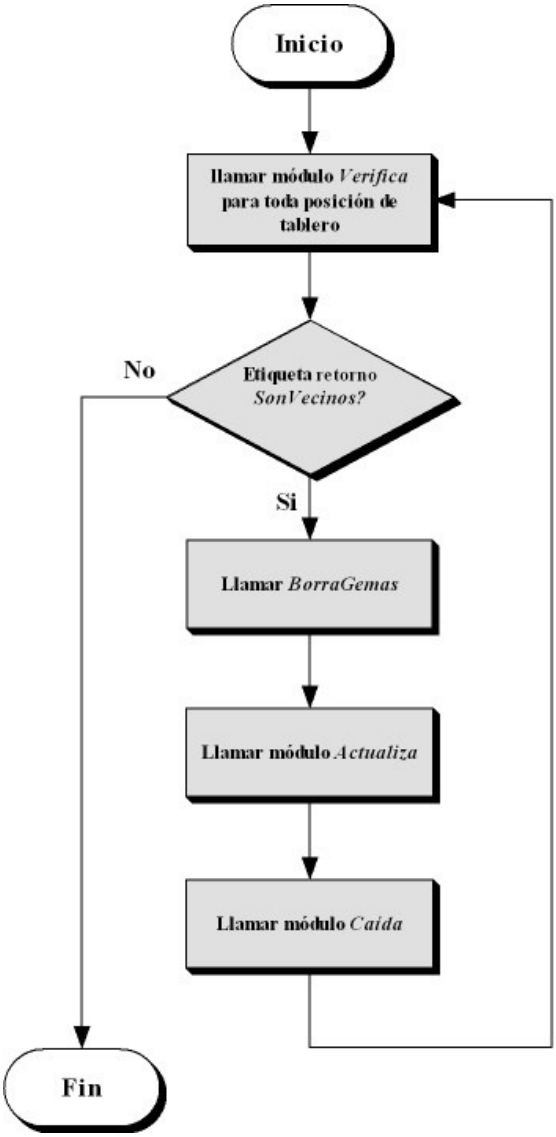


Figura 4.7.1 - Diagrama de flujo del módulo *ReaccionEnCadena*.

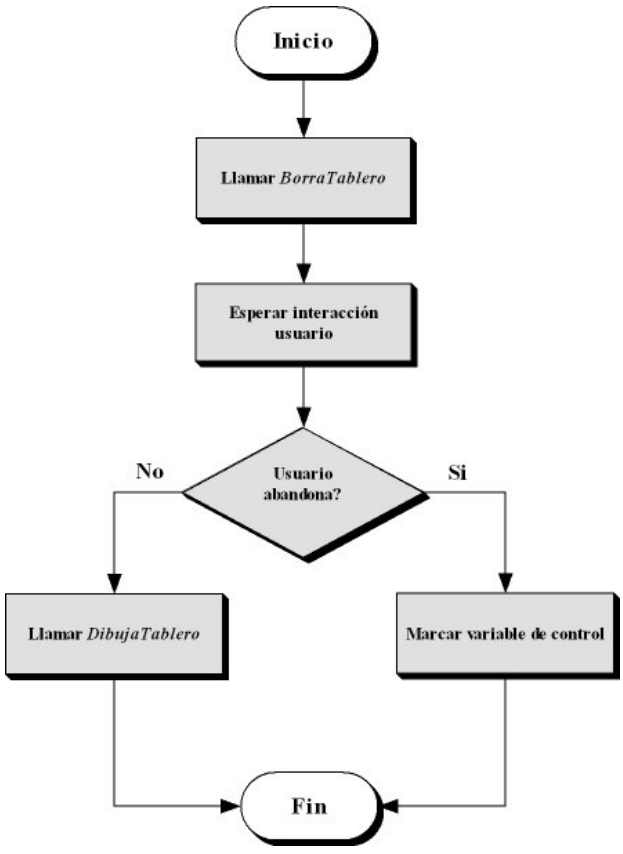


Figura 4.7.2 - Diagrama de flujo del módulo *Abandonar*

4.8 Solución *Amulet*

Toda la parte relativa a la pantalla a excepción de los dibujos de las piezas de juego serán gestionados por diferentes. Estos se incluyen el código Html como si de un applet java se tratara. Su código nos permite la ejecución de comandos que enviarán señales a través de su UART. Para la interacción HCS12/usuario hemos utilizado uno que enviará un código de *RPC (Request Procedure* Call) que especificaremos en el propio comando.

El código especificado deberá ser interpretado por la aplicación desde el microcontrolador. Solo algunos de los botones o pulsaciones del usuario en la pantalla deberán enviar un código ya que otras simplemente servirán para cambiar de una pantalla a otra.

4.8.1 Tablero de juego

Para la realización de el tablero de juego se han barajado diferentes soluciones y la decisión final ha influido, en mayor medida que el resto. en la solución de la parte correspondiente a nuestro *HCS12*. Se barajaron en total 3 posibilidades y se descartaron 2 de ellas por diferentes razones.

La primera opción consistía en definir en μHtml la carga de imágenes vacías en cada posición del tablero y definir maps sobre cada una de ellas para poder ejecutar llamadas a procedimiento remoto diferentes. La pantalla se limitaría entonces a enviar los códigos de dichos RPCs al HCS12 cada vez que una parte del tablero fuera seleccionada y dejaría a este la misión de dibujar cada una de las piezas en función de la partida. Para ello la pantalla cuenta con primitivas gráficas tales como líneas, rectángulos y simples píxeles que el microcontrolador puede dibujar en pantalla en 16 tonos diferentes, y en el caso de rectas y rectángulos con 16 tipos de grosor de línea. Esta opción otorga todo el trabajo, gráficamente hablando a nuestro microprocesador y nos obliga a realizar dibujos de gemas simples.

Para la segunda opción nuestra pantalla tiene capacidad para recibir vía *UART* imágenes gif animadas, en tiempo de ejecución, a través de un protocolo específico llamado *Xmodem*. Dicho protocolo incluye *CRC* y divide el archivo en paquetes de datos de 128 bytes y los encapsula en paquetes de 133 bytes a su vez incluyendo en dicho paquete el *CRC*, control y checksum. La idea consistía en enviar la imagen correspondiente a cada sección del tablero cuando el juego lo hiciera necesario. Teniendo en cuenta el protocolo descrito, necesitaríamos la transmisión de 1064 bytes por cada *KB* de información real. Si tenemos en cuenta el peor de los casos: conseguir encadenar 5 piezas del mismo tipo en la fila inferior con las de los extremos siendo de la clase especial, equivaldría a tener que modificar la imagen de 8 filas x 7 columnas. Este movimiento supondría tener que actualizar la imagen de 56 posiciones del tablero, y suponiendo un tamaño teórico de 1KB de tamaño por imagen, obligaría a enviar desde el *HCS12* un total de 59.584 bytes. Si configuramos la *UART* de la pantalla a máxima velocidad de funcionamiento esta funcionará a 115.200 *bps*. A esta velocidad el refresco de las posiciones del tablero, teniendo en cuenta el caso antes descrito, tardaría algo más de 4 segundos. Teniendo en cuenta que el cambio de gemas por pantalla se realizaría sin ningún tipo de movimiento no lo consideré aceptable.

Esta opción quedó así descartada pero obligó a insistir en la posibilidad de utilizar imágenes gif animadas por la estética que se podía conseguir gracias a ellas así como evitar realizar los dibujos de las piezas.

Gracias a un tipo de widget llamado *ImageSequence* definimos la tercera y última opción. Este widget consulta una variable interna y en función de su valor y del rango definido en su declaración carga en pantalla la imagen definida para ser mostrada en cada uno de los rangos. Definiendo un rango de 8 unidades se podría cambiar entonces desde el HCS12 el valor de un simple byte por cada casilla del tablero. Esto resolvería la representación gráfica de las piezas con tan solo cargar las 8 imágenes al programar la flash de la pantalla, y, en tiempo de ejecución modificando el valor de las variables asignadas a cada uno de los widgets. Al definir imágenes transparentes sobre los widgets y realizando un *map* sobre cada uno de ellos conseguiríamos que nuestro microcontrolador supiera en todo momento que casillas eran pulsadas.

Durante las pruebas en el laboratorio comprobé que en primer lugar los gifs animados resultan presentarse como estáticos al mostrarse a través de dichos widgets. Esto le quitó atractivo a esta opción, pero la sencillez de desarrollo que permitía hizo que continuara con ella. Sin embargo fue también desechada en el momento en que tras realizar con éxito pruebas sencillas intenté probarlo a nivel de tablero cargando en una sola página todos los widgets definidos para el resto de elementos junto con los 64 *ImageSequence* del tablero. El resultado fue un mensaje del compilador anunciándome insuficiente memoria en la pantalla.

La primera propuesta ha sido la desarrollada en la fase de implementación. A continuación se puede ver el código correspondiente a la casilla superior izquierda del tablero de juego.

```
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area0">
<MAP NAME="area0">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(0)">
```

En el anexo número tres se encuentra la relación de códigos *RPC* que se emitirán desde el Amulet así como el detalle de los elementos que lo lanzarán y a que pantalla de la aplicación pertenece cada uno de ellos. Destacar que se estos códigos se han asignado de manera que el código RPC recibido por el HC12 sirva como índice de la estructura de datos utilizada para representar las gemas de forma interna.

4.8.2 Resto de elementos

El resto de elementos de la pantalla de juego se han implementado utilizando diferentes widgets.

Lo botones de ayuda, abandonar partida, cancelar abandono y confirmar abandono serán los 4 *Function Buttons*. Todos ellos enviarán un código RPC al microprocesador HC12. Serán ellos mismos quienes se hagan invisibles a través de métodos de cada objeto. Es el caso del botón de abandono que hará aparecer los de confirmación y cancelación al ser pulsado.

La barra de progreso se ha implementado con un widget *BarGraph*. Los marcadores se han implementado con diferentes widgets. El marcador de nivel mediante un *Numeric Filed* y el de puntuación mediante un *String Field*. En el anexo número cuatro se encuentra el código de todas las pantallas de la aplicación.

4.9 Solución *HCS12*

Para la implementación de la parte del microcontrolador *HCS12* he optado por programar íntegramente el código en lenguaje *C*. Existía la posibilidad de poder programar parcial o íntegramente en lenguaje *ensamblador*, más cómodo a la hora de configurar ciertas partes *hardware* del MCU. A pesar de ello los recursos utilizados han sido configurados muy fácilmente gracias a los *beans* que incorpora la herramienta *Code Warrior*. Entre ellos he utilizado un contador, configurado con uno de los timers disponibles, así como un *AsynchroSerial bean* que implementa un puerto de comunicaciones asíncrona. Este ha sido configurado a través del puerto *SC0* tal y como se especificó en la etapa de análisis.

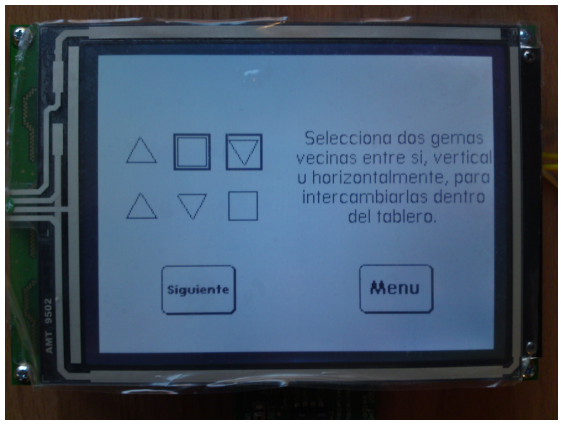
Dadas las restricciones que existen en cuanto a memoria y potencia de cálculo se ha implementado la solución pensando en realizar menos llamadas a funciones de las que serían deseables para conseguir un código fluido y comprensible. Existen funciones muy grandes y existe código duplicado en algunas de ellas, pero todo ello ha sido hecho de forma consciente para intentar evitar el anidamiento de muchas funciones.
También a causa de estas restricciones no se ha implementado una función que busque y decida que gemas nuevas introducir en el tablero para asegurar su viabilidad. Las nuevas gemas se introducen de forma aleatoria. En el caso de llegar a cierto número de intentos de conseguir un tablero viable introduciremos un comodín en el juego para asegurar dicha viabilidad. Debo comentar que a lo largo de todas las pruebas realizadas descritas en el apartado siguiente nunca se ha dado el caso de que el tablero generado de forma aleatoria no fuera viable.

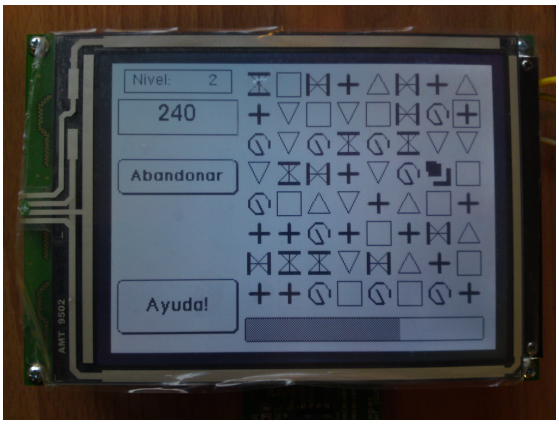
El código fuente se encuentra en el anexo número cinco

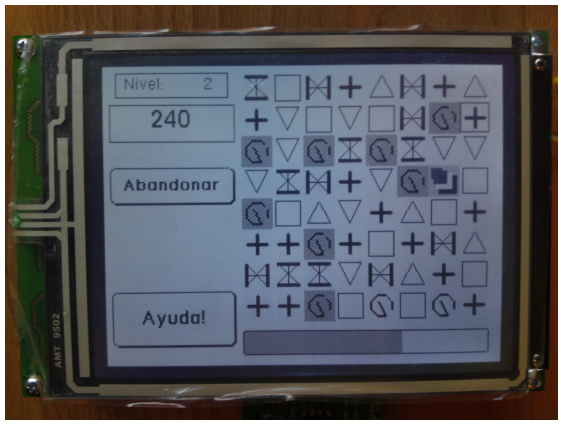
.

4.10 Pruebas

Para comprobar el correcto funcionamiento de la aplicación se han realizado pruebas de los módulos a lo largo del la implementación así como pruebas generales al producto total. Se han introducido manualmente diferentes configuraciones de tablero, de puntuación y de progreso para observar su comportamiento. También se ha comprobado el funcionamiento general realizando partidas más o menos largas. Todas las pruebas de la versión final funcionan correctamente. En las dos imágenes siguientes se puede observar el menú de juego y una de las pantallas de instrucciones. En las otras dos se pueden apreciar el estado del tablero con un comodín en juego y el mismo tablero mientras se procesa este mismo comodín y todo un tipo de gemas.







Imágenes 4.10.1, 4.10.2, 4.10.3 y 4.10.4 – Diferentes fotos del juego

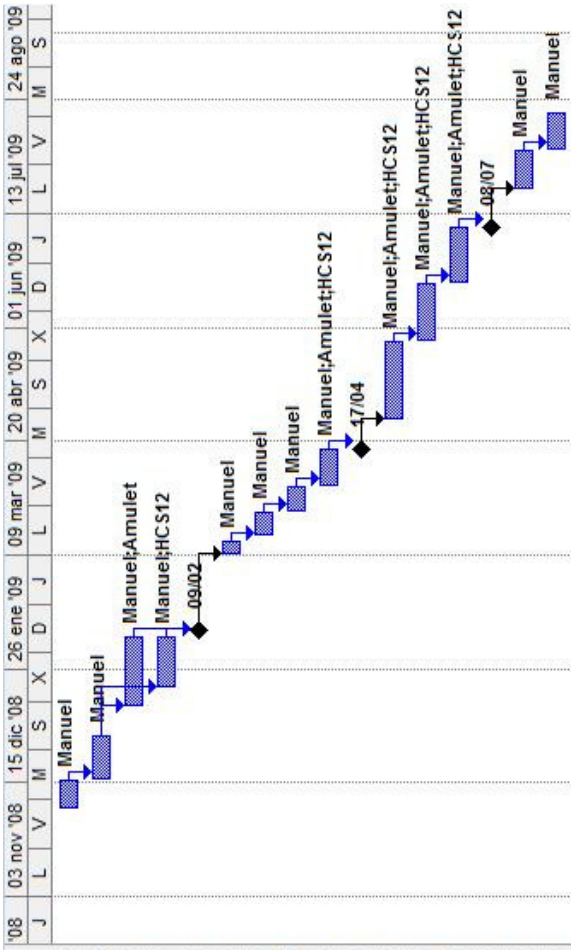
4.11 Problemas encontrados

La realización del proyecto viene marcada por un mal inicio. Conseguir comunicación, de cualquier tipo, entre los dispositivos fue el punto crítico que marcó el poder continuar con el análisis del material disponible, así como el tomar la decisión del juego a desarrollar. A lo largo de todas las pruebas no conseguí ningún tipo de comunicación. Tras diferentes consultas me di cuenta de que se sin conectar el hilo de tierra los dispositivos no tenían punto de referencia para la diferencia de potencial. Una vez conectados ambos pines la comunicación bidireccional funcionó a la perfección.

Otro de los grandes problemas vino al comenzar a probar el envío de comandos gráficos a la pantalla para dibujar las gemas. Tal y como se puede ver en el anexo número tres los datos deben enviarse en formato *ASCII* para ser interpretados correctamente. Al realizar la conversión utilizaba la función de *C sprintf*. Esta convierte una variable, de la forma que se especifique por medio de sus parámetros, en una cadena de texto. En mi caso convertía variables enteras en formato hexadecimal a formato cadena de texto. Al realizar las pruebas de todo el código que realizaba me encontré con que las posiciones de memoria del contenido de las casillas del tablero perdían su valor al cabo de cierto tiempo de ejecución. Dependiendo de la prueba los valores variaban sin aparente sentido. Tras revisar toda la parte de código donde se modificaban esos valores descarté que se tratara de un error en ese sentido. Tras descartar algún puntero mal diseccionado busqué el error por medio del depurador del CodeWarrior[14]. Finalmente encontré las líneas de código culpables de la perdida de datos: la función *sprintf*. Tras varias investigaciones descubrí que dicha función guarda en la pila grandes buffers en función de sus parámetros. Esto en dispositivos como el nuestro, con poca memoria, puede llegar a provocar desbordamientos. La solución al problema fue la implementación de una función a medida que realizara la misma funcionalidad que utilizaba de la anterior, así como aumentar el espacio de memoria dedicado a la pila. Dicha función se puede ver en el código fuente incluido en el anexo número cinco con nombre *Traduce*. La función Sprintf se ha seguido

utilizando para la conversión de los datos a enviar al widget *String Field* que muestra la puntuación y sin embargo este no ha dado ningún tipo de problema.

Aparte de los errores que se pueden considerar normales a la hora de implementar código, que son fáciles de identificar y corregir, los dos anteriores son los que han provocado una mayor demora en la realización del proyecto. El primero de ellos se puede achacar a la falta de experiencia con este tipo de dispositivos e interconexiones. El segundo fue provocado por utilizar funciones que en dispositivos con más recursos, como puede ser un PC, funcionan a la perfección. En el siguiente diagrama se puede ver la diferencia temporal entre la previsión temporal inicial y la final.



Nombre de tarea	Duración	Comienzo	Fin
Recopilación documentos de los dispositivos	7 días	vie 05/12/08	lun 15/12/08
Estudio de la documentación	12 días	mar 16/12/08	mié 31/12/08
Pruebas con Amuet	20 días	lun 12/01/09	vie 06/02/09
Pruebas con HCS12	15 días	lun 19/01/09	vie 06/02/09
Conseguir comunicación entre dispositivos	20 días	lun 09/02/09	vie 06/03/09
Decidir juego a diseñar	5 días	lun 09/03/09	vie 13/03/09
Redacción introducción y objetivos Memoria PFC	7 días	lun 16/03/09	mar 24/03/09
Comenzar diseño	7 días	mié 25/03/09	jue 02/04/09
Primeras pruebas	10 días	vie 03/04/09	jue 16/04/09
Diseño definitivo	7 días	vie 17/04/09	lun 27/04/09
Implementación	21 días	mar 28/04/09	mar 26/05/09
Pruebas	15 días	mié 27/05/09	mar 16/06/09
Corrección de errores	15 días	mié 17/06/09	mar 07/07/09
Pruebas finales	10 días	mié 08/07/09	mar 21/07/09
Redacción Memoria PFC completa	10 días	mié 22/07/09	mar 04/08/09
Correcciones sobre Memoria de PRC	10 días	mié 05/08/09	mar 18/08/09

Figura 4.11.1 – Diagrama de Gant real

CAPÍTULO 5: Conclusiones

Llegados a este punto se debe volver atrás y repasar los objetivos del proyecto. Se ha conseguido diseñar e implementar un juego sencillo, utilizando los recursos disponibles. Este es plenamente operativo y asegura tantas horas de entretenimiento como el usuario decida jugar. Se puede afirmar en este punto que se han conseguido todos los objetivos planteados inicialmente.

A nivel personal estoy satisfecho con el trabajo realizado. A pesar de ser un juego sencillo, gráficamente pobre para lo que podríamos estar acostumbrados hoy día, se ha conseguido un nivel de jugabilidad bueno. La velocidad de movimiento de las piezas se ha ajustado para que no resulte demasiado rápido para la vista, ni tan lento como para resultar tedioso. También hay que decir que para la implementación de este se han utilizado diferente recursos disponibles en ambos dispositivos, obteniendo así mejor resultado. También quiero destacar la experiencia adquirida durante la realización el proyecto. Si hubiera comenzado su con los conocimientos que he ido reuniendo puedo decir que tardaría considerablemente menos de lo que he tardado.

Como nota negativa resaltar algún que otro aspecto. Como síntesis se pueden resaltar dos palabras: limitaciones físicas. El material disponible puede resultar suficiente para realizar prácticas, siendo estas sencillas. Ha sido suficiente para poder implementar con él este juego, pero a lo largo de su realización me he encontrado con problemas que en otras circunstancias no habría tenido. Ello se ha debido a causa de contar con poca memoria. Se ha debido hacer cierta previsión de posibles desbordamientos de pila, que en otras circunstancias se podrían haber evitado, aumentando su tamaño directamente así como evitando implementar muchas llamadas a funciones, o cálculos complejos y largos. Esto ha influido también en el juego elegido para desarrollar así como de la simplicidad gráfica de las piezas de juego. Por otro lado, se puede decir lo mismo de la memoria de nuestra pantalla. Al intentar desarrollar el juego de la mejor manera, gráficamente hablando, me he encontrado con que ni siquiera podía probar un tablero completo con esa solución porque el compilador me alertaba de que había memoria insuficiente en el módulo. El aumento del tamaño en ambos dispositivos abriría

nuevas posibilidades a la hora de desarrollar juegos, o aplicaciones en general, de calidad superior al que aquí se está exponiendo.

Cabe destacar que el uso de este tipo de dispositivos para la realización de prácticas en laboratorio sería sin duda un gran aliciente para los estudiantes. Estos, acostumbrados a conseguir que un led se encienda o a leer un par de palabras recorriendo un display verían con gran interés el desarrollo de la práctica y la realizarían con mucho más entusiasmo que el actual. Para ello incluso propondría utilizar este mismo proyecto como base a la planificación y desarrollo de dichas prácticas, en las que se podría aportar una base sobre la que trabajar y reducir así drásticamente el tiempo de desarrollo de estas. Que el alumno pudiera elegir incluso el tipo de aumentaría el valor pedagógico de las prácticas al entrar en juego la creatividad personal y la creatividad del grupo.

Como conclusión final decir que este proyecto se podría ampliar en un futuro pudiendo realizar mejoras sobre el actual. Dichas mejoras serían ampliaciones funcionales realizadas a través de diferentes dispositivos. El juego se podría completar, por poner un par de ejemplos, con unos altavoces conectados al microcontrolador, en los que poder oír una melodía o simples sonidos provocados por los movimientos de las gemas, mientras se juega. Asimismo se podría también conectar un *joystick* con el que poder controlar la aplicación a pesar de que sería en detrimento de la característica táctil de la pantalla,

CAPÍTULO 6: Bibliografía

[1] http://www.intel.com/museum/archives/4004.htm

[2] http://download.intel.com/museum/archives/pdf/4004_datasheet.pdf

[3] http://www.zilog.com/docs/z80/um0080.pdf

[4] http://www.freescale.com/webapp/sps/site/overview.jsp?nodeId=06258A25802570256D
http://www.softecmicro.com/downloads/doc/indart-hcs12_manual.pdf

[5] http://www.adese.es/web/main.asp

[6] http://www.amulettechnologies.com/

[7] http://www.amulettechnologies.com/pdf/_brochure_ds/dataSheet.pdf

[8] http://www.amulettechnologies.com/pdf/_brochure_ds/RS232_Interface_DS.pdf

[9] http://www.softecmicro.com/downloads/doc/pk-hcs12e128_poster.pdf
http://www.softecmicro.com/downloads/doc/pk-hcs12e128_poster.pdf

[10] http://www.freescale.com/files/microcontrollers/doc/data_sheet/MC9S12E128V1.pdf

[11] http://www.freescale.com/

[12] http://www.amulettechnologies.com/pdf/_brochure_ds/RS232_Interface_DS.pdf

[13] http://www.popcap.com/

[14] http://www2.metrowerks.com/

ANEXOS

Anexo 1: Listado de comandos *Amulet* y códigos *MSOM*

La siguiente tabla muestra el listado de todos los comandos que se pueden enviar los dispositivos. La columna *Command Opcode* indica la cabecera *MSOM* del mensaje que contendrá ducho comando.

Command Opcode	Response Opcode	Description	Command Can Be Sent by Amulet	Command Can Be Sent by External Processor
0xD0	0xE0	Get byte variable	X	X
0xD1	0xE1	Get word variable	X	X
0xD2	0xE2	Get string variable	X	X
0xD3	0xE3	Get label variable	X	
0xD4	0xE4	Get RPC buffer		X
0xD5	0xE5	Set byte variable	X	X
0xD6	0xE6	Set word variable	X	X
0xD7	0xE7	Set string variable	X	X
0xD8	0xE8	Invoke RPC	X	
0xD9	0xE9	Draw Line		X
0xDA	0xEA	Draw Rectangle		X
0xDB	0xEB	Draw Filled Rectangle		X
0xDC	0xEC	Draw Pixel		X
0xDD	0xED	Get byte variable array	X	
0xDE	0xEE	Get word variable array	X	
0xDF	0xEF	Set byte variable array		X
	0xF0	Acknowledgment (ACK)	X	X
	0xF1	Negative Acknowledgment (NAK)		X
0xF2	0xF3	Set word variable array		X

Figura A.1 - Relación de comandos *MSOM*

Anexo 2: Detalle de comunicación del módulo *Amulet*

Se trata de un protocolo ASCII que enviará 2 bytes por cada uno de información real que se quiera enviar. Cada uno de estos bytes se llama *nibble*. En el ejemplo siguiente se detalla el protocolo.

Si el HC12 pretende cambiar el valor de una variable de 16 bits de la pantalla deberá enviar en primer lugar el *MSOM* del comando; este se puede consultar en la tabla del anexo anterior y es "0xD6". Después se enviará el indice de la variable a modificar. Si se trata de la variable número 1 deberemos enviar los bytes "0x30" y "0x31" ya que la primera representa el cero y la segunda el uno en código ASCII. Si el valor que se le quiere dar a esta variable es 4831 deberemos enviar los dos siguientes bytes: "0x12" y "0xDF" ya que 4831 en equivale a 0x12DF en hexadecimal.

Cuando la pantalla actúa como esclava el procesador deberá responder con un eco del mensaje original para demostrar la correcta recepción de los datos. Lo único que deberá cambiar será el *MSOM* poniendo el correspondiente al comando original seguido del resto de los datos.

A continuación un ejemplo:

Amulet LCD Module	Dir.	External Processor	Description
0xD0 0x30 0x31 '0' '1'	>>		Get byte variable 1
	<<	0xE0 0x30 0x31 0x33 0x38 '0' '1' '3' '8'	Return byte of byte var 1
0xD2 0x30 0x31 '0' '1'	>>		Get string variable 1
	<<	0xE2 0x30 0x31 0x41 0x62 0x63 0x00 '0' '1' 'A' 'b' 'c'	Return string of string var 1
0xD5 0x30 0x31 0x46 0x45 '0' '1' 'F' 'E'	>>		Set byte variable 1 to 0xFE
	<<	0xE5 0x30 0x31 0x46 0x45 '0' '1' 'F' 'E'	Confirm set byte variable 1

Figura A.2 - Ejemplo de comunicación

Anexo 3: Relación de códigos *RPC* de la pantalla

Pantalla	Objeto	#RPC	Pantalla	Objeto	#RPC	Pantalla	Objeto	#RPC
Partida	gema 1	0	Partida	gema 31	30	Partida	gema 61	60
Partida	gema 2	1	Partida	gema 32	31	Partida	gema 62	61
Partida	gema 3	2	Partida	gema 33	32	Partida	gema 63	62
Partida	gema 4	3	Partida	gema 34	33	Partida	gema 64	63
Partida	gema 5	4	Partida	gema 35	34	Menú	Jugar	64
Partida	gema 6	5	Partida	gema 36	35	Partida	Ayuda	65
Partida	gema 7	6	Partida	gema 37	36	Partida	Abandonar	66
Partida	gema 8	7	Partida	gema 38	37	Partida	Abandona Si	67
Partida	gema 9	8	Partida	gema 39	38	Partida	Abandona No	68
Partida	gema 10	9	Partida	gema 40	39			
Partida	gema 11	10	Partida	gema 41	40			
Partida	gema 12	11	Partida	gema 42	41			
Partida	gema 13	12	Partida	gema 43	42			
Partida	gema 14	13	Partida	gema 44	43			
Partida	gema 15	14	Partida	gema 45	44			
Partida	gema 16	15	Partida	gema 46	45			
Partida	gema 17	16	Partida	gema 47	46			
Partida	gema 18	17	Partida	gema 48	47			
Partida	gema 19	18	Partida	gema 49	48			
Partida	gema 20	19	Partida	gema 50	49			
Partida	gema 21	20	Partida	gema 51	50			
Partida	gema 22	21	Partida	gema 52	51			
Partida	gema 23	22	Partida	gema 53	52			
Partida	gema 24	23	Partida	gema 54	53			
Partida	gema 25	24	Partida	gema 55	54			
Partida	gema 26	25	Partida	gema 56	55			
Partida	gema 27	26	Partida	gema 57	56			
Partida	gema 28	27	Partida	gema 58	57			
Partida	gema 29	28	Partida	gema 59	58			
Partida	gema 30	29	Partida	gema 60	59			

Figura A.3 - Relación de *RPC*s

Anexo 4: Código Amulet

A continuación se listará el código de los diferentes archivos Html y cargados en el módulo Amulet, a excepción de los de instrucciones de juego al ser todo código básico.

4.1 *Intro.html*

```
<HTML>
<HEAD>
<META NAME="Amulet" Content="Baud.Project=19200">
<META NAME="Amulet" Content="SlaveNoRsp.Project">
<META NAME="Amulet" Content="UARTDelay">
<META NAME="initInternalRAM" SRC="valores.ini">
</HEAD>
<body>
<div style="position:absolute; top:0px; left:0px;width:320px;height:240px;background-image:url('images/intro.gif');">
<table width="320" height="240" border="0" cellpadding="0" >
  <tr>
    <td align="center" valign="bottom">
    <APPLET CODE="FunctionButton.class" WIDTH="160" HEIGHT="40" NAME="ButtonIntro">
    <PARAM NAME="href" VALUE="Menu.html">
    <PARAM NAME="fontSize" VALUE="3">
    <PARAM NAME="fontStyle" VALUE="BOLD">
    <PARAM NAME="label" VALUE="Continuar">
    <PARAM NAME="buttonType" VALUE="SPRING-LOADED">
    <PARAM NAME="onButtonPress" VALUE="DEPRESS">
    </APPLET>
    </td>
  </tr>
  <tr>
    <td height="30">
    </td>
  </tr>
</table>
</div>
</BODY>
</HTML>
```

4.2 Menu.html

```
<HTML>
<HEAD>
<META NAME="Amulet" Content="Baud.Project=19200">
<META NAME="Amulet" Content="SlaveNoRsp">
<META NAME="Amulet" Content="UARTDelay">
</HEAD>
<body>
<div style="position:absolute; top:0px; left:0px;width:320px;height:240px;background-image:url('images/intro.gif');">
<table width="320" height="240" border="0" cellpadding="0" >
 <tr>
  <td width="320" height="20">
  </td>
 </tr>
 <tr>
 <td>
  <table width="320" height="180" border="0" cellpadding="0" >
   <tr>
          <td width="160" height="180" align="center" valign="bottom">
          <APPLET CODE="FunctionButton.class" WIDTH="60" HEIGHT="40" NAME="Button1">
          <PARAM NAME="href" VALUE="Amulet:UART.invokeRPC(64), Juego.html;">
          <PARAM NAME="fontSize" VALUE="3">
          <PARAM NAME="fontStyle" VALUE="BOLD">
          <PARAM NAME="label" VALUE="Jugar">
          <PARAM NAME="buttonType" VALUE="SPRING-LOADED">
          <PARAM NAME="onButtonPress" VALUE="DEPRESS">
          </APPLET>
          </td>

          <td width="160" height="180" align="center" valign="bottom">
          <APPLET CODE="FunctionButton.class" WIDTH="60" HEIGHT="40" NAME="Button2">
          <PARAM NAME="href" VALUE="Help01.html;">
          <PARAM NAME="fontSize" VALUE="3">
          <PARAM NAME="fontStyle" VALUE="BOLD">
          <PARAM NAME="label" VALUE="Help">
          <PARAM NAME="buttonType" VALUE="SPRING-LOADED">
          <PARAM NAME="onButtonPress" VALUE="DEPRESS">
                  </APPLET>
          </td>


   </tr>
  </table>
 </td>
</tr>
 <tr>
   <td width="320" height="20" align="center" valign="middle">
  </td>
</tr>
```

```
</table>
</div>
</BODY>
</HTML>
```

4.3 Juego.html

```
<HTML>
<HEAD>
<META NAME="Amulet" Content="Baud.Project=19200">
<META NAME="Amulet" Content="SlaveNoRsp">
<META NAME="Amulet" Content="UARTDelay">
</HEAD>
<body>
<table width="320" height="240" border="0" cellpadding="0" cellspacing="0">
 <tr>
        <td width="110" height="240" align="center" valign="middle">
        <table width="110" height="240" border="0" cellpadding="0" cellspacing="0" align="left">
        <tr>
        <td width="110" height="5">
        </td>
        </tr>
        <tr>
        <td width="100" height="20" align="center" valign="middle">
                <APPLET CODE="NumericField.class" WIDTH="90" HEIGHT="20" NAME="Nivel">
                <PARAM NAME="href" VALUE="Amulet:InternalRAM.word(1).value()">
                <PARAM NAME="fontSize" VALUE="2">
                <PARAM NAME="fontStyle" VALUE="PLAIN">
                <PARAM NAME="horizontalAlign" VALUE="CENTER">
                <PARAM NAME="border" VALUE="1">
                <PARAM NAME="min" VALUE="0">
                <PARAM NAME="max" VALUE="65534">
                <PARAM NAME="minFld" VALUE="0">
                <PARAM NAME="maxFld" VALUE="65534">
                <PARAM NAME="printf" VALUE="Nivel: %5i ">
                <PARAM NAME="updateRate" VALUE="1.0,0.01">
                </APPLET>
        </td>
        </tr>
        <tr>
        <td width="100" height="5">
        </td>
        </tr>
        <tr>
        <td width="100" height="30" align="center" valign="middle">
                <APPLET CODE="StringField.class" WIDTH="100" HEIGHT="30" NAME="Puntuacion">
                <PARAM NAME="href" VALUE="Amulet:InternalRAM.string(0).value()">
                <PARAM NAME="fontSize" VALUE="4">
                <PARAM NAME="fontStyle" VALUE="PLAIN">
```

```
            <PARAM NAME="horizontalAlign" VALUE="CENTER">
            <PARAM NAME="border" VALUE="1">
            <PARAM NAME="printf" VALUE="%s">
            <PARAM NAME="updateRate" VALUE="1,0.1">
            </APPLET>
</td>
</tr>
<tr>
<td width="100" height="20">
</td>
</tr>
<tr>
<td width="100" height="30" align="center" valign="middle">
            <APPLET CODE="FunctionButton.class" WIDTH="100" HEIGHT="30" NAME="Abandonar">
            <PARAM NAME="href" VALUE="Amulet:UART.invokeRPC(66), Amulet:document.BotonSi.reappear(),
                  Amulet:document.BotonNo.reappear(), Amulet:document.BotonAyuda.disappear();">
            <PARAM NAME="fontSize" VALUE="2">
            <PARAM NAME="fontStyle" VALUE="BOLD">
            <PARAM NAME="label" VALUE="Abandonar">
            <PARAM NAME="buttonType" VALUE="SPRING-LOADED">
            <PARAM NAME="onButtonPress" VALUE="DEPRESS">
            </APPLET>
</td>
</tr>
<tr>
<td width="100" height="20">
</td>
</tr>
<tr height="30" align="center" valign="middle">
<td width="100" height="30" align="center" valign="middle">
<table width="100" height="30" align="center">
<tr>
<td width="50" height="30" align="center" valign="middle">
             <APPLET CODE="FunctionButton.class" WIDTH="40" HEIGHT="30" NAME="BotonSi">
             <PARAM NAME="href" VALUE="Amulet:UART.invokeRPC(67), Menu.html">
             <PARAM NAME="fontSize" VALUE="2">
             <PARAM NAME="fontStyle" VALUE="BOLD">
             <PARAM NAME="label" VALUE="Si">
             <PARAM NAME="buttonType" VALUE="SPRING-LOADED">
             <PARAM NAME="onButtonPress" VALUE="DEPRESS">
             <PARAM NAME="invisible" VALUE="TRUE">
             </APPLET>
</td>
<td width="50" height="30" align="center" valign="middle">
             <APPLET CODE="FunctionButton.class" width="40" height="30" name="BotonNo">
             <PARAM NAME="href" value="Amulet:UART.invokeRPC(68), Amulet:document.BotonSi.disappear(),
                        Amulet:document.BotonNo.disappear(),  Amulet:document.Abandonar.reappear(),
                        Amulet:document.BotonAyuda.reappear();">
            <ARAM NAME="fontSize" value="2">
            <ARAM NAME="fontStyle" value="BOLD">
             <ARAM NAME="label" value="No">
```

```
            <ARAM NAME="buttonType" value="SPRING-LOADED">
            <ARAM NAME="onButtonPress" value="DEPRESS">
            <PARAM NAME="invisible" VALUE="TRUE">
            </APLELT>
</td>
</tr>
</table>
</td>
</tr>
<tr height="20">
<td width="100" height="20">
</td>
</tr>
<tr height="45" align="center" valign="middle">
<td width="100" height="45" align="center" valign="middle">
            <APPLET CODE="FunctionButton.class" WIDTH="100" HEIGHT="45" NAME="BotonAyuda">
            <PARAM NAME="href" VALUE="Amulet:UART.invokeRPC(65)">
            <PARAM NAME="fontSize" VALUE="3">
            <PARAM NAME="fontStyle" VALUE="BOLD">
            <PARAM NAME="label" VALUE="Ayuda!">
            <PARAM NAME="buttonType" VALUE="SPRING-LOADED">
            <PARAM NAME="onButtonPress" VALUE="DEPRESS">
            </APPLET>
</td>
</tr>
<tr height="5">
<td width="100" height="5">
</td>
</tr>
</table>
</td>
<td width="200" height="240" align="center" valign="middle">
<table width="200" height="5" border="0" cellpadding="0" cellspacing="0">
<tr>
<td>
</td>
</tr>
</table>
<table width="200" height="200" border="0" cellpadding="0" cellspacing="0">
<tr>
<td width="25" height="25">
            <CENTER>
            <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
            USEMAP="#area0">
            <MAP NAME="area0">
            <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(0)">
            </MAP>
            </CENTER>
</td>
<td width="25" height="25">
<CENTER>
```

```
        <IMG SRC="images/casilla.gif" WIDTH="25" HEIGHT="25" hspace="0" vspace="0" BORDER="0"
        USEMAP="#area1">
        <MAP NAME="area1">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(1)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area2">
        <MAP NAME="area2">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(2)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area3">
        <MAP NAME="area3">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(3)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area4">
        <MAP NAME="area4">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(4)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area5">
        <MAP NAME="area5">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(5)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area6">
        <MAP NAME="area6">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(6)">
        </MAP>
</CENTER>
```

```
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area7">
        <MAP NAME="area7">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(7)">
        </MAP>
</CENTER>
</td>
</tr>
<tr>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area8">
        <MAP NAME="area8">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(8)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area9">
        <MAP NAME="area9">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(9)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area10">
        <MAP NAME="area10">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(10)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area11">
        <MAP NAME="area11">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(11)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
```

```
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area12">
<MAP NAME="area12">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(12)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area13">
<MAP NAME="area13">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(13)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area14">
<MAP NAME="area14">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(14)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area15">
<MAP NAME="area15">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(15)">
</MAP>
</CENTER>
</td>
</tr>
<tr>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area16">
<MAP NAME="area16">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(16)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area17">
<MAP NAME="area17">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(17)">
```

```
            </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
            <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
            USEMAP="#area18">
            <MAP NAME="area18">
            <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(18)">
            </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
            <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
            USEMAP="#area19">
            <MAP NAME="area19">
            <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(19)">
            </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
            <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
            USEMAP="#area20">
            <MAP NAME="area20">
            <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(20)">
            </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
            <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
            USEMAP="#area21">
            MAP NAME="area21">
            <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(21)">
            </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
            <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
            USEMAP="#area22">
            <MAP NAME="area22">
            <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(22)">
            </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
```

```
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area23">
<MAP NAME="area23">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(23)">
</MAP>
</CENTER>
</td>
</tr>
<tr>
 <td width="25" height="25">
 <CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area24">
<MAP NAME="area24">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(24)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area25">
<MAP NAME="area25">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(25)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area26">
<MAP NAME="area26">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(26)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area27">
<MAP NAME="area27">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(27)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area28">
<MAP NAME="area28">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(28)">
```

```
		</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
		<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
		USEMAP="#area29">
		<MAP NAME="area29">
		<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(29)">
		</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
		<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
		USEMAP="#area30">
		<MAP NAME="area30">
		<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(30)">
		</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
		<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
		USEMAP="#area31" >
		<MAP NAME="area31">
		<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(31)">
		</MAP>
</CENTER>
</td>
</tr>
<tr>
<td width="25" height="25">
<CENTER>
		<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
		USEMAP="#area32">
		<MAP NAME="area32">
		<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(32)">
		</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
		<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
		USEMAP="#area33">
		<MAP NAME="area33">
		<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(33)">
		</MAP>
</CENTER>
</td>
<td width="25" height="25">
```

```
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area34">
        <MAP NAME="area34">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(34)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area35" >
        <MAP NAME="area35">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(35)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area36">
        <MAP NAME="area36">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(36)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area37">
        <MAP NAME="area37">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(37)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area38">
        <MAP NAME="area38">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(38)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area39">
        <MAP NAME="area39">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(39)">
        </MAP>
```

```
</CENTER>
</td>
</tr>
<tr>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area40">
        <MAP NAME="area40">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(40)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area41">
        <MAP NAME="area41">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(41)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area42">
        <MAP NAME="area42">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(42)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area43">
        <MAP NAME="area43">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(43)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area44">
        <MAP NAME="area44">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(44)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
```

```
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area45">
<MAP NAME="area45">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(45)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area46"
<MAP NAME="area46">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(46)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area47">
<MAP NAME="area47">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(47)">
</MAP>
</CENTER>
</td>
<tr>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area48">
<MAP NAME="area48">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(48)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area49">
<MAP NAME="area49">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(49)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area50">
<MAP NAME="area50">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(50)">
</MAP>
```

```
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area51">
        <MAP NAME="area51">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(51)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area52">
        <MAP NAME="area52">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(52)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area53">
        <MAP NAME="area53">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(53)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area54">
        <MAP NAME="area54">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(54)">
        </MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
        <IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
        USEMAP="#area55">
        <MAP NAME="area55">
        <area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(55)">
        </MAP>
</CENTER>
</td>
<tr>
<td width="25" height="25">
<CENTER>
```

```
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area56">
<MAP NAME="area56">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(56)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area57">
<MAP NAME="area57">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(57)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area58">
<MAP NAME="area58">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(58)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area59">
<MAP NAME="area59">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(59)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area60">
<MAP NAME="area60">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(60)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area61">
<MAP NAME="area61">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(61)">
</MAP>
</CENTER>
```

```
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area62">
<MAP NAME="area62">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(62)">
</MAP>
</CENTER>
</td>
<td width="25" height="25">
<CENTER>
<IMG SRC="images/casilla.gif" vspace="0" hspace="0" BORDER="0" WIDTH="25" HEIGHT="25"
USEMAP="#area63">
<MAP NAME="area63">
<area shape="rect" coords="0,0,25,25" href="Amulet:UART.invokeRPC(63)">
</MAP>
</CENTER>
<td>
</tr>
</table>
<table width="200" height="35" border="0" cellpadding="0" cellspacing="0" align="right">
<tr>
<td width="200" height="35" align="right" valign="middle">
<APPLET CODE="BarGraph.class" WIDTH="200" HEIGHT="20" NAME="Barra">
<PARAM NAME="href" VALUE="Amulet:InternalRAM.word(0).value()">
<PARAM NAME="min" VALUE="0">
<PARAM NAME="max" VALUE="65535">
<PARAM NAME="fillPattern" VALUE="1">
<PARAM NAME="sweepFrom" VALUE="left">
<PARAM NAME="updateRate" VALUE=".20,.01">
</APPLET>
</td>
</tr>
</table>
</td>
<td width="5" height="200"></td>
</tr>
</table>

</body>
</html>
```

Anexo 5: Código *HCS12*

5.1 Bejeweld.c

```
#include "Cpu.h"
#include "Events.h"
#include "AS1.h"
#include "TI1.h"
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"




unsigned short TraduceNibble(unsigned char);
unsigned short TraduceMensaje(unsigned char *);
unsigned short Recibe(void);
void InitJuego(void);
unsigned short GeneraGema(int);
void GeneraTablero(void);
void DibujaTablero(bool);
void Dibuja(unsigned short, bool);
void Juego(unsigned short,unsigned short);
void Ayuda(void);
void Abandonar(void);
void EnviaDatos(unsigned short);
void Actualiza(int *);
bool VerificaCoord(unsigned short, unsigned short, unsigned char*);
bool SonVecinos(unsigned short, unsigned short);
void Caida(void);
void ReaccionEnCadena(void);
bool Viable(unsigned char *);
void Traduce(unsigned char * , unsigned short);
short BorraGemas(void);
unsigned short Especial(unsigned short);
unsigned short Comodin(unsigned short, unsigned short);
void DibujaGema_1(unsigned short, bool);
void DibujaGema_2(unsigned short, bool);
void DibujaGema_3(unsigned short, bool);
void DibujaGema_4(unsigned short, bool);
void DibujaGema_5(unsigned short, bool);
```

```
void DibujaGema_6(unsigned short, bool);
void DibujaGema_7(unsigned short, bool);
void DibujaGema_8(unsigned short, bool);
void DibujaEspecial(unsigned short, bool);
void DibujaSeleccion(unsigned short, bool);
void DibujaCorrecto(unsigned short, bool);
void DibujaNivel(bool);

const unsigned CoordTableroX = 108;       //X inicial del tablero
const unsigned CoordTableroY = 3;         //Y inicial del tablero
const unsigned AnchoCasilla = 25;         //Ancho de cada casilla de juego.
const unsigned LargoCasilla = 25;         //Largo de cada casilla de juego.

unsigned int  Progreso=0 ;
unsigned long Puntuacion=0;
unsigned short Help=100, Nivel=0, tick=0;
unsigned char Tablero[64];
bool Envio=0, Abandona=0, Estado=0;

void main(void)
{

 unsigned short RPC=100, coord1=324;

 /*** un Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
 PE_low_level_init();
 /*** End of Processor Expert internal initialization.            ***/

 while(1){

    while(RPC!=64){                    //Recibimos la señal de comenzar partida.
      RPC=Recibe();
    }


    Abandona=0;
    InitJuego();


    while(Abandona==0){

    RPC=Recibe();

     if(RPC<64){
       switch(coord1){
         case 324: coord1=RPC;                    //No tenemos ninguna coordenada y recibimos la primera.
                DibujaSeleccion(coord1, 1);
                break;

         default:  Juego(coord1, RPC);            //Tenemos una coordenada y recibimos la segunda.
                          coord1=324;
```

```
                        break;
      }
     }else{
      switch(RPC){                    //Se han pulsado uno de los 4 botones complementarios al juego.
        case 65: Ayuda();              //Se ha pulsado el botón de ayuda.
              break;
        case 66: Abandonar();            //Se ha pulsado el botón "Abandonar".
                        break;
      }
     }
    }
  }

  /*** Processor Expert end of main routine. DON'T MODIFY THIS CODE!!! ***/
  for(;;){}
  /*** Processor Expert end of main routine. DON'T WRITE CODE BELOW!!! ***/
}/*** End of main routine. DO NOT MODIFY THIS TEXT!!! ***/


/*
** ===================================================================
**     Función     : Recibe
**     Descripción :
**         Esta función recibe un char enviado por la
**         pantalla al microcontrolador y de la forma "0xZZ".
**         La 'traducción' sigue la forma siguiente:
**         "0x30"->"0"
**         ...
**         "0x39"->"9"
**         "0x41"->"A"
**         ...
**         "0x46"->"F"
**     Parametros  :
**          char c        - char recibido
**     Devuelve    :
**          ---           - char traducido
** ===================================================================
*/


unsigned short Recibe(){
unsigned char mensaje[3];
unsigned short RPC=0;
word i;
bool fuera=0;

  while(!fuera){

                        while(AS1_InpLen<3){                    //Esperamos el paquete de datos entero: 3 bytes.
  }                                                             //ya que desde aquí solo recibiremos mensajes de InvokeRPC
              while(AS1_RecvBlock(mensaje, 3, &i)!=ERR_OK){
```

```
   }
   if(i==3||(mensaje[0]==0xd8)){                    //Primer byte = cabecera MSOM
    mensaje[0]=0xE8;
    while(AS1_SendBlock(mensaje, 3, &i)!=ERR_OK){   //Enviamos al buffer el mensaje
    }
    while(AS1_OutLen!=0){
    }
    RPC= TraduceMensaje(&mensaje[1]);
    fuera=1;
   }else{
    while(AS1_SendChar(0xF1)!=ERR_OK){
    }
                while(AS1_OutLen!=0){
    }
              }
  }
return(RPC);
}
```

```
** ===================================================================
**     Función     : TraduceNibble
**     Descripción :
**         Esta función recibe uno de los chars recibidos de la pantalla y lo devuelve
**         en formato ASCII en una variable numerica.
**     Parametros  :
**          char c        - char recibido
**     Devuelve    :
**          ---           - short con la traducción
** ===================================================================
unsigned short TraduceNibble(unsigned char c){

 if((c>='0')&&(c<='9'))
 {
 return(c-'0');
 }
 else if((c>='A')&&(c<='F'))
 {
 return(10+c-'A');
 }
 return(0);
}

** ===================================================================
**     Función     : TraduceMensaje
**     Descripción :
**         Esta función recibe dos chars enviados por la
**         pantalla al microcontrolador y llama para cada uno de
```

```
**        ellos a la función TraduceNibble.
**        Devuelve un entero cuyos 4 primeros bits son la respuesta
**        de la primera llamada y cuyos últimos 4 son la respuesta
**        a la segunda. El valor es devuelto en variable numérica.
**    Parametros :      Ninguno
**         char c        - char recibido
**    Devuelve    :
**         ---             - short con la traducción
** ================================================================
unsigned short TraduceMensaje(unsigned char *mess){

return(TraduceNibble(*mess)<<4|TraduceNibble(*(mess+1)));
}

** ================================================================
**    Función     : Espera
**    Descripción :
**        Esta función esperará que la variable gomal tick varíe su valor
**        en x unidades.
**    Parametros  :
**         int x            - int x
**    Devuelve    :      Nada
** ================================================================


void Espera(int x){
unsigned short k;

            k=tick;
            while((tick-k)<x){     //Esperamos x decimas de segundo.
            }
}

** ================================================================
**    Función     : Juego
**    Descripción :
**        Esta función recibe dos coordenadas de dos gemas del
**        tablero. Si estas son vecinas las intercambia y
**        llama a la función VerificaTablero para comprobar
**        si el movimiento es correcto.
**        Si no son vecinas deselecciona las gemas y prosigue el
**        juego.
**    Parametros  :
**         int coord1    - 1ª coordenada
**         int coord2    - 2ª coordenada
**    Devuelve    :      Nada
** ================================================================
void Juego(unsigned short coord1, unsigned short coord2){
unsigned char i;
int puntos=0;
bool temp1=0, temp2=0;
```

```
        if(SonVecinos(coord1, coord2)){
          DibujaSeleccion(coord2, 1);

                Espera(10);                                     //Esperamos 100 ms

                DibujaSeleccion(coord1, 0);
                DibujaSeleccion(coord2, 0);

                if((Tablero[coord1]==8)||(Tablero[coord2]==8)){
                  puntos=Comodin(coord1, coord2);
        }else{
                  i=Tablero[coord1];
          Tablero[coord1]=Tablero[coord2];
          Tablero[coord2]=i;
          DibujaCorrecto(coord1, 0);                            //Borramos gemas y las redibujamos intercambiadas.
                        Dibuja(coord1, 0);
                        DibujaCorrecto(coord2, 0);
                        Dibuja(coord2, 0);
                        Espera(20);
                        temp1=VerificaCoord(coord1, 0, Tablero);
                        temp2=VerificaCoord(coord2, 0, Tablero);
                        if(temp1||temp2){
                          puntos=BorraGemas();
                        }
        }

                if(puntos!=0){                          //Actualizamos marcadores.
                  Actualiza(&puntos);                   //Hacemos caer las piezas sobre los huecos de las
                  Caida();                              //desaparecidas
                  ReaccionEnCadena();                   //Analizamos posibles casos de reacciones en cadena.
                }else{
                  i=Tablero[coord1];                    //Si el movimiento no es correcto
                  Tablero[coord1]=Tablero[coord2];      //borramos gemas y las redibujamos intercambiadas.
          Tablero[coord2]=i;
          DibujaCorrecto(coord1, 0);
          Dibuja(coord1, 0);
          DibujaCorrecto(coord2, 0);
          Dibuja(coord2, 0);

                }
        }else{
                DibujaSeleccion(coord1, 0);             //Si no son vecinas las deseleccionamos.
        }
}

** ============================================================
**    Función    : ReaccionEnCadena
**
**    Descripción :
**       Esta función comprueba que el tablero resultante de un movimiento correcto
```

```
**        no contenga tras la cíada de piezas y generación de nuevas piezas, otras cadenas
**        formdas. Para ello verifica una a una las piezas, y, si se da el caso llama
**        las funciones Borragema, Actualiza y Caida hasta que el tablero sea estable.
**    Parametros :     Ninguno
**    Devuelve   :     Nada
** ===================================================================
void ReaccionEnCadena(){
int puntos;
unsigned short j=2;
unsigned char i;
byte b=0, temp=0;


  while(b==0){
   i=0;
            puntos=0;
            temp=0;
            while(i!=62){
              if(Tablero[i]!=8){
                temp+=VerificaCoord(i, 0, Tablero);
              }
                        i++;
            }
            if(temp==0){
              b=1;
            }else{
              puntos=BorraGemas()*j;
                        Actualiza(&puntos);
                        Caida();
                        j++;
            }
  }
}

** ===================================================================
**    Función    : BorraGemas
**
**    Descripción :
**       Esta función será llamada durante un movimiento normal,
**       es decir sin comodines.
**       Primero dibujaremos como correctas todas las gemas
**       implicadas en el movimiento y después se borrarán.
**       Después se dibujarán las especiales, se marcarán como correctas
**       y al final se borrarán
**       Durante todo el proceso se irán sumando los puntos
**       correspondientes.
**    Parametros :     Ninguno
**    Devuelve   :
**        ---          - puntos conseguidos
** ===================================================================
short BorraGemas(){
```

```
unsigned short i=0;
short  puntos=0;
byte b=0;



        while(i!=64){
                if(Tablero[i]>20){                      //Dibujamos las gemas normales y especiales como correctas.
                        DibujaCorrecto(i,1);
                        Dibuja(i,0);
                        puntos+=10;
                }
                i++;
        }

 Espera(10);                                            //Esperamos 10ms.
 i=0;

 while(i!=64){
  if((Tablero[i]<30)&&(Tablero[i]>20)){
                   Tablero[i]=0;
                        DibujaCorrecto(i,0);            //Borramos las correctas normales de pantalla. Las especiales no.
                        Espera(10);
                }
  i++;
 }

 i=0;

 while(i!=64){
  if(Tablero[i]>40){
    if(Tablero[i]==48){
                        DibujaCorrecto(i,0);
     Tablero[i]=8;
                        Dibuja(i,0);
    }else{
     DibujaCorrecto(i,0);
     Tablero[i]+=-30;
     Dibuja(i,0);
    }
                        Dibuja(i, 0);                   //Redibujamos las especiales para el caso de haber aparecido alguna
nueva.
                        Espera(10);
  }
  i++;
 }

        i=0;

        while(i!=64){
                if(Tablero[i]>30){
```

```
                        b=1;
                                puntos+=Especial(i);            //Procesamos las especiales implicadas.
                        }
                        i++;
 }
            Espera(10);
            i=0;
 if(b==1){
            while(i!=64){
                        if(Tablero[i]==0){              //En caso de haber explotado gemas especiales las borramos de pantalla.
                                DibujaCorrecto(i, 0);
                                Tablero[i]=0;
                        }
                        i++;
            }
 }
            Espera(10);
            return(puntos);
}

** ================================================================
**    Función    : InitJuego
**
**    Descripción :
**       Inicializa las variables de juego: puntuación y progreso
**       del valor del parámetro recibido y las envía a la
**       pantalla. Después genera y dibuja el tablero de juego
**       inicial.
**    Parametros :     Ninguno
**    Devuelve   :     Nada
** ================================================================
void InitJuego(){
bool ok=0;

            Nivel=1;
            Puntuacion=0;                   //Inicializamos nivel progreso y puntuación.
 Progreso=0;
 EnviaDatos(0);
 EnviaDatos(1);
 EnviaDatos(2);
 while(!ok){
   GeneraTablero();                         //Generamos un tablero viable
   ok=Viable(Tablero);
 }
 Tablero[15]=14;
 DibujaTablero(1);
}

** ================================================================
**    Función    : GeneraTablero
**
```

```
**    Descripción :
**       Genera aleatoriamente un tablero de juego asegurándose
**       de que no existan más de dos gemas contigüas del mismo
**       tipo.
**    Parametros :     Ninguno
**    Devuelve   :     Nada
** ================================================================
*/

void GeneraTablero(){
 int i=0;
 bool sigue=1;
                                        //Generamos las 64 gemas del tablero de tal manera
 while(i<64){                           //que no hayan mas de 3 del mismo tipo que sean contigüas.
            sigue=1;
                    Tablero[i]=GeneraGema(i);
            if((i>15)&&(Tablero[i]==Tablero[i-8])){                 //Comprobamos verticales.
                     if(Tablero[i]==Tablero[i-16]){
                              sigue=0;
                     }
            }
            if((i%8>1)&&(Tablero[i]==Tablero[i-1])){                //Comprobamos horizontales.
                     if(Tablero[i]==Tablero[i-2]){
                       sigue=0;
                     }
                    }
            if(sigue){
              i++;
            }
 }
}

** ================================================================
**    Función    : DibujaTablero
**
**    Descripción :
**       Dibuja cada una de las gemas del tablero o lo borra por
**       completo en función de la variable modo.
**    Parametros :
**        int modo     - Indica si borrar o dibujar.
**                    (0=borra, 1=dibuja)
**    Devuelve   :     Nada
** ================================================================

void DibujaTablero(bool modo){
 unsigned int i=0;
 unsigned char  temp[20], p;
 bool fuera=0;

 if(modo==1){
   while(i<64){
```

```
                    Dibuja(i,0);
                    i++;
   }
  }else{
   temp[0]=0xDB;                                          //Rectángulo relleno.
   Traduce(&temp[1], CoordTableroX);
   Traduce(&temp[5], CoordTableroY);
   Traduce(&temp[9], AnchoCasilla*8);
   Traduce(&temp[13],LargoCasilla*8);
   temp[17]=0x46;
   temp[18]=0x31;

   temp[19]=0x13;//XOFF

   while(AS1_SendBlock(temp, 20, &i)!=ERR_OK){
   }
   while(fuera==0){
      while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
      }
      if(p==0x11){
       fuera=1;
      }
   }
  }
  i=0;
}

** ===================================================================
**     Función     : EnviaDatos
**
**     Descripción :
**         Esta función recibe un entero que indica el dato a enviar.
**         Los datos a enviar serán la puntuación, el progreso y el
**         nivel de la partida.
**         Para cada caso se envía el mensaje correspondiente.
**     Parametros  :
**          int modo      - Indica el dato a enviar
**                        (0=puntos, 1=progreso, 2=nivel)
**     Devuelve    :      Nada
** ===================================================================

void EnviaDatos(unsigned short modo){
  unsigned char mensaje[15];
  unsigned char temp[9];
  unsigned int i=0;

  switch(modo){
    case 0: mensaje[0]=0xD7;                    //La puntuación estará en el String #0 de la RAM del Amulet.
          mensaje[1]=0x30;
          mensaje[2]=0x30;
          i=sprintf(temp, "%lu", Puntuacion);       //Pasamos el entero a string con null en última posición.
```

```
        i=0;
        while(temp[i]!=0){
          mensaje[3+i]=temp[i];
          i++;
        }
        mensaje[3+i]=0x00;

        while(AS1_SendBlock(mensaje, i+5, &i)!=ERR_OK){
        }
        break;

  case 1: mensaje[0]=0xD6;
        mensaje[1]=0x30;                    //El progreso estará en el word #0 de la RAM del Amulet.
        mensaje[2]=0x30;
        Traduce(&mensaje[3], Progreso);

        while(AS1_SendBlock(mensaje, 8, &i)!=ERR_OK){
        }
        break;
  case 2: mensaje[0]=0xD6;
        mensaje[1]=0x30;                    //El Nivel estará en el word #1 de la RAM del Amulet.
        mensaje[2]=0x31;
        Traduce(&mensaje[3], Nivel);        //Pasamos el entero a string con null en última posición.

        while(AS1_SendBlock(mensaje, 8, &i)!=ERR_OK){
        }
        break;
 }
 while(AS1_OutLen!=0){
 }
}

** ===================================================================
**     Función     :  Actualiza
**
**     Descripción :
**         Esta función es la responsable de cambiar de nivel y
**         actualizar  por pantalla y calcular el nivel, puntuación y progreso.
**     Parametros  :
**         int * puntos  - puntos a sumar
**     Devuelve    :      Nada
** ===================================================================

void Actualiza(int *puntos){
long int temp, temp2;

 if(((*puntos)<0)&&(Puntuacion<(abs(*puntos)))){     //Cuando los puntos a restar son mas de los que
    Puntuacion =0;                                   //tenemos acumulados.
 }else{
  Puntuacion+=(*puntos);
 }
```

```
 if(Puntuacion!=0){
  temp=(*puntos)*65535/(Nivel*1000);

  if(temp>0){
   if(temp<65535-Progreso){                          //Que el resultado sea inferior a 65535 => sumamos.
    Progreso+=temp;
   }else{
    //Que se supere Nivel
    Nivel++;
    EnviaDatos(2);

    DibujaNivel(1);
    Espera(5);
    DibujaNivel(0);
    Espera(5);
    DibujaNivel(1);
    Espera(5);
    DibujaNivel(0);
    Espera(5);
    DibujaNivel(1);
    Espera(5);
    DibujaNivel(0);
    Espera(5);


    Progreso+=temp;
    temp2=Progreso+temp-65535;
    Progreso=temp2;
   }
  }else{
   if(abs(temp)>Progreso){
    Progreso=0;
   }else{
    Progreso+=temp;
   }
  }
 }


 EnviaDatos(0);
 EnviaDatos(1);
}

** ================================================================
**    Función    : GeneraGema
**
**    Descripción :
**       Esta función genera un número aleatorio entre 1 y 7 y lo
**       devuelve.
**    Parametros :    Ninguno
```

```
**    Devuelve    :
**          ---          - entero entre 1 y 7
** ===================================================================
*/

unsigned short GeneraGema(int temp){
 int temp2;

 temp2=TIM0_TCNT+temp;
 srand(temp2);
 return((rand()%7)+1);
}

** ===================================================================
**    Función     : Dibuja
**
**    Descripción :
**       Esta función recibe dos parámetros. El primero dirá que se
**       debe dibujar. El segundo especificará en que posición se hará.
**       En el caso de tratarse de una gema  especial primero dibujaremos la pieza para luego
**       hacerlo con la carácterística de las gemas especiales.
**    Parametros  :
**        bool modo     - Indica el modo de dibujo
**                    (0=normal, 1=en caída)
**        int indice    - Indica la posición de la gema a dibujar
**    Devuelve    :     Nada
** ===================================================================
*/

void Dibuja(unsigned short indice, bool modo){
 unsigned int TipoGema;

            TipoGema=((Tablero[indice])%10);                    //Gema original

            switch(TipoGema){

              case 1 :   DibujaGema_1(indice, modo);
                                                          break;
                         case 2 :    DibujaGema_2(indice, modo);
                                                          break;
                         case 3 :    DibujaGema_3(indice, modo);
                                                          break;
                         case 4 :    DibujaGema_4(indice, modo);
                                                          break;
                         case 5 :    DibujaGema_5(indice, modo);
                                                          break;
                         case 6 :    DibujaGema_6(indice, modo);
                                                          break;
                         case 7 :    DibujaGema_7(indice, modo);
                                                          break;
              case 8 :  DibujaGema_8(indice, modo);
```

```
                    break;
              default: break;
             }
 if(Tablero[indice]>10){
                    DibujaEspecial(indice, modo);
 }
}

** ================================================================
**    Función     : Ayuda
**
**    Descripción :  Esta función muestra por pantalla una, o la
**                   única, gema cuyo movimiento es correcto.
**                   Trás resaltar dicha gema por pantalla dos veces
**                   se descontarán los puntos
**                   invocando la función "Actualiza".
**
**    Parametros :     Ninguno
**    Devuelve   :     Nada
** ================================================================

void Ayuda(){
 int i;

 DibujaCorrecto(Help,1);
 Dibuja(Help,0);
 Espera(10);
 DibujaCorrecto(Help,0);
 Dibuja(Help,0);
 Espera(10);
 DibujaCorrecto(Help,1);
 Dibuja(Help,0);
 Espera(10);
 DibujaCorrecto(Help,0);
 Dibuja(Help,0);
 Espera(10);

 i=-30;
 Actualiza(&i);
}

** ================================================================
**    Función     : Abandonar
**
**    Descripción :  Cuando el usuario pulse el botón "Abandonar"
**                   aparecerán en pantalla dos botones de
**                   confirmación: "Si" y "No". La zona del tablero
**                   de juego se borrará.
**                   En caso de cancelar se redibuja
**
**    Parametros  :
```

```
**        unsigned short i - Valor del 'RPC' recibido de la pantalla.
**    Devuelve   :     Nada
** ===================================================================

void Abandonar(){
 int j=0;
 unsigned short RPC=123;
 bool fuera=0;

 DibujaTablero(0);

 while(fuera==0){
  RPC=Recibe();

  switch(RPC){

    case 67:  Abandona=1;
              fuera=1;
              break;

    case 68:  DibujaTablero(1);
              fuera=1;
              break;

    default:  break;
  }
 }
}

** ===================================================================
**    Función    :  SonVecinos
**
**    Descripción :  Esta función comprueba si las dos gemas
**                   seleccionadas son vecinas o no.
**
**    Parametros  :
**                        - coordenadas de las gemas
**    Devuelve   :
**                        - boleano: (1= vecinas  0=no vecinas)
** ===================================================================

bool SonVecinos(unsigned short coord1, unsigned short coord2){

if( ((abs(coord1-coord2)==1)&&((coord1/8)==(coord2/8))) || (abs(coord1-coord2)==8)){
  return(1);
}
return(0);
}

** ===================================================================
**    Función    :  VerificaCoord
```

```
**
**    Descripción :  comprueba si una gema se
**                alinea con como mínimo tres del mismo
**                tipo en horizontal y/o vertical.
**                Dependiendo del modo, recibido también como
**                parámetro, marcaremos las que posteriormente
**                desaparecerán e introduciremos en el tablero
**                las especiales generadas por el movimiento.
**
**    Parametros  :
**                        - Coordenada a comprobar
**                        - Modo: 0=se marcarán   1= solo se comprueba
**    Devuelve    :       Nada
** ====================================================================
*/

bool VerificaCoord(unsigned short coord, unsigned short modo, unsigned char * tablero){
        unsigned short temphor=1,tempver=1, tipo, x, y;
        short i=0;
        bool juego=0,explosion=0;

        if(tablero[coord]>10){
                explosion=1;
        }

        tipo=tablero[coord]%10;
        x=coord;
        y=coord;


        //*****************************Horizontal izquierda.

        if(tipo!=0){

        i=coord;
        while((i%8)>0){
                if(tipo==(tablero[i-1]%10)){
                        if(tablero[i-1]>10){
                                explosion=1;
                        }
                        x=i-1;                  //coordenada de la gema mas a la izquierda igual a tipo de coord1.
                        temphor++;
                        i--;
                }
                else{
                        break;
                }
        }

        //*****************************Horizontal derecha.
```

```
i=coord;
while((i%8)<7){
        if(tipo==(tablero[i+1]%10)){
                if(tablero[i+1]>10){
                        explosion=1;
                 }
                temphor++;
                i++;
        }
        else{
                break;
        }
}

//*******************************Vertical arriba

i=coord;
while((i-8)>=0){
        if(tipo==(tablero[i-8]%10)){
                if(tablero[i-8]>10){
                        explosion=1;
                 }
                 y=i-8;                 //coordenada de la gema mas superior igual a la base.

                tempver++;
                i+=-8;
        }
        else{
                break;
        }
}

//*******************************Vertical abajo

i=coord;
while((i+8)<63){
        if(tipo==(tablero[i+8]%10)){
                if(tablero[i+8]>10){
                        explosion=1;
                 }
                tempver++;
                i+=8;
        }
        else{
                break;
        }
}

 //*******************************      Marcamos las gemas a desaparecer y ponemos a uno la marca de
                                        //movimiento correcto si es necesario.
i=temphor;
```

```
        if(i>2){
    if(modo==0){
                                while(i>0){
                                 if(tablero[x+i-1]<20){
                                        tablero[x+i-1]+=20;
                                 }
                        i--;
              }
    }
        }
        i=tempver;
        if(i>2){
    if(modo==0){
                                while(i>0){
                                 if(tablero[y+(i-1)*8]<20){
                                        tablero[y+(i-1)*8]+=20;
                                 }
                                 i--;
                                }
    }
        }

        if((temphor>2)||(tempver>2)){
         juego=1;
        }

        //*******************************

        if((modo==0)&&(juego)&&(!explosion)){     //Siempre que no explote una gema especial comprobaremos si debemos
          if((temphor>=5)||(tempver>=5)){         //hacer que aparezcan nuevas gemas especiales o comodines.
            tablero[coord]=48;
          }else{
            if(temphor==4){
              tablero[x+2]=tipo+40;
            }else{
              if(tempver==4){
                tablero[coord]=tipo+40;
              }else{
                if((temphor==3)&&(tempver==3)){
                  tablero[coord]=tipo+40;
                }
              }
            }
          }
        }
        }

        return(juego);
}
```

```
** ===================================================================
**     Función     : Traduce
**
**     Descripción :  Esta función realiza la misma operación que la
**                 llamada a "sprintf" consumiendo menos recursos.
**                 Recibimos un entero y lo pasamos a 'char'.
**
**     Parametros  :
**         unsigned short valor  - valor a "traducir"
**         unsigned char * apunt - apuntador a la posición desde la
**                                 que comenzaremos a escribir.
**     Devuelve    :      Nada
** ===================================================================

void Traduce(unsigned char * apunt, unsigned short valor){
  unsigned short temp;
  short i=0;


  while(i<4){
    temp=(valor>>(4*i))&0x000F;
    *(apunt+(3-i))=temp+48;
              if(temp>9){
                        *(apunt+(3-i))=temp+55;
                }
    i++;

  }
}

** ===================================================================
**     Función     : Caida
**
**     Descripción : Genera un nuevo tablero y se asegura de que sea viable. Después realiza la
**                   caída de las gemas antiguas y nuevas en el tablero de juego.
**
**     Parametros  :      Ninguno
**     Devuelve    :      Nada
** ===================================================================

void Caida(){
              int j=8, k=8, i=1, x=0;
              unsigned char tablerotemp[64], temporales[8], temp[8];
              bool fuera=0, ok=0, crisis=0;

  while(i!=9){                                        //copiamos en el tablero temporal la configuracion del original
                        while(j>0){                   //con las gemas ya caídas y marcamos cuantas faltan en cada columna
                                  tablerotemp[k*8-i]=Tablero[j*8-i];                    //en "temporales".
                                  if(Tablero[8*j-i]!=0){
                                            k--;
                                  }
```

```
                        j--;
                        if(Tablero[j*8-i]==8){          //Si existe un comodín no hará falta compobar la continuidad
                                ok=1;                   //de juego.
                                Help=j*8-i;
                        }
                }
                temporales[8-i]=k;
                while(k>0){
                        tablerotemp[k*8-i]=0;
                        k--;
                }
                i++;
                j=8;
                k=8;
}


        while(fuera==0){                                //Hasta que el tablero sea viable.
                i=1;
                while(i!=9){
                k=temporales[8-i];                      //Generamos las gemas que faltan en tablerotemp
                        while(k!=0){
                          if(x==5000){
                            tablerotemp[k*8-i]=8;
                            k--;
                          }else{
                            tablerotemp[k*8-i]=GeneraGema(i+k);
                                k--;
                          }
                        }
                        i++;
                }
                if((!ok)&&(x<5000)){            //si después de 50 generaciones aleatorias seguimos sin viabilidad
                  fuera=Viable(tablerotemp);    //se introducirá en el juego un comodín.
                }else{
                  fuera=1;
                };
                x++;
        }

        while(ok!=8){

 ok=0;
 i=0;
 while(i!=8){
        temp[i]=0;
        i++;
 }
 i--;
 while(i>=0){
        j=0;
```

```
		while(j!=8){
			if(Tablero[i*8+j]==0){
				temp[j]=1;
			}
			if(temp[j]==1){
				if(i!=0){
					DibujaCorrecto((i-1)*8+j,0);	//Borro casilla
			Dibuja((i-1)*8+j,1);			//Dibujo caida
				}
			}
		j++;
		}
i--;
}
i++;
while(i!=8){
		temp[i]=0;
		i++;
}
i--;
while(i>=0){
		j=0;
		while(j!=8){
			if(Tablero[i*8+j]==0){
				temp[j]=1;
			}
			if(temp[j]==1){
				if(i!=0){
					DibujaCorrecto((i-1)*8+j,0);		//Borro casilla superior
					Tablero[i*8+j]=Tablero[(i-1)*8+j];	//Cambiamos gema de posicion en Tablero

					DibujaCorrecto(i*8+j, 0);		//Borro casilla
			Dibuja(i*8+j,0);					//Dibujo normal
				}else{
					Tablero[i*8+j]=tablerotemp[((temporales[j])-1)*8+j];
					DibujaCorrecto(i*8+j, 0);
					Dibuja(i*8+j,0);
					temporales[j]+=-1;
				}
			}
		j++;
		}
i--;
}
i++;
while(i!=8){
		if(temporales[i]==0){
			ok++;
		}
		i++;
}
```

```
            }
}

** ================================================================
**    Función     : Viable
**
**    Descripción :  Revisa las gemas del tablero y comprueba que este sea viable
**                   Al encontrar una gema que produzca un movimiento
**                   correcto guarda en la variable Help su coordenada para cuando
**                   el usuario pulse el botón Ayuda.
**
**    Parametros  :
**                        - Tablero a comprobar
**    Devuelve    :
**                        - Boleano: 0= no viable   1=viable
** ================================================================

bool Viable(unsigned char *tablero){
 bool fuera=0, ok=0;
 short i=0,j=0;

 while((i!=62)&&(!fuera)){
                        if(i<56){
                         if((tablero[i]==8)||(tablero[i+8]==8)){
                          if(tablero[i]==8){
                            Help=i;
                          }else{
                            Help=i+8;
                          }
                          fuera=1;
                         }
                         j=tablero[i];
                         tablero[i]=tablero[i+8];                    //Comprobamos si hay continuidad en el nuevo tablero
                         tablero[i+8]=j;
                                  ok=VerificaCoord(i,1,tablero);  //y guardamos la posicion de la gema
                                  if(ok){                                 // que lo genera en la variable Help.
                                    Help=i+8;
                                  }
                                  ok=VerificaCoord(i+8,1,tablero);
                                  if(ok){
                                    Help=i;
                                  }
                                  j=tablero[i];
                         tablero[i]=tablero[i+8];
                         tablero[i+8]=j;
                         if(ok){
                          fuera=1;
                         }
                        }
                        if((i%8)!=7){
                         if((tablero[i]==8)||(tablero[i+1]==8)){
```

```
                        if(tablero[i]==8){
                          Help=i;
                        }else{
                          Help=i+1;
                        }
                        fuera=1;
                      }
                      j=tablero[i];
                      tablero[i]=tablero[i+1];
                      tablero[i+1]=j;
                              ok=VerificaCoord(i,1,tablero);
                              if(ok){
                                Help=i+1;
                              }
                              ok=VerificaCoord(i+1,1,tablero);
                              if(ok){
                                Help=i;
                              }
                              j=tablero[i];
                      tablero[i]=tablero[i+1];
                      tablero[i+1]=j;
                      if(ok){
                        fuera=1;
                      }
                    }
                    i++;
 }
 return(fuera);
}

** ================================================================
**    Función     : Especial
**
**    Descripción :  Revisa las 8 casillas de alrededor de la gema
**                   especial en busca de otra. En este caso se desata
**                   lo que llamaremos una reacción en cadena  de gemas especiales y la
**                   fiunción se llamará a sí misma de forma recursiva.
**
**    Parametros  :
**                        - Coordeada de la gema a comprobar
**    Devuelve    :
**                        - Puntuación generada.
** ================================================================

unsigned short Especial(unsigned short i){
unsigned short temp=80;


  DibujaCorrecto(i,1);              //Resaltamos la pieza especial con un rápido parpadeo.
  Dibuja(i,0);
  Espera(10);
```

```
DibujaCorrecto(i,0);
Dibuja(i,0);
Espera(10);
DibujaCorrecto(i,1);
Dibuja(i,0);
Espera(10);
DibujaCorrecto(i,0);
Dibuja(i,0);
Espera(10);


Tablero[i]=0;                    //Ponemos a cero su posición.

if(i%8!=0){
  DibujaCorrecto(i-1, 1);
                          if(Tablero[i-1]<30){
                            if(Tablero[i-1]>10){
                              temp+=Especial(i-1);
                            }else{
                              Tablero[i-1]=0;
                            }
                          }
  if(i/8!=0){
    DibujaCorrecto(i-9, 1);
                          if(Tablero[i-9]<30){
                             if(Tablero[i-9]>10){
                               temp+=Especial(i-9);
                             }else{
                               Tablero[i-9]=0;
                             }
                          }
  }
  if(i/8!=7){
    DibujaCorrecto(i+7, 1);
                          if(Tablero[i+7]<30){
                             if(Tablero[i+7]>10){
                               temp+=Especial(i+7);
                             }else{
                               Tablero[i+7]=0;
                             }
                          }
  }
}
if(i/8!=0){
  DibujaCorrecto(i-8, 1);
                          if(Tablero[i-8]>10){
                             temp+=Especial(i-8);
                            }else{
                             Tablero[i-8]=0;
                            }
}
```

```
 if(i%8!=7){
  DibujaCorrecto(i+1, 1);
                        if(Tablero[i+1]>10){
                          temp+=Especial(i+1);
                         }else{
                          Tablero[i+1]=0;
                         }
  if(i/8!=0){
   DibujaCorrecto(i-7, 1);
                        if(Tablero[i-7]>10){
                          temp+=Especial(i-7);
                         }else{
                          Tablero[i-7]=0;
                         }
  }
  if(i/8!=7){
   DibujaCorrecto(i+9, 1);
                        if(Tablero[i+9]>10){
                          temp+=Especial(i+9);
                         }else{
                          Tablero[i+9]=0;
                         }
  }
 }
 if(i/8!=7){
  DibujaCorrecto(i+8, 1);
            if(Tablero[i+8]>10){
                          temp+=Especial(i+8);
                         }else{
                          Tablero[i+8]=0;
                         }
 }
 Espera(10);
 return(temp);
}

** ================================================================
**    Función     : Comodin
**
**    Descripción :    Esta función se encargará de borrar todas las gemas del mismo
**                     tipo que la escogida por el usuario junto al comodín.
**
**    Parametros  :
**                     - coordenadas de las gemas
**    Devuelve    :
**                     - los puntos conseguidos.
** ================================================================

unsigned short Comodin(unsigned short coord1, unsigned short coord2){
 unsigned char tipo;
 short puntos=0, i=0, como;
```

```
 if(Tablero[coord1]==8){
  tipo=Tablero[coord2];
  como=coord1;
 }else{
  tipo=Tablero[coord1];
  como=coord2;
 }

 while(i!=64){
  if(Tablero[i]==tipo){
   DibujaCorrecto(como,1);
   Dibuja(como,0);
   DibujaCorrecto(i, 1);
                        Dibuja(i, 0);
                        Espera(2);
                        DibujaCorrecto(como,0);
   Dibuja(como,0);
   Espera(2);
  }
  i++;
 }

 Espera(10);

 i=0;

 while(i!=64){
  if(Tablero[i]==tipo){
   DibujaCorrecto(como,1);
   Dibuja(como,0);
   DibujaCorrecto(i, 0);
   Tablero[i]=0;
   puntos+=10;
   Espera(2);
                        DibujaCorrecto(como,0);
   Dibuja(como,0);
   Espera(2);
  }
  i++;
 }

 DibujaCorrecto(como, 0);
 Tablero[coord1]=0;
 Tablero[coord2]=0;


return(puntos);
}
```

```
/*=============================================================================================
**
**          Todas estas funciones son las que dibujarán por pantalla.
**
**          * Las funciones "DibujaGema_X" dibujarán la gema de tipo X sobre la casilla especificada por parámetro.
**          El modo indica si se debe dibujar centrada o a media caída para dar el efecto de movimiento.
**          * DibujaEspecial dibujará sobre una gema especial un cuadro para distinguirla de las normales
**          * DibujaSelección dibuja un recuadro sobre la gema seleccionada y lo borra también en función del parámetro modo.
**          * DibujaCorrecto dibuja un rectangulo relleno donde dibujar de nuevo la gema y marcarla como gema correcta.. En
**            función del modo también borra la casilla especificada como parámetro.
**          * DibujaNivel repetirá un rectangulo alrededor del marcador para mostrar que este se ha incrementado.
**===========================================================================================*/

void DibujaGema_1(unsigned short indice, bool modo){          //Modo: 0=normal
            unsigned char  temp[20];                          //      1=en caída.
            unsigned char p;
            unsigned short x=0, y=0;
            unsigned int i=0;
            bool fuera=0;


  x=CoordTableroX+((indice%8)* AnchoCasilla);
  y=CoordTableroY+((indice/8)*AnchoCasilla);
  if(modo){
   y=y+12;
  }
  temp[0]=0xDA;
  Traduce(&temp[1], x+4);                                 //Rectángulo
  Traduce(&temp[5], y+4);
  Traduce(&temp[9], 21);
  Traduce(&temp[13],21);

  temp[17]=0x30;
  temp[18]=0x31;

  temp[19]=0x13;//XOFF
  while(AS1_SendBlock(temp, 20, &i)!=ERR_OK){
  }
  while(fuera==0){
     while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
     }
     if(p==0x11){
      fuera=1;
     }
  }
}

void DibujaGema_2(unsigned short indice, bool modo){          //Modo: 0=normal
            unsigned char  temp[58];                          //      1=en caída.
            unsigned char p;
            unsigned short x=0, y=0;
```

```
            unsigned int i=0;
            bool fuera=0;


  x=CoordTableroX+((indice%8)* AnchoCasilla)+1;
  y=CoordTableroY+((indice/8)*AnchoCasilla)+1;
  if(modo){
   y=y+12;
  }
  temp[0]=0xD9;
  Traduce(&temp[1], x+13);                          //Primera línea
  Traduce(&temp[5], y+4);
  Traduce(&temp[9], x+4);
  Traduce(&temp[13], y+20);
  temp[17]=0x30;
  temp[18]=0x31;

  temp[19]=0xD9;
  Traduce(&temp[20], x+13);                          //Segunda línea
  Traduce(&temp[24], y+4);
  Traduce(&temp[28], x+22);
  Traduce(&temp[32], y+20);
  temp[36]=0x30;
  temp[37]=0x31;

  temp[38]=0xD9;
  Traduce(&temp[39], x+4);                          //Tercera línea
  Traduce(&temp[43], y+21);
  Traduce(&temp[47], x+22);
  Traduce(&temp[51], y+21);
  temp[55]=0x30;
  temp[56]=0x31;

  temp[57]=0x13;//XOFF
  while(AS1_SendBlock(temp, 58, &i)!=ERR_OK){
  }
  while(fuera==0){
     while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
     }
     if(p==0x11){
      fuera=1;
     }
  }
}

void DibujaGema_3(unsigned short indice, bool modo){
            unsigned char  temp[58];
            unsigned char p;
            unsigned short x=0, y=0;
            unsigned int i=0;
            bool fuera=0;
```

```
  x=CoordTableroX+((indice%8)* AnchoCasilla)+1;
  y=CoordTableroY+((indice/8)*AnchoCasilla);
  if(modo){
   y=y+12;
  }

  temp[0]=0xD9;                              //Primera línea
  Traduce(&temp[1], x+4);
  Traduce(&temp[5], y+4);
  Traduce(&temp[9], x+21);
  Traduce(&temp[13], y+4);
  temp[17]=0x30;
  temp[18]=0x31;

  temp[19]=0xD9;                              //Segunda línea
  Traduce(&temp[20], x+4);
  Traduce(&temp[24], y+5);
  Traduce(&temp[28], x+13);
  Traduce(&temp[32], y+22);
  temp[36]=0x30;
  temp[37]=0x31;

  temp[38]=0xD9;                              //Tercera línea
  Traduce(&temp[39], x+13);
  Traduce(&temp[43], y+22);
  Traduce(&temp[47], x+22);
  Traduce(&temp[51], y+4);
  temp[55]=0x30;
  temp[56]=0x31;

  temp[57]=0x13;//XOFF
  while(AS1_SendBlock(temp, 58, &i)!=ERR_OK){
  }
  while(fuera==0){
     while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
     }
     if(p==0x11){
      fuera=1;
     }
  }
}


void DibujaGema_4(unsigned short indice, bool modo){
            unsigned char  temp[83];
            unsigned char p;
            unsigned short x=0, y=0;
            unsigned int i=0;
 bool fuera=0;
```

```
x=CoordTableroX+((indice%8)* AnchoCasilla)+1;
y=CoordTableroY+((indice/8)*AnchoCasilla)+1;
if(modo){
 y=y+12;
}

temp[0]=0xD9;
Traduce(&temp[1], x+12);                    //Primera línea
Traduce(&temp[5], y+5);
Traduce(&temp[9], x+12);
Traduce(&temp[13], y+21);
temp[17]=0x30;
temp[18]=0x33;

temp[19]=0xD9;                              //Segunda línea
Traduce(&temp[20], x+5);
Traduce(&temp[24], y+12);
Traduce(&temp[28], x+21);
Traduce(&temp[32], y+12);
temp[36]=0x30;
temp[37]=0x33;

temp[38]=0xDC;                              //Primer pixel
Traduce(&temp[39], x+4);
Traduce(&temp[43], y+13);

temp[47]=0x30;
temp[48]=0x31;

temp[49]=0xDC;                              //Segundo pixel
Traduce(&temp[50], x+13);
Traduce(&temp[54], y+4);
temp[58]=0x30;
temp[59]=0x31;

temp[60]=0xDC;                              //Tercer pixel
Traduce(&temp[61], x+22);
Traduce(&temp[65], y+13);
temp[69]=0x30;
temp[70]=0x31;

temp[71]=0xDC;                              //Cuarto pixel
Traduce(&temp[75], x+13);
Traduce(&temp[79], x+22);
temp[80]=0x30;
temp[81]=0x31;

temp[82]=0x13;//XOFF
while(AS1_SendBlock(temp, 83, &i)!=ERR_OK){
```

```
  }
  while(fuera==0){
     while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
     }
     if(p==0x11){
      fuera=1;
     }
  }
}

void DibujaGema_5(unsigned short indice, bool modo){
              unsigned char  temp[96];
              unsigned char p;
              unsigned short x=0, y=0;
              unsigned int i=0;
              bool fuera=0;


  x=CoordTableroX+((indice%8)* AnchoCasilla)+1;
  y=CoordTableroY+((indice/8)*AnchoCasilla);
  if(modo){
   y=y+12;
  }

  temp[0]=0xD9;                              //Primera línea
  Traduce(&temp[1], x+4);
  Traduce(&temp[5], y+4);
  Traduce(&temp[9], x+22);
  Traduce(&temp[13], y+4);
  temp[17]=0x30;
  temp[18]=0x33;

  temp[19]=0xD9;                              //Segunda línea
  Traduce(&temp[20], x+4);
  Traduce(&temp[24], y+21);
  Traduce(&temp[28], x+22);
  Traduce(&temp[32], y+21);
  temp[36]=0x30;
  temp[37]=0x33;

  temp[38]=0xD9;                              //Tercera línea
  Traduce(&temp[39], x+13);
  Traduce(&temp[43], y+4);
  Traduce(&temp[47], x+13);
  Traduce(&temp[51], y+22);
  temp[55]=0x30;
  temp[56]=0x31;

  temp[57]=0xD9;                              //Cuarta línea
  Traduce(&temp[58], x+4);
  Traduce(&temp[62], y+4);
```

```
  Traduce(&temp[66], x+21);
  Traduce(&temp[70], y+21);
  temp[74]=0x30;
  temp[75]=0x31;

  temp[76]=0xD9;                              //Quinta línea
  Traduce(&temp[77], x+22);
  Traduce(&temp[81], y+4);
  Traduce(&temp[85], x+4);
  Traduce(&temp[89], y+21);
  temp[93]=0x30;
  temp[94]=0x31;

  temp[95]=0x13;//XOFF
  while(AS1_SendBlock(temp, 96, &i)!=ERR_OK){
  }
  while(fuera==0){
     while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
     }
     if(p==0x11){
      fuera=1;
     }
  }

}

void DibujaGema_6(unsigned short indice, bool modo){
              unsigned char  temp[96];
              unsigned char p;
              unsigned short x=0, y=0;
              unsigned int i=0;
              bool fuera=0;


  x=CoordTableroX+((indice%8)* AnchoCasilla);
  y=CoordTableroY+((indice/8)*AnchoCasilla)+1;
  if(modo){
   y=y+12;
  }

  temp[0]=0xD9;                               //Primera línea
  Traduce(&temp[1], x+4);
  Traduce(&temp[5], y+4);
  Traduce(&temp[9], x+4);
  Traduce(&temp[13], y+22);
  temp[17]=0x30;
  temp[18]=0x33;

  temp[19]=0xD9;                               //Segunda línea
  Traduce(&temp[20], x+21);
  Traduce(&temp[24], y+4);
```

```
  Traduce(&temp[28], x+21);
  Traduce(&temp[32], y+22);
  temp[36]=0x30;
  temp[37]=0x33;

  temp[38]=0xD9;                          //Tercera línea
  Traduce(&temp[39], x+4);
  Traduce(&temp[43], y+13);
  Traduce(&temp[47], x+22);
  Traduce(&temp[51], y+13);
  temp[55]=0x30;
  temp[56]=0x31;

  temp[57]=0xD9;                          //Cuarta línea
  Traduce(&temp[58], x+5);
  Traduce(&temp[62], y+4);
  Traduce(&temp[66], x+21);
  Traduce(&temp[70], y+21);
  temp[74]=0x30;
  temp[75]=0x31;

  temp[76]=0xD9;                          //Quinta línea
  Traduce(&temp[77], x+21);
  Traduce(&temp[81], y+4);
  Traduce(&temp[85], x+5);
  Traduce(&temp[89], y+21);
  temp[93]=0x30;
  temp[94]=0x31;

  temp[95]=0x13;//XOFF
  while(AS1_SendBlock(temp, 96, &i)!=ERR_OK){
  }
  while(fuera==0){
    while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
    }
    if(p==0x11){
     fuera=1;
    }
  }
}

void DibujaGema_7(unsigned short indice, bool modo){
            unsigned char  temp[153];
            unsigned char p;
            unsigned short x=0, y=0;
            unsigned int i=0;
            bool fuera=0;


  x=CoordTableroX+((indice%8)* AnchoCasilla)+1;
  y=CoordTableroY+((indice/8)*AnchoCasilla)+1;
```

```
if(modo){
 y=y+12;
}

temp[0]=0xD9;                          //Primera línea
Traduce(&temp[1], x+4);
Traduce(&temp[5], y+10);
Traduce(&temp[9], x+4);
Traduce(&temp[13], y+16);
temp[17]=0x30;
temp[18]=0x32;

temp[19]=0xD9;                          //Segunda línea
Traduce(&temp[20], x+21);
Traduce(&temp[24], y+10);
Traduce(&temp[28], x+21);
Traduce(&temp[32], y+16);
temp[36]=0x30;
temp[37]=0x32;

temp[38]=0xD9;                          //Tercera línea
Traduce(&temp[39], x+10);
Traduce(&temp[43], y+4);
Traduce(&temp[47], x+16);
Traduce(&temp[51], y+4);
temp[55]=0x30;
temp[56]=0x32;

temp[57]=0xD9;                          //Cuarta línea
Traduce(&temp[58], x+10);
Traduce(&temp[62], y+21);
Traduce(&temp[66], x+16);
Traduce(&temp[70], y+21);
temp[74]=0x30;
temp[75]=0x32;

temp[76]=0xD9;                          //Quinta línea
Traduce(&temp[77], x+10);
Traduce(&temp[81], y+4);
Traduce(&temp[85], x+4);
Traduce(&temp[89], y+10);
temp[93]=0x30;
temp[94]=0x32;

temp[95]=0xD9;                          //Sexta línea
Traduce(&temp[96], x+16);
Traduce(&temp[100], y+4);
Traduce(&temp[104], x+21);
Traduce(&temp[108], y+10);
temp[112]=0x30;
temp[113]=0x31;
```

```
  temp[114]=0xD9;                          //Séptima línea
  Traduce(&temp[115], x+5);
  Traduce(&temp[119], y+16);
  Traduce(&temp[123], x+10);
  Traduce(&temp[127], y+21);
  temp[131]=0x30;
  temp[132]=0x32;

  temp[133]=0xD9;                          //Octava línea
  Traduce(&temp[134], x+16);
  Traduce(&temp[138], y+21);
  Traduce(&temp[142], x+10);
  Traduce(&temp[146], y+10);
  temp[150]=0x30;
  temp[151]=0x32;

  temp[152]=0x13;//XOFF
  while(AS1_SendBlock(temp, 153, &i)!=ERR_OK){
  }
  while(fuera==0){
     while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
     }
     if(p==0x11){
      fuera=1;
     }
  }
}



void DibujaGema_8(unsigned short indice, bool modo){
unsigned char  temp[77];
              unsigned char p;
              unsigned short x=0, y=0;
              unsigned int i=0;
              bool fuera=0;


  x=CoordTableroX+((indice%8)* AnchoCasilla);
  y=CoordTableroY+((indice/8)*AnchoCasilla);
  if(modo){
   y=y+12;
  }
  temp[0]=0xDB;                            //Rectángulo
  Traduce(&temp[1], x+10);
  Traduce(&temp[5], y+10);
  Traduce(&temp[9], 15);
  Traduce(&temp[13], 15);
  temp[17]=0x30;
  temp[18]=0x31;
```

```
  temp[19]=0xDB;                        //Rectángulo
  Traduce(&temp[20], x+8);
  Traduce(&temp[24], y+8);
  Traduce(&temp[28], 13);
  Traduce(&temp[32], 13);
  temp[36]=0x46;
  temp[37]=0x31;

  temp[38]=0xDB;                        //Rectángulo
  Traduce(&temp[39], x+6);
  Traduce(&temp[43], y+6);
  Traduce(&temp[47], 11);
  Traduce(&temp[51], 11);
  temp[55]=0x30;
  temp[56]=0x31;

  temp[57]=0xDB;                        //Rectángulo
  Traduce(&temp[58], x+4);
  Traduce(&temp[62], y+4);
  Traduce(&temp[66], 9);
  Traduce(&temp[70], 9);
  temp[74]=0x30;
  temp[75]=0x31;

  temp[76]=0x13;//XOFF
  while(AS1_SendBlock(temp, 77, &i)!=ERR_OK){
  }
  while(fuera==0){
     while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
     }
     if(p==0x11){
      fuera=1;
     }
  }
}

void DibujaEspecial(unsigned short indice, bool modo){
 unsigned char  temp[20];
              unsigned char p;
              unsigned short x=0, y=0;
              unsigned int i=0;
              bool fuera=0;
              ;

  x=CoordTableroX+((indice%8)* AnchoCasilla)+2;
  y=CoordTableroY+((indice/8)*AnchoCasilla)+2;
  if(modo){
   y=y+12;
  }
  temp[0]=0xDA;                         //Rectángulo
```

```
  Traduce(&temp[1], x);
  Traduce(&temp[5], y);
  Traduce(&temp[9], 24);
  Traduce(&temp[13], 24);
  temp[17]=0x30;
  temp[18]=0x31;
  temp[19]=0x13;//XOFF

  while(AS1_SendBlock(temp, 20, &i)!=ERR_OK){
  }
  while(fuera==0){
     while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
     }
     if(p==0x11){
      fuera=1;
     }
  }
}

void DibujaSeleccion(unsigned short indice, bool modo){
              unsigned char  temp[20];
              unsigned char p;
              unsigned short x=0, y=0;
              unsigned int i=0;
              bool fuera=0;


  x=CoordTableroX+((indice%8)* AnchoCasilla)+2;
  y=CoordTableroY+((indice/8)*AnchoCasilla)+2;
  temp[0]=0xDA;                                  //Rectángulo
  Traduce(&temp[1], x-1);
  Traduce(&temp[5], y-1);
  Traduce(&temp[9], 26);
  Traduce(&temp[13], 26);

  if(modo){
   temp[17]=0x30;
  }else{
   temp[17]=0x46;
  }
  temp[18]=0x31;
  temp[19]=0x13;//XOFF

  while(AS1_SendBlock(temp, 20, &i)!=ERR_OK){
  }
  while(fuera==0){
     while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
     }
     if(p==0x11){
      fuera=1;
     }
```

```
  }
}

void DibujaCorrecto(unsigned short indice, bool modo){
        unsigned char  temp[21];
        unsigned char p;
        unsigned short x, y;
        unsigned int i=0;
        bool fuera=0;


  x=CoordTableroX+((indice%8)* AnchoCasilla)+2;
  y=CoordTableroY+((indice/8)*AnchoCasilla)+2;

  temp[0]=0xDB;                         //Rectángulo relleno.
  Traduce(&temp[1], x);
  Traduce(&temp[5], y);
  Traduce(&temp[9], 25);
  Traduce(&temp[13],25);

  if(modo){
   temp[17]=0x31;
  }else{
   temp[17]=0x46;
  }
  temp[18]=0x31;

  temp[19]=0x13;//XOFF
  while(AS1_SendBlock(temp, 20, &i)!=ERR_OK){
  }
  while(fuera==0){
     while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
     }
     if(p==0x11){
      fuera=1;
     }
  }
}

void DibujaNivel(bool modo){

        unsigned char  temp[20];
        unsigned char p;
        unsigned short x=0, y=0;
        unsigned int i=0;
        bool fuera=0;



  temp[0]=0xDA;                         //Rectángulo
  Traduce(&temp[1], 5);
```

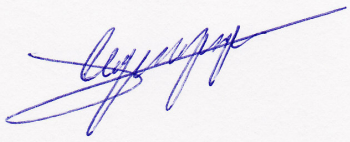
```
  Traduce(&temp[5], 3);
  Traduce(&temp[9], 100);
  Traduce(&temp[13], 23);

  if(modo){
   temp[17]=0x30;
  }else{
   temp[17]=0x46;
  }
  temp[18]=0x31;
  temp[19]=0x13;//XOFF

  while(AS1_SendBlock(temp, 20, &i)!=ERR_OK){
  }
  while(fuera==0){
     while((AS1_RecvChar(&p)!=ERR_OK)){   //XON
     }
     if(p==0x11){
      fuera=1;
     }
  }
}
```

En Bellaterra a 16 de Septiembre del 2009:

Manuel Vázquez Ortega



Ena un època com l' actual, en que es fa necessari disposar d' equips amb grans característiques y recursos per poder gaudir dels últims llançaments en jocs, no deixen de produir-se equips de baix cost basats en microcontroladors que poden entretenir al mateix nivell que els més cars. Aquest projecte reuneix el disseny de la configuració, interconnexió i programació de codi necessaris per a poder executar un joc senzill. Per a això s' ha utilitzat una pantalla LCD tàctil i un microcontrolador HCS12.

En una época como la actual, en la se hace necesario disponer de equipos con grandes características y recursos para poder disfrutar de los últimos lanzamientos en juegos, no dejan de producirse equipos de bajo coste basados en microcontroladores que pueden entretener al mismo nivel que los más caros. Este proyecto reúne el diseño de la configuración, interconexión y programación de código necesarios para ejecutar un juego sencillo. Para ello se ha utilizado una pantalla LCD táctil y un microcontrolador HCS12.

Nowadays, when it is necessary to have great feature computers to enjoy the latest releases in games, low-cost devices based on microcontrollers, can entertain similarly than the most expensive. This project meets the configuration design, networking and programming code needed to run a simple game. We use an LCD touch screen and a HCS12 microcontroler.