Algoritmos y Estructuras de Datos II

Parcial 03-05: Tema C - Batalla Naval

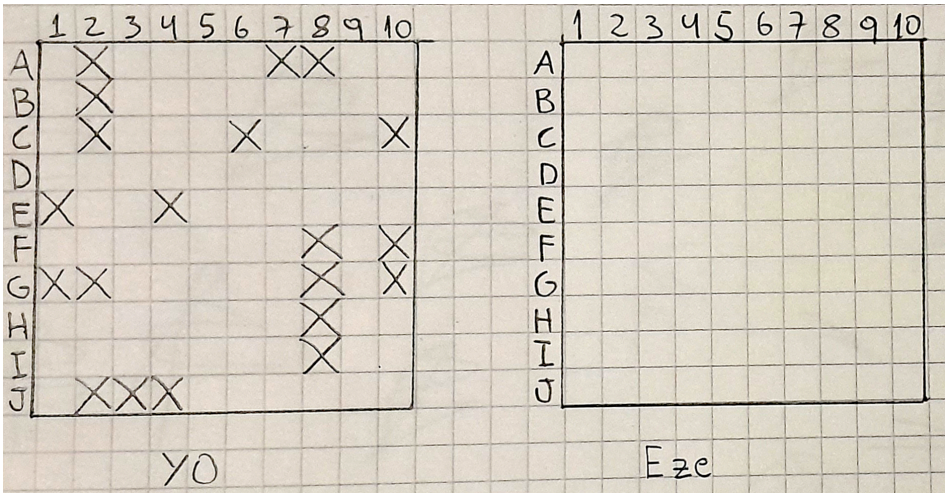
El juego "Batalla Naval" (Battleship en inglés) es un juego que puede jugarse con lápiz y papel. No es necesario saber jugar para poder completar los ejercicios, pero sí entender cómo es el tablero por lo que se explicará brevemente el juego.

Juego y Tablero

Consiste en una cuadrícula (comúnmente de 10x10) donde las filas se nombran con letras comenzando con la letra "A" (hasta la "J" si es 10x10) y las columnas con números comenzando desde 1. Al inicio del juego se elige donde posicionar los barcos. Cada barco puede estar orientado vertical u horizontalmente (no se vale en diagonal!) y ocupan cierta cantidad de celdas consecutivas. En el caso de una cuadrícula de 10x10 los barcos deben ser:

- Un barco que ocupe 4 celdas
- Dos barcos de 3 celdas
- Tres barcos de 2 celdas
- Cuatro barcos de 1 celda

Los barcos no deben "tocarse" entre sí (deben estar separados al menos por una celda). Cada participante tiene en su hoja la grilla con sus barcos y además la grilla de sus oponentes para ir marcando los barcos que encuentra.



El objetivo de cada participante es destruir los barcos de su o sus contrincantes. Aunque lo más común es jugar de a dos, se puede de a más personas. En la imagen de ejemplo se ve una partida (que aún no inició) de a dos donde se juega contra "Eze" (Ezequiel).

Se establece un orden para iniciar y se lo mantiene durante todo el juego. Quien empiece debe elegir un lugar de la grilla a donde realiza su disparo (por ejemplo "A 8!"). Luego el resto debe indicar si el disparo fue certero o no diciendo "agua" en caso que no haya impactado en un barco, "averiado" si acertó parcialmente a un barco o "hundido" si terminó de hundir un barco. Luego se procede al siguiente participante, que deberá hacer el mismo procedimiento. Una vez que se completan los disparos de cada participante se pasa

a la próxima ronda y todo se repite.

Ejercicio 1: Implementación de un Tablero de Batalla Naval

El programa que se va a implementar consiste en almacenar la información de una partida que posiblemente aún no se haya terminado (se guarda un juego en proceso). Entonces en una matriz multidimensional se guardará la grilla de cada participante, donde cada celda de la matriz guarda el contenido (agua o barco), el estado (si se disparó a ella o no), y la ronda (si se le disparó en la ronda n, se guarda ese número). Entonces la matriz tiene tres dimensiones, la primera para las filas, la segunda para las columnas y la tercera para participantes (players).

En el directorio del ejercicio se encuentran los siguientes archivos:

Archivo	***Descripción***
`main.c`	Contiene la función principal del programa y **se debe completar** para el ej2.
`cell.h`	Declaraciones relativas a la estructura de las celdas y de funciones de carga y escritura de datos.
`cell.c`	Implementaciones **incompletas** de las funciones.
`array_helpers.h`	Declaraciones / prototipos de las funciones que manejan el tablero.
`array_helpers.c`	Implementaciones **incompletas** de las funciones que manejan el arreglo.
`ansicolors.h`	Definiciones auxiliares para usar color en la consola.

Abrir el archivo **`battles/battle1.in`** para ver cómo se estructuran los datos.

Cada línea del archivo contiene los datos correspondiente a una coordenada (fila, columna), incluyendo por cada participante (en este caso dos) los datos de la celda con esas coordenadas en su grilla. El formato de la línea es el siguiente:

(<fila>**,** <columna>**)**	**[**<contenido> <estado> <ronda>**]**	**[**<contenido> <estado> <ronda>**]**

para tener en cuenta:

- La coordenada **`<fila>`** es un caracter **`'A'`**, **`'B'`**, **`'C'`**, ... tantos como filas tenga la grilla.
- La coordenada **`<columna>`** es un número entero sin signo que va de **`1`** hasta la cantidad de columnas de la grilla.
- Luego de las coordenadas, separado por un espacio, se encontrarán los datos para la celda perteneciente a **`player1`** y a continuación los datos para la celda de **`player2`**.
- Los datos de cada celda son tres:
    - **`<contenido>`**: un caracter que puede ser **`'w'`** que indica que hay agua (*water*) o **`'s'`** que indica que hay parte de un barco (*ship*).
    - **`<estado>`**: un entero que indica con **`0`** que la celda no recibió disparos y con **`1`** indica que sí se le disparó.
    - **`<ronda>`**: un entero sin signo que indica en qué ronda la celda recibió un disparo (si no se le disparó tendrá un **`0`**).

Consideraciones:

- Se debe asumir sin culpa que hay exactamente 2 participantes.
- La cantidad de coordenadas cargadas (líneas en el archivo) será exactamente **`MAX_COORDS`.**
    - Si hay menos datos el programa debería comunicar un error.
    - Queda a criterio decidir cómo manejar el caso en el cual los archivos de entrada contengan datos de más.
- No es necesario verificar que el archivo tenga coordenadas repetidas.
- Las líneas en el archivo pueden aparecer en cualquier orden, líneas consecutivas no necesariamente son de celdas contiguas.

El ejercicio consiste en completar el procedimiento de carga de datos en los archivos **`array_helpers.c`** y **`cell.c`**. Los datos deben cargarse de manera correcta en el arreglo usando los índices adecuados. Entonces por ejemplo en **`battleship_board[1][4][0]`** se deberían haber cargado los datos de la celda correspondiente a **`player1`** que en el archivo de entrada figuraba con coordenada **`(B, 5)`**.

Recordar que el programa tiene que ser *robusto*, es decir, debe tener un comportamiento bien definido para los casos en que la entrada no tenga el formato esperado.

Una vez completada la lectura de datos se puede verificar si la carga funciona compilando,

```
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 -c array_helpers.c cell.c main.c
$ gcc -Wall -Werror -Wextra -pedantic -std=c99 array_helpers.o cell.o main.o -o battleboard
```

y luego ejecutar por ejemplo

```
$ ./battleboard inputs/battle1.in
```

esto resultará en que se muestren los datos leidos del archivo:

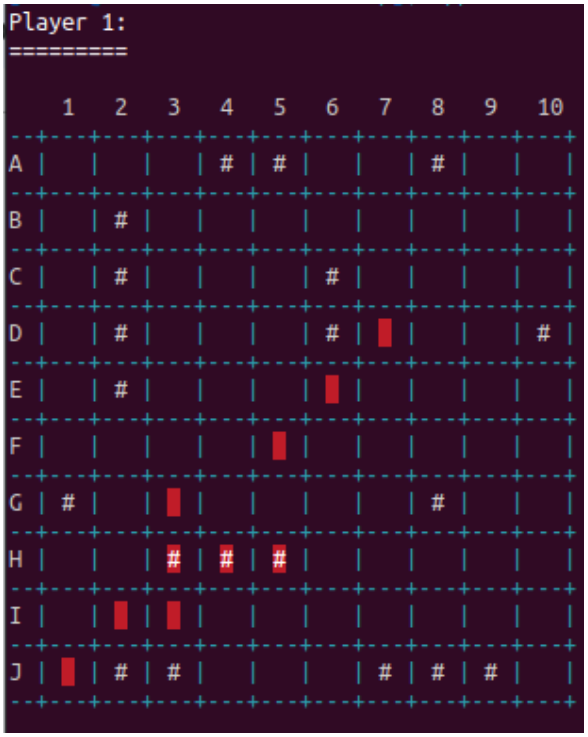
```
(A,  1) player1: water [untouched] (round  0) | raw content: (0,0,0)
(A,  1) player2: water [untouched] (round  0) | raw content: (0,0,0)
(A,  2) player1: water [untouched] (round  0) | raw content: (0,0,0)
(A,  2) player2: ship  [untouched] (round  0) | raw content: (1,0,0)
(A,  3) player1: water [untouched] (round  0) | raw content: (0,0,0)
(A,  3) player2: water [untouched] (round  0) | raw content: (0,0,0)
(A,  4) player1: ship  [untouched] (round  0) | raw content: (1,0,0)
(A,  4) player2: ship  [untouched] (round  0) | raw content: (1,0,0)
(A,  5) player1: ship  [untouched] (round  0) | raw content: (1,0,0)
(A,  5) player2: ship  [untouched] (round  0) | raw content: (1,0,0)
(A,  6) player1: water [untouched] (round  0) | raw content: (0,0,0)
(A,  6) player2: ship  [untouched] (round  0) | raw content: (1,0,0)
(A,  7) player1: water [untouched] (round  0) | raw content: (0,0,0)
(A,  7) player2: water [untouched] (round  0) | raw content: (0,0,0)
(A,  8) player1: ship  [untouched] (round  0) | raw content: (1,0,0)
(A,  8) player2: ship  [untouched] (round  0) | raw content: (1,0,0)
(A,  9) player1: water [untouched] (round  0) | raw content: (0,0,0)
(A,  9) player2: water [untouched] (round  0) | raw content: (0,0,0)
(A, 10) player1: water [untouched] (round  0) | raw content: (0,0,0)
(A, 10) player2: water [hit!     ] (round 10) | raw content: (0,1,10)

(B,  1) player1: water [untouched] (round  0) | raw content: (0,0,0)
(B,  1) player2: water [untouched] (round  0) | raw content: (0,0,0)
                          (:)
```

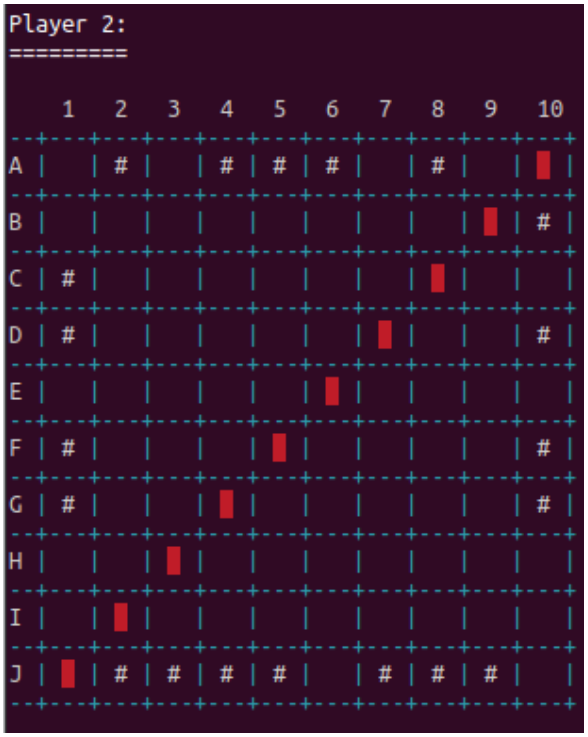
Otra forma de usar el programa, que puede servir para asegurarse que esté cargando bien la información es usar la opción **`draw`**:

```
$ ./battleboard inputs/battle1.in draw
```

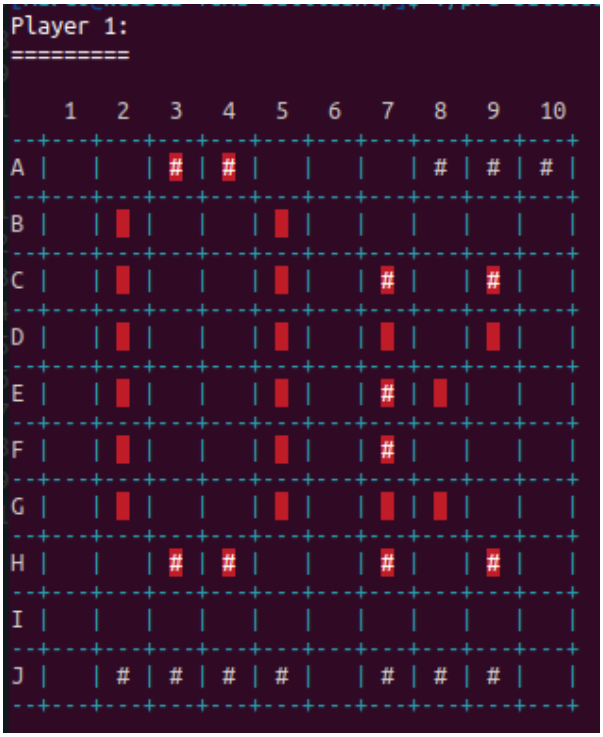
la salida será en forma de grilla donde se dibujan los tableros mostrando en rojo las celdas que recibieron disparos. A continuación dos ejemplos para que puedan verificar:

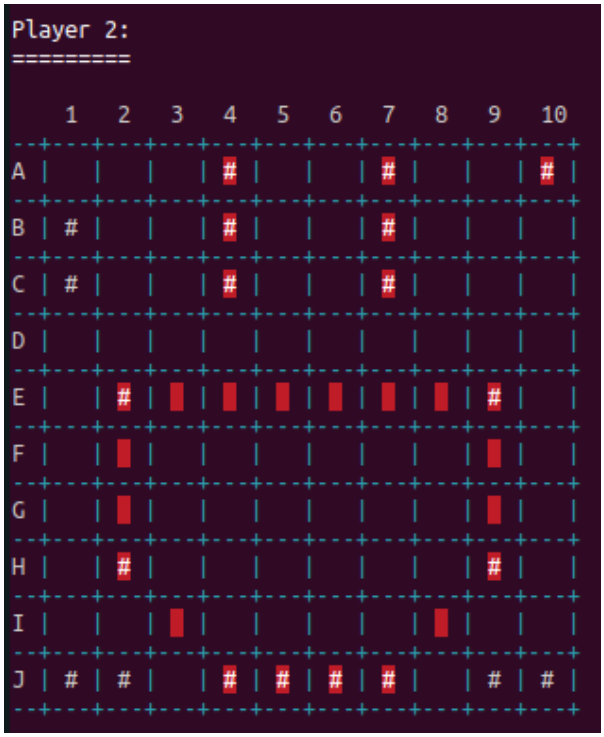
```
$ ./battleboard inputs/battle1.in draw
```






```
$ ./battleboard inputs/battle3.in draw
```





Notar que es más cómodo para verificar, pero aquí no se visualiza la información de las rondas. Si se utiliza la opción **`sequence`** se puede ver paso a paso cómo evoluciona el juego:

```
$ ./battleboard inputs/battle1.in sequence
```

de igual manera a modo de verificación es más práctico el método **`draw`**.

Los archivos **`battle1.in`**, **`battle2.in`** y **`battle3.in`** tienen los datos ordenados por coordenada, sin embargo no se debe asumir este orden por lo que se incluyen los archivos **`battle1_unsorted.in`**, **`battle2_unsorted.in`** y **`battle3_unsorted.in`** que contienen las mismas partidas que los originales pero con los datos desordenados. Los resultados para un archivo ordenado y su versión desordenada deben ser los mismos.

Además se incluyen ejemplos de batallas inválidas: **`battle_invalid1.in`**, **`battle_invalid2.in`**, etc; que tienen inconsistencias y que al intentar cargarse el programa debería abortar su ejecución.

Ejercicio 2: Análisis de los datos

Completar la siguiente función, definida en **`array_helpers:`**

```
unsigned int get_points(Board board, player_t player, unsigned int round);
```

Esta función debe retornar la cantidad de puntos que tiene **`player`** teniendo en cuenta el juego hasta la ronda **`round`**. Cada vez que un disparo acierta a un barco se suman 100 puntos. Recordar que los disparos en la grilla de **`player1`** fueron efectuados por **`player2`** y viceversa, tenerlo en cuenta cuando se haga el cáluco del puntaje.

Finalmente modificar el archivo **`main.c`** para que se muestre el puntaje de **`player1`** y **`player2`** teniendo en cuenta todas las rondas.

Archivo	Puntos player1	Puntos player2
battles/battle1.in	0	300
battles/battle2.in	700	400
battles/battle3.in	1500	1000