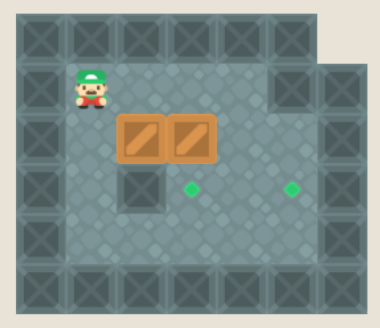
1.- JUEGO SOKOBAN

PROFESOR: ***FULGENCIO MONTILLA MEORO***



El trabajo consiste en desarrollar un programa en ensamblador 68k sobre el entorno EASY68K implementando una versión del conocido videojuego del Sokoban, un clásico del estilo Puzzle creado por el programador japonés Hiroyuki Imabayashi de la compañía Thinking Rabbit Inc.

El objetivo es que un personaje empuje las cajas del escenario hasta situarlas en unos puntos determinados, a ser posible en el menor número de empujes, de pasos y de tiempo

Desarrollo del juego

El juego se desarrolla en un escenario delimitado por un muro y en cuyo interior puede haber algunos obstáculos fijos, siendo el resto de libre circulación. En el mismo hay distribuidas unas cajas y están marcados unos puntos a donde se deben trasladar por el personaje a base de empujarla desde atrás, no siendo posible tirar de ellas.

Hay que tener en cuenta que las cajas no se podrán empujar más cuando estas chocan con algún obstáculo fijo (muro o interno) y tampoco es posible empujar una caja con otra (exceso de peso). Cuando las cajas no se pueden empujar por los motivos anteriores, también supondrán un obstáculo para el personaje en su movimiento. Cuando todas las cajas existentes son situadas en los puntos destino marcados al efecto, se habrá completado el nivel y se procede a iniciar otro nuevo reto de mayor dificultad (con más cajas o con caminos más complejos).

Algunas veces, por un error de la estrategia, la situación puede quedar imposible de seguir y en ese caso el juego te permite reiniciar a la posición inicial o bien deshacer uno o varios de los últimos pasos para corregir el procedimiento por la secuencia que el jugador vea más correcta.

Implementación del juego

Para la implementación del juego se recomienda disponer de un mapa de información que contenga la posición inicial y sea capaz de evolucionar actualizando la información conforme se van haciendo los movimientos correspondientes.

```
######
#@   ##
# $$  #
# #. .#
#     #
#######
```

```
* Variables and Strings
MAPA    DC.B '###### ',0
        DC.B '#@   ##',0
        DC.B '# $$  #',0
        DC.B '# #. .#',0
        DC.B '#     #',0
        DC.B '#######',0
```

Formato . XSB

Se sugiere esta estructura de datos porque proviene de un formato que más o menos se ha estandarizado para poder intercambiar escenarios (niveles) entre las múltiples implementaciones de este famoso juego. Serían archivos de texto que contienen esos caracteres y que se guardan con la extensión. XSB.

Ese mismo formato serviría también para almacenar la situación del juego en cada momento como una estructura de caracteres organizados en memoria y que equivalen a una matriz de ***m*** filas X ***n*** columnas, en este ejemplo concreto el escenario sería de 6 X 7

Los diferentes caracteres que se utilizan en ese formato serían:

- **#** −> muro u obstáculo
- **$** −> caja
- **.** −> punto de destino
- ***** −> caja sobre un punto de destino (no hay en este ejemplo, pero es una posibilidad)
- **@** −> hombrecito
- **+** −> hombrecito en un punto de destino (no hay en este ejemplo, pero es una posibilidad)

Para la implementación del juego se podría optar por una versión simplificada funcional con uno o varios escenarios de un solo tamaño y un número fijo de cajas, aunque para aspirar a calificaciones más elevadas debería ser más flexible y permitir diferentes tamaños, formas y número de cajas.

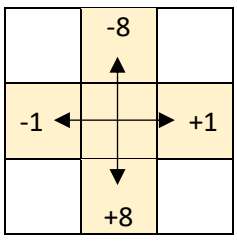
Para realizar los movimientos se podrá hacer mediante la pulsación de tecla (numérica u otras con las Tasks #4, #5 de las TRAP #15), mediante un movimiento de cursor con las flechas (Task #19 de las TRAP #15) o mediante el control del ratón que también se puede hacer con las funciones del sistema EASY68K (Task #61 de las TRAP #15).

La visualización se podrá realizar a partir de la información del mapa de recipientes, desde la manera más simple, directamente volcando la información del mapa en modo texto (Tasks #13 o #14 de las TRAP #15) o de formas más elaboradas mediante recursos gráficos que incluyen colores, líneas, rectángulos, círculos, etc. (Tasks #80 a #96 de las TRAP #15)

Para validar que los movimientos del personaje son válidos, así como los de las cajas empujadas se requiere comprobar que no hay obstáculos delante en la dirección correspondiente. Para ello hay que consultar los elementos del mapa en las posiciones implicadas. Para evaluarlas, además de manejar el contenido del mapa con ayuda de los modos de direccionamiento indirecto, se debe buscar un algoritmo que nos permita saber las posiciones de memoria de las casillas en las cuatro direcciones posibles (izquierda, derecha, arriba o abajo) y para ello se sugiere la siguiente idea para posicionar o mover personaje o cajas dentro del mapa, así como poder detectar posibles obstáculos. Si distribuimos los datos de manera secuencial y los interpretamos en dos dimensiones, FILA y COLUMNA, tendremos la relación:

Posición casilla = MAPA + FILA*(Nº COLUMNAS) + COLUMNA

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31
4	32	33	34	35	36	37	38	39
5	40	41	42	43	44	45	46	47



En el ejemplo que consiste en una matriz 6 X 7, vemos que los movimientos a izquierda y derecha corresponderían a moverse a la posición anterior y posterior (-1 y +1 respectivamente) mientras que arriba y abajo sería disminuir 8 posiciones o aumentar 8 posiciones (-8 y +8 respectivamente), una más de las columnas disponibles para contar con los 0's finales. Con eso podemos a partir de una posición determinada, observar si tenemos algún obstáculo en la dirección del movimiento con solo comprobar qué contiene esa casilla (dirección de memoria) adyacente. Para otros mapas con diferente número de columnas, cambiaríamos el 8 por el número de columnas correspondientes más uno (n+1)

Así como muchas implementaciones permiten deshacer movimientos para evitar situaciones sin salida, sería opcional pero muy valorable el implementarlo también en esta versión para Easy68k. Para ello se puede optar desde la opción menos complicada que sería retroceder todo al inicio del nivel y que solo requiere recuperar el estado inicial, o bien por la capacidad de deshacer uno por uno todos los movimientos. Eso requiere haber almacenado en una cadena de memoria, todos los movimientos realizados y codificados de alguna manera que luego permitan a demanda, realizarlos en orden inversa. Como eso sería lo más complicado, se podría optar por una solución intermedia suficiente que solo guardara los movimientos de las cajas y no los del personaje, ya que a fin de cuentas son las cajas las que una vez se mueven a donde no se debe, imposibilitan la consecución del objetivo.

Opcionalmente se podría añadir un sistema que cuente y presente los pasos totales, los empujes de las cajas o bien el tiempo y ser capaz de registrar las mejores marcas

Otra opción para valorar sería incorporar un constructor de escenarios personales que permita situar los obstáculos, las cajas y los puntos de colocación.

Ejemplos de funcionamiento e información complementaria

Hay numerosas implementaciones y variantes del juego del Sokoban, aunque en esencia es el mismo y solo cambian los skins y los niveles disponibles. Los que utilizan el formato XSB, permiten también intercambiar niveles de juego entre ellos.

http://www.rodoval.com/heureka/sokoban/sokoban.html .- Aquí hay mucha información acerca de la historia y del juego del Sokoban así como muchos enlaces a otras páginas con versiones en línea y de mapas de juego

https://www.minijuegos.com/juego/sokoban-online .- Esta es una implementación sencilla y escenarios pequeño

http://www.game-sokoban.com/ .- Otra versión con enlace a una app similar para móvil

https://www.sokobanonline.com/ .- Una página con muchos escenarios y con consejos para resolverlos

http://www.abelmartin.com/rj/sokoban_prog1.html .- Otra recopilación de diferentes versiones en línea y de escritorio, así como mucha información acerca del juego y sus múltiples variantes

https://ksokoban.online/ .- Versión Online del ksokoban que es la versión disponible para Linux. Dispone de numerosísimos niveles y estilos diferentes

http://sokoban.e-contento.com/ .- Versión en Javascript descargable o jugable online

https://www.granvino.com/jam/stuff/juegos/yasminuroban/spanish/index.htm .- Otra versión online

http://jose-juan.computer-mind.com/jose-juan/Resolver-Sokoban.php .- Curioso programa escrito en C++ que resuelve escenarios de Sokoban por fuerza bruta (comprobando todas las posibilidades)

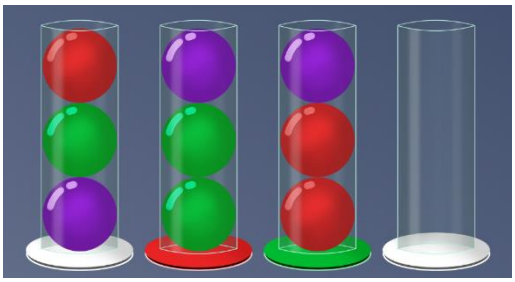
2.- JUEGO SORT-IT

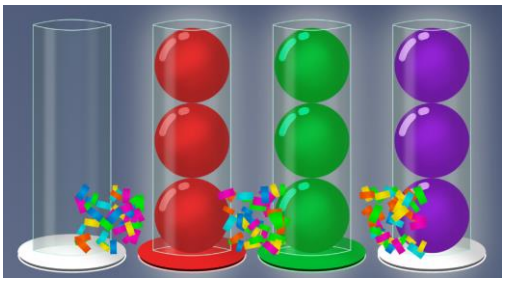
PROFESOR: ***FULGENCIO MONTILLA MEORO***

El trabajo consiste en desarrollar un programa en ensamblador 68k sobre el entorno EASY68K implementando una versión simplificada de un videojuego de estilo Puzzle denominado "***Sort it***" en el que se dispone de una serie de recipientes y unas bolas de colores que se deben ordenar de manera que deben finalizar con completando todos ellos con bolas de un solo color excepto uno que quedará vacío.

Desarrollo del juego

El juego se desarrolla a partir de una situación inicial (diferente en cada nivel) de bolas colocadas de forma desordenada dentro de los recipientes que pueden ser de iguales o distintas capacidades, y a partir de ahí se puede ir trasladando cualquiera de las bolas que esté en la posición más alta de cualquier recipiente a la posición más alta de cualquier otro recipiente que tenga capacidad libre (los recipientes con bolas actúan como pilas LIFO, "Last In, First Out"). Para hacerlo basta señalar el recipiente origen y después el destino y se producirá el traslado de la bola siempre que el primero tuviera alguna bola y el segundo no estuviera ya completo. Esto habrá que ir haciéndolo hasta que se consiga que cada recipiente esté completo con bolas de un único color quedando vacío uno de ellos.





Conforme se van completando niveles, aparecen nuevas dificultades como puede ser aumentar el número de recipientes o sus capacidades y el número de bolas, o tener recipientes de distinta capacidad, así como un número de bolas también diferente de cada color, de manera que quizás no en todos ellos cabrían las bolas de un determinado color.

Otra restricción más avanzada sería el que solo permita que un recipiente termine lleno de bolas de un color específico, de manera que, si lo llenásemos de bolas de otro color, no serviría para completar el nivel. Esta restricción se observa en el color marcado en la base del recipiente que debe coincidir con el color de las bolas que terminen completando su capacidad. Si el color de la base es el blanco, significa que no impone restricción en el color del conjunto de bolas con el que se pueden llenar. En principio, aunque un recipiente tenga un color en la base, ese recipiente podría llenarse de bolas de ese color o también podría ser el que quedara vacío para finalizar el nivel, sin embargo, si hay un recipiente con la base negra, ese tiene que ser obligatoriamente el que ha de quedar vacío.

Implementación del juego

Para la implementación del juego se podría optar por una versión simplificada funcional con un número fijo de recipientes y de capacidad de estos, con las reglas básicas de traslado bolas y control de capacidad, pero sin añadir restricciones ni variaciones avanzadas, es decir, algo así como implementar solo uno de los primeros niveles. No obstante, para aspirar a calificaciones más elevadas se deberían añadir algunas de las variaciones y restricciones comentadas y que estas se vayan añadiendo confirme se van superando los diferentes niveles, al igual que en el juego real.

Como sugerencia, el programa podría almacenar la información de cada distribución de recipientes y bolas en una ***estructura de datos*** como la que se muestra a continuación. Cada recipiente sería como un vector cuyos componentes equivalen a las bolas de colores que contienen. Cada número representa un color y el 0 sería un hueco vacío. Así mismo, para indicar el color de la base del recipiente y no confundirlo con una bola, le sumaríamos 10. Para saber que no hay más recipientes, se puede marcar el final con algún valor no utilizado, por ejemplo, en este caso con $FF.

```
* Rojo=1 Verde=2 Naranja=3
* Fondo Blanco=10 Fondo Rojo=11
* Mapa de recipientes
*
RECIP1 DC.B 2,1,1,10
RECIP2 DC.B 3,2,3,11
RECIP3 DC.B 3,2,1,10
RECIP4 DC.B 0,0,0,10
FINAL  DC.B $FF
```

El traslado de las bolas consistiría en manejar esa estructura de datos con ayuda de los modos de direccionamiento indirecto. Coger una bola de un recipiente sería buscar el primer valor que sea distinto de 0 y dejarla en otro recipiente consiste en copiar ese valor sobre el último 0 del destino (si lo hay)

La selección de los recipientes origen y destino se podrían hacer indicándolo mediante la pulsación de una tecla (numérica u otras con las Tasks #4 o #5 de las TRAP #15), mediante un movimiento de cursor con las flechas o mediante el control del ratón que también se puede hacer con las funciones del sistema EASY68K (Task #61 de las TRAP #15).

La visualización se podrá realizar a partir de la información del mapa de recipientes, desde la manera más simple, directamente en modo texto mediante letras o números (Tasks #3, #6, #11 de las TRAP #15) o de formas más elaboradas mediante recursos gráficos que incluyen colores, líneas, círculos, etc. (Tasks #80 a #96). En la implementación gráfica no sería necesario emular las animaciones originales al detalle ya que puede suponer una complejidad que podría superar la intención de la realización de este trabajo

Una opción sin embargo que sería interesante es añadir una cuenta de tiempo y/o movimientos realizados hasta alcanzar los objetivos e incluso la capacidad de registrar marcas.

Ejemplos de funcionamiento

Como ejemplo de referencia para observar el funcionamiento del juego se puede visitar:

https://www.crazygames.com/game/sort-it

Otras versiones en aplicaciones móviles serían:

https://play.google.com/store/apps/details?id=com.game.sortit3d

https://apps.apple.com/es/app/sort-it-3d/id1493125671

3.- Imágenes en Escala de Grises

PROFESOR: ***IVÁN PATRAO HERRERO***

El trabajo consiste en desarrollar un programa, en ensamblador 68k sobre el entorno EASY68K, capaz de trabajar con imágenes en escala de grises.

Una imagen en escala de grises es, básicamente, una matriz de datos de las dimensiones n·m de la imagen (siempre que no se encuentre en algún formato con compresión de datos, como jpeg). El valor de cada celda expresa la intensidad del píxel individual. Por ejemplo, con una profundidad de color de 8bits, un valor $00 implica que el píxel está completamente negro, mientras que un valor $FF supone un píxel blanco.

El histograma de una imagen es una representación en la que se indica cuántos píxeles hay de cada valor de intensidad. Se trata de una herramienta útil en fotografía que permite aplicar ciertas correcciones sobre la imagen para balancear las zonas claras/oscuras (https://www.blogdelfotografo.com/histograma/).

Por otro lado, cuando se incrementa el tamaño de una imagen se necesita mayor cantidad de píxeles que en la imagen original. El programa de escalado de la fotografía debe ser capaz de darles un valor de intensidad a esos nuevos píxeles (https://www.cambridgeincolour.com/tutorials/image-interpolation.htm).

La imagen a procesar, de tamaño 10x10, se proporciona al final de este documento, como una serie de 100 valores que representan de manera consecutiva los niveles de gris de 10 filas de 10 píxeles cada una (los primeros 10 valores corresponden con la primera fila íntegra).

Así, debe realizarse un programa con las siguientes funciones:

1) Histograma de la imagen proporcionada, mostrado de forma gráfica con una resolución en el eje horizontal de, al menos, 10 intervalos.
2) Escalado de la imagen: preguntar al usuario qué tamaño de imagen desea (manteniendo las proporciones) en el rango 10 – 80 píxeles de ancho y realizar el escalado de la imagen, rellenando los nuevos píxeles creados mediante interpolación lineal.

Opcional:

a) Alcanzar en el histograma una resolución horizontal de 255 divisiones
b) Implementar algún método de interpolación avanzado (bicúbica, etc).
c) Mostrar la imagen original y escalada por pantalla

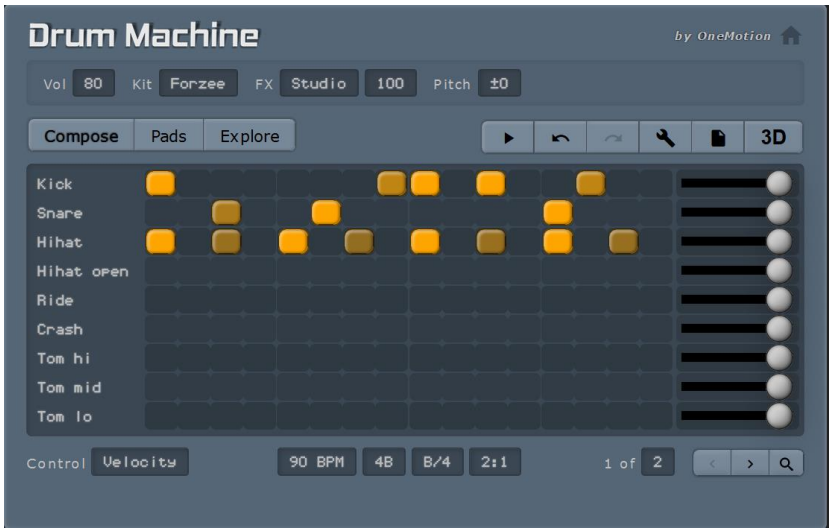
Anexo: Tabla de datos de la imagen.

```
IMAGEN DC.B $80, $80, $80, $80, $80, $80, $80, $80, $80, $80
       DC.B $80, $00, $80, $FF, $FF, $FF, $FF, $80, $00, $80
       DC.B $80, $80, $FF, $FF, $FF, $FF, $FF, $FF, $80, $80
       DC.B $80, $80, $FF, $FF, $80, $80, $FF, $FF, $80, $80
       DC.B $80, $80, $80, $80, $80, $80, $FF, $FF, $80, $80
       DC.B $80, $80, $80, $80, $FF, $FF, $FF, $80, $80, $80
       DC.B $80, $80, $80, $80, $FF, $FF, $80, $80, $80, $80
       DC.B $80, $80, $80, $80, $80, $80, $80, $80, $80, $80
       DC.B $80, $00, $80, $80, $FF, $FF, $80, $80, $00, $80
       DC.B $80, $80, $80, $80, $80, $80, $80, $80, $80, $80
```

4.- Caja de ritmos. Secuenciador de audio

PROFESOR: ***DAVID LORENTE***

Este trabajo consiste en desarrollar un programa en ensamblador 68k sobre el entorno ***EASY68K*** implementando una versión básica de un secuenciador de audio, caja de ritmos o "beat-box".



Una caja de ritmos reproduce secuencialmente distintos sonidos para crear un patrón rítmico.

Algunos ejemplos para ayudar a comprender el funcionamiento de una caja de ritmos:

https://www.onemotion.com/drum-machine/

https://splice.com/sounds/beatmaker

https://drumbit.app/

https://musiclab.chromeexperiments.com/Song-Maker/

Se reproducirán secuencialmente ocho sonidos seleccionados por el usuario a una velocidad también seleccionada por el usuario.

El usuario debe introducir por medio de texto:

1. Qué sonido, codificado como un número, corresponde a cada compás.
2. El tempo en beats por minuto (bpm).

Se deberán reproducir al menos 8 compases, y debe existir además la posibilidad de no reproducir ningún sonido (silencio) en cualquier compás.

Deben estar disponibles al menos 12 sonidos diferentes, además del silencio, para que el usuario pueda seleccionar el que desee. Cada sonido se puede utilizar tantas veces como se necesite, pero sólo se reproducirá un único sonido en cada compás (secuenciador monofónico).

Los ficheros de audio seleccionados (.wav) se deben reproducir secuencialmente a la velocidad introducida por el usuario en bpm.

Se usarán dos pulsadores como controles de 'play' y 'stop' para poner en marcha o parar la reproducción.

Se usará un switch para indicar reproducción en modo loop: cuando se llega al último compás se vuelve a reproducir el primero, hasta que el usuario pare el secuenciador.

POSIBLES MEJORAS:

Visualización gráfica del secuenciador, con indicación en una cuadrícula de cada uno de los sonidos seleccionados en cada compás.

Seguimiento gráfico del secuenciador en tiempo real mediante una barra vertical que indique qué compás se está reproduciendo.

Opción de generar distintos grupos de ocho compases y combinarlos para crear variaciones sobre un mismo patrón.

Opción de cargar distintos bancos de sonidos por parte del usuario (Drums, Electrónica, Instrumentos…)

Opción de cambiar el tempo mientras se está reproduciendo la secuencia.

5.- GESTIÓN DE ENTRADAS AL BIOPARC

PROFESOR: ***ANTONIO MARTINEZ MILLANA***

El trabajo consiste en desarrollar un programa en ensamblador 68k sobre el entorno EASY68K implementando un sistema para gestionar la recaudación de las entradas al Bioparc de Valencia

Existen dos tipos de entradas: acceso al parque (20€) y acceso al parque con guía (30€). Además, con un suplemento de 10€ se puede adquirir un ticket para el restaurante del parque.

El parque proporciona descuentos del 50% para menores de 10 años y jubilados (65+), y de un 20% para menores de 18 años, aplicables a todo el valor de la entrada adquirida (incluido el ticket restaurante). El sistema también permite realizar devoluciones.

El programa debe permitir operativa diaria al personal del Bioparc, de manera que pueda registrar las ventas y devoluciones que realiza, así como analizar los movimientos de caja. Por ello al principio de cada operación se le pedirá al usuario que indique la operación quiere hacer (menú inicial): realizar venta, realizar devolución o analizar caja.

Para el registro de movimientos de venta y de devolución, las especificaciones son:

La caja registradora debe almacenar un máximo de 20 operaciones. Para cada una de ellas debe registrarse:

a) Tipo de operación: venta o devolución.
b) El tipo de entrada: acceso al parque o acceso al parque con guía y si se adquiere el ticket restaurante.
c) Edad del usuario
d) Se mostrará el importe de venta o devolución.
e) Se mostrará el tipo de descuento correspondiente.
f) Se mostrará el importe del descuento, calculado sobre el importe.
g) El total de la venta o devolución (total-descuento).
h) En caso de venta, importe abonado por el cliente
i) Si procede, la devolución.

Deberá mostrarse por pantalla un dialogo con el usuario para registrar y mostrar la información de manera intuitiva. Al finalizar el registro se volverá al menú inicial.

Para el análisis de caja, las especificaciones son:

El usuario podrá elegir que quiere visualizar: las ventas, las devoluciones o el análisis agregado (ventas-devoluciones). Deberá mostrarse un informe descriptivo de todas las operaciones realizadas para la opción seleccionada. Para las ventas y las devoluciones deberá mostrarse el total de las operaciones, número de usuarios en cada grupo de edad, total de descuentos aplicados, etc. Para el análisis agregado se mostrará la diferencia del total recaudado menos el importe devuelto. Las analíticas podrán mostrarse agrupando los datos en función de tipo de producto (a elección del usuario).

Ampliación: El sistema será capaz de registrar una venta o devolución múltiple. Es decir, cada operación podrá contener la venta o devolución de varias entradas (de distintos tipos). No se podrán registrar operaciones de venta y devolución simultáneamente. Para ello se deberá preguntar al usuario si quiere finalizar la operación o desea añadir otra venta/devolución.

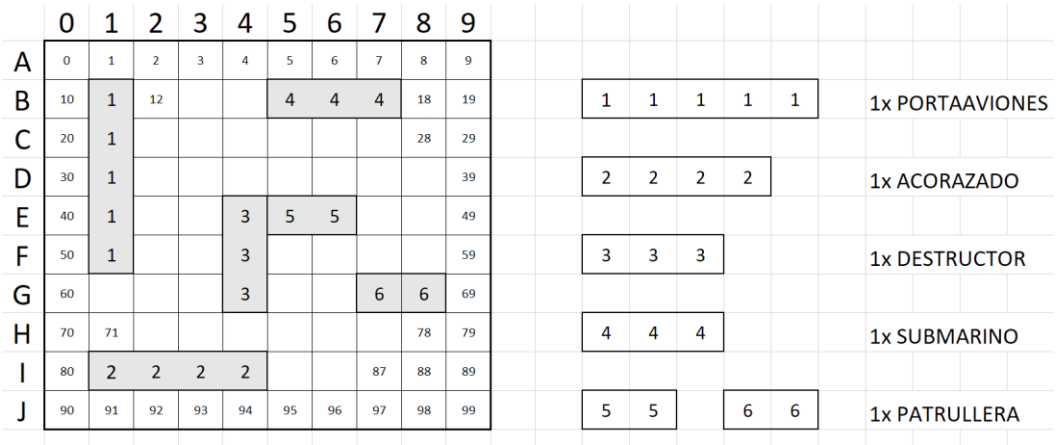
6.- HUNDIR LA FLOTA, POR COMPUTADOR (Electronic Battleship)

PROFESOR: ***JOSÉ VICENTE GARCÍA NARBÓN***

El trabajo consiste en desarrollar un programa en ensamblador 68k, sobre el entorno EASY68K, implementando una versión del clásico juego de estrategia HUNDIR LA FLOTA (*Battleship*).

Es un juego mundialmente conocido, jugado en su versión en lápiz y papel desde antes de la primera guerra mundial. Fue comercializado por diferentes compañías en ese formato en la década de 1930. La primera versión comercializada como juego de mesa, incluyendo un tablero, miniaturas representativas de los barcos y pines de plástico, fue introducida en 1967 por Milton Bradley (MB). Posteriormente se han comercializado versiones en tablero con control electrónico del juego, video juegos, juegos para web y aplicaciones para teléfono móvil.

Descripción del juego



Cada jugador dispone en su tablero de dos cuadriculas que representan el mapa de los barcos del jugador y el mapa de disparos realizados contra el otro jugador. Cada una de las cuadriculas está dividida en casillas. Las cuadrículas tienen generalmente un tamaño de 10 x 10, y cada una de las casillas está identificada por un número para las columnas y una letra para las filas.

En el mapa de barcos, el jugador distribuye sus barcos (sin que el oponente lo vea) y registra los disparos que realiza el oponente. En el mapa de disparos, el jugador registra sus disparos (como “agua”, si no ha alcanzado a un barco del adversario, “tocado” si sí lo ha hecho, o “hundido” cuando se ha disparado a todas las casillas correspondientes a un barco.

Una vez comenzado el juego, los jugadores no pueden mover sus barcos de posición. Cada jugador se alterna en la realización de disparos, salvo cuando uno de los jugadores alcanza una casilla ocupada por un barco del oponente. En ese caso el jugador que ha “tocado” o “hundido” un barco del rival, por lo general puede volver a disparar. El jugador que primero hunda la flota de su rival será el ganador.

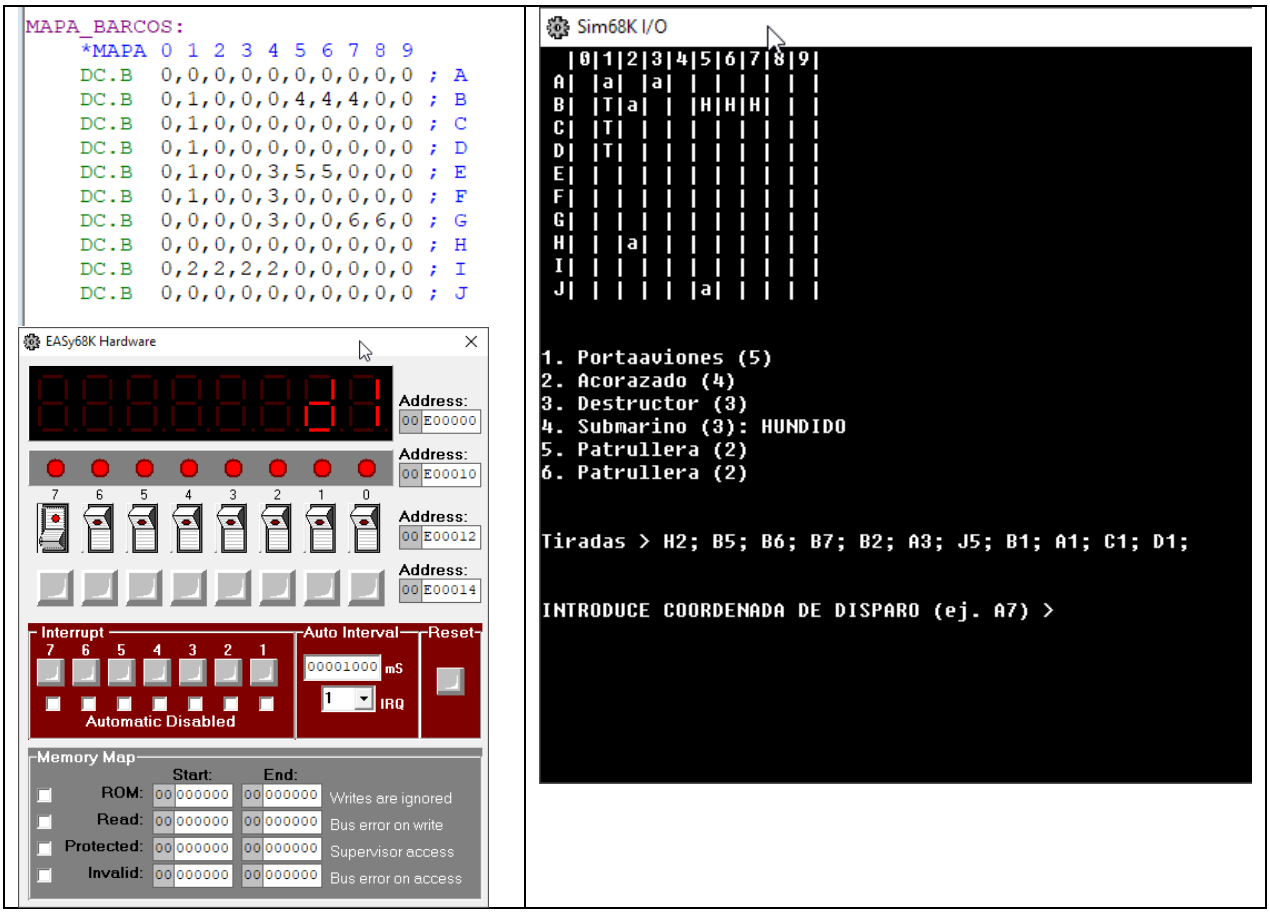
Aunque existen diferentes variantes de las reglas y del juego, la flota de barcos disponible habitualmente consiste en: un Portaaviones de tamaño 5 casillas, un Acorazado de tamaño 4 casillas, un Destructor de tamaño 3 casillas, un Submarino de tamaño 3 casillas y dos Patrulleras de tamaño 2 casillas. Los barcos deben distribuirse en el tablero de forma que queden completamente dentro del tablero, orientados horizontal o verticalmente (no en diagonal) y sin que se solapen ni se crucen.

Implementación del juego

El juego se iniciará cuando este se habilite mediante uno de los interruptores disponibles en el simulador. En una primera versión, se implementará parte del juego de un único jugador. Se deberá realizar el programa necesario para la introducción de la coordenada de disparo, la comprobación contra el mapa de barcos, y la representación por la salida del simulador del mapa y el resultado de los disparos.

En esta versión básica, la posición de los barcos en el mapa se podrá introducir directamente mediante la programación de una estructura en memoria que almacene convenientemente su posición. El programa desarrollado comprobará la coordenada introducida en la tirada frente al mapa almacenado, determinando si el resultado de la tirada ha sido "agua", "tocado" o "hundido". El mapa y su representación por la salida del simulador se irá actualizando con cada tirada, indicando el resultado de esta. El programa verificará que las coordenadas de disparo sean válidas y no se repitan, indicando que se vuelvan a introducir en caso de error.

Las coordenadas de la última tirada realizada se verán reflejadas en dos dígitos de los displays 7 segmentos. En caso de que el resultado de la tirada sea "tocado", también se deberá señalizar mediante los leds del simulador, por ejemplo, realizando un desplazamiento de un bit por todos los leds, a izquierda y derecha. Cuando un barco sea "hundido" deberá realizarse una indicación diferente. Por ejemplo, encendiendo y apagando todos los leds de forma intermitente durante varios segundos. Esta señalización hará uso de las interrupciones periódicas. Como mejoras, también se pueden añadir a estos efectos la reproducción de sonidos característicos del juego para el disparo y el alcance a los barcos. (TASK 70-77 de TRAP 15)



En las figuras anteriores se muestra un ejemplo de estructura de memoria para el mapa de barcos donde se almacenar la posición, así como ejemplos de una posible representación en modo texto por la salida del simulador.

Sobre esta funcionalidad básica se plantean a continuación POSIBLES MEJORAS:

- INTRODUCCIÓN MANUAL MAPA DE BARCOS: Desarrollar e implementar un método para introducir los barcos en el mapa de batalla de forma manual, a través de la entrada de texto. Por ejemplo mediante la activación del modo carga usando un interruptor, e introduciendo un listado de coordenadas o solicitando la posición al usuario para cada barco.
- INTRODUCCIÓN MANUAL MAPA DE BARCOS CON COMPROBACIÓN: Sobre el punto anterior, incorporar al programa las comprobaciones necesarias para garantizar que las posiciones introducidas son válidas, evitando barcos situados fuera del mapa de batalla o solapes entre los mismos.
- JUEGA CONTRA LA MÁQUINA: Incorporar al programa las funcionalidades necesarias para poder jugar contra la máquina, añadiendo un segundo mapa de barcos y las funcionalidades necesarias (inteligencia Artificial) para que la máquina realice tiradas (a partir también de la generación de números aleatorios). Una versión mejorada de esta IA podría ser una versión que busque en las casillas adyacentes en caso de "tocado" o que analice las casillas restantes y busque dónde podrían esconderse los diferentes barcos en función de sus tamaños.
- GENERACIÓN ALEATORIA MAPA DE BARCOS: Incorporar al programa la funcionalidad necesaria para que la generación del mapa de barcos sea aleatoria. Se partirá de la generación de un número pseudoaleatorio (se entregaría al alumno una subrutina ya programada para generar el numero pseudoaleatorio). Para cada posición y orientación generadas se realizarían las comprobaciones oportunas para verificar que el posicionado es correcto (solapes, barcos fuera de mapa, etc.…).
- MODO GRÁFICO: Representación del mapa de barcos y disparos en modo gráfico por la salida del Simulador. Representando de forma gráfica la cuadrícula del mapa de barcos, indicando las casillas dónde se han realizado las tiradas y el resultado de forma diferente al modo texto sugerido. (TASK 80-96 de TRAP 15).
- MODO GRÁFICO: Detección de la coordenada donde se va a realizar la siguiente tirada sobre el mapa de batalla gráfico, realizando la indicación mediante el uso del ratón (TASK 60-62 de TRAP 15).