Battleship - Sprint 1

A lo largo de este semestre, deberás implementar cinco versiones (sprints) diferentes de dificultad incremental del juego "Hundir la Flota" (Battleship).

Deberás **enviar una solución intermedia**, **antes del 22 de marzo a las 13:00 que contenga los tres primeros *sprints***, y una solución final el 9 de mayo. Ambas entregas deben ser enviadas, y ambas deben tener una calidad aceptable, para tener oportunidad de aprobar el curso por evaluación continua ordinaria.

1 El juego básico

Battleship es un juego muy popular de **dos jugadores** que cuentan con una flota de barcos cada uno y deben disparar a la flota del oponente hasta hundir completamente la flota. Cada jugador dispone de **dos tableros** que usa para marcar la posición de sus barcos en uno y los disparos realizados a la flota del oponente en otro.

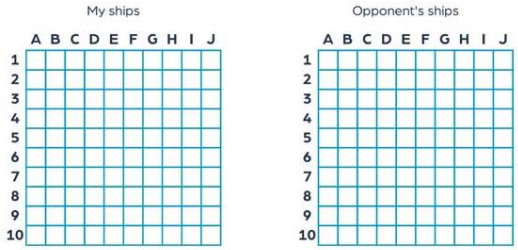


Figura 1: Típicos tableros para jugar a batalla de barcos

Los jugadores alternan turnos, realizando un disparo a una casilla del tablero de los barcos del oponente, indicando para ello las coordenadas de dicha casilla.

El objetivo del juego es destruir la flota del jugador oponente hundiendo todos sus barcos.

2 Implementación básica del juego

Los jugadores serán un usuario humano (*user*) que juega contra un usuario artificial, la máquina (*computer*).

El juego se lleva a cabo a través de la terminal. Es decir, cualquier interacción con el usuario humano (imprimir los tableros, disparar y mensajes de error, así como entradas del usuario) serán hechas a través de la consola.

Los barcos serán colocados inicialmente en coordenadas **fijas** tanto para la flota del usuario como para la flota de la máquina.

2.1 Configuración del juego

La aplicación guardará el estado del juego usando dos tableros, cada uno almacena inicialmente los barcos de un jugador y posteriormente se añadirán los disparos del oponente. Al comienzo serán colocados diferentes tipos de **barcos**. El resto de las celdas contendrán **agua**.

El tamaño del tablero así como el número y longitud de los barcos será fija. Los tableros son cuadrados, con tamaño fijo de 10×10 en esta primera versión, y cada casilla individual en el tablero está identificada por su columna y su fila.

Cada flota consistirá en **varios barcos de diferentes longitudes**; esto es, cada barco ocupa diferente número de casillas en el tablero. En este primer sprint, las flotas contienen **un barco de Batalla** (longitud cuatro), **dos Cruceros** (longitud tres), **tres Destructores** (longitud dos) y **cuatro Submarinos** (longitud uno). Nótese que cada uno tiene longitud diferente y hay varios de cada tipo.

Al comienzo del juego, la aplicación creará dos tableros y **colocará una flota de barcos en cada tablero, en las mismas coordenadas para ambos**.

2.2 El tablero

Los tableros serán implementados a través de una rejilla (*grid*) como un array de enteros:

- Cada tipo de barco se representa por un número del 1 al 4 (1 para **S**ubmarinos, 2 para **D**estructores, 3 para **C**ruceros y 4 para barcos de **B**atalla) **mientras que no hayan sido disparados**.
- **Cuando se dispara a una casilla con barco**, la aplicación reemplaza el contenido por **un número negativo con el mismo valor absoluto**.
- **Las Casillas con agua no disparadas** contienen un **0**, mientras que las **casillas con agua ya disparadas** contienen un **-10**.

1	0	1	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	0
2	2	0	2	2	0	2	2	0	0
0	0	0	0	0	0	0	0	0	0
3	3	3	0	3	3	3	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	4	4	4	4	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

Array 1: Contenido inicial del array

2.3 Los jugadores

Cada jugador tiene un nombre y mantendrá referencias a ambos tableros. Un tablero estará referido como ***myShips*****,** y contendrá los **barcos del propio jugador, así como los disparos del oponente**. El otro tablero, ***opponentShips*****,** contendrá la **flota del oponente, así como los disparos hechos por el propio jugador sobre el oponente.**

El otro jugador contendrá referencias a los mismos tableros, pero en el orden opuesto. Esto es, el tablero *myShips* de un jugador será el tablero *opponentShips* para el otro jugador.

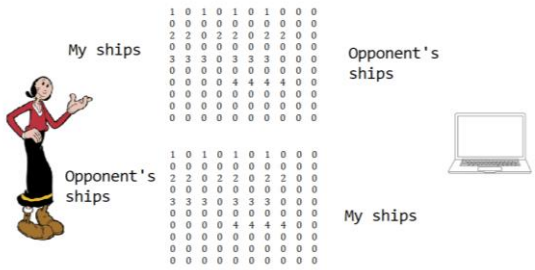


Figura 1: Contenido de los tableros al comienzo del juego

2.4 Juego

Los jugadores **realizan turnos de disparos eligiendo una posición (coordenada) y disparando en el tablero del oponente**. La aplicación responde con **"*HIT*"** o con "***MISS***", según sea el resultado (tocado o agua) y cambia el contenido de la cuadrícula objetivo de manera apropiada, como se indica en el apartado anterior.

Por ejemplo, dada la situación inicial del Array 1, si Olivia dispara F-8, se refiere a la columna 5, fila 7 del Array. Como el oponente (computer) no tiene ningún barco localizado en esa posición del Array, la aplicación responderá con Miss, y se deberá asignar el valor -10 a dicha posición dentro del **tablero del oponente**.

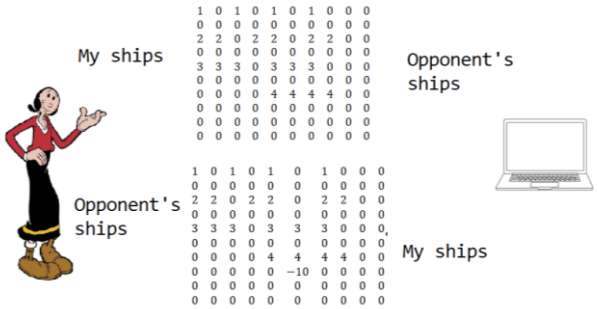


Figura 3: Contenido de los tableros después de disparar F-8

Una vez que un jugador ha usado su turno y se ha procesado, **le toca al otro jugador**, aún en el caso de que el primero haya logrado disparar a un barco.

Tan pronto como todos **los barcos de un jugador hayan sido hundidos**, el **juego finaliza** y **el oponente gana** el juego.

2.5 Procesamiento de un turno

Un turno de un jugador consiste en las **acciones** siguientes, **independientemente de que el jugador en turno sea el usuario o la máquina**. La única diferencia está en el paso 2. El jugador usuario siempre tiene el primer turno.

1. **Mostrar ambos tableros** desde la perspectiva del usuario. En el propio muestra sus barcos y los disparos de la máquina, en el del oponente muestra los disparos que él hizo sobre el tablero de la máquina. (ver apartado 3.2 y 3.3).
2. El jugador al que le toca jugar **elige la casilla objetivo** (coordenada) en el tablero del oponente para disparar. Cómo elegir este disparo es la única acción diferente para usuario (que introduce la posición por teclado) y máquina (que se genera aleatoriamente).
3. El jugador **realiza el disparo** sobre el tablero de barcos del oponente (recuérdese que el tablero *opponentShips* para el usuario es el mismo tablero que *myShips* para el oponente). Nótese que no te advierte cuando un disparo se refiere a una coordenada ya disparada. Es procesada justo como cualquier otra.
4. Si un barco (o un trozo de él) **ocupa** (**u ocupó**) la casilla que dispara el usuario, la aplicación debe anunciar un disparo mostrando ***"HIT"*** por consola. En otro caso, si **no ocupa (ni ocupó)** la casilla ningún barco, sino que hay agua, **se muestra** "***MISS***" por consola.
5. **Se comprueba si hay ganador** (si todos los barcos del oponente han sido hundidos) en cuyo caso imprime un mensaje de felicitación, o bien se cambia el turno de juego si no hay ganador todavía.

2.6 Modo de juego

Battleship se puede configurar para jugar en **modo normal** o en **modo de depuración** y el modo de juego afecta sólo a la visualización, no al procedimiento del juego en sí.

El juego se puede jugar en modo normal o de depuración. En la configuración de la ejecución se incluirá el parámetro para iniciar el juego de una forma u otra, según tu voluntad. Si se ejecuta en modo normal el tablero del oponente se mostrará de forma muy restringida. En el modo de depuración, el tablero del oponente se muestra dejando ver todo su contenido, por lo que el usuario puede probar el juego y hundir la flota de la computadora rápidamente.

3 Interfaz con el Usuario

Esta sección describe la forma en que se debe mostrar en pantalla los tableros propios y cómo el usuario introduce coordenadas y cualquier salida producida por la aplicación. Esta versión usará una **interfaz de usuario basada en texto**.

3.1 Imprimir tableros de usuario

Imprime ambos tableros en consola desde la perspectiva del usuario: *myShips* y *oppoentShip* del usuario. Se deben imprimir **uno al lado del otro**, **con una línea de título identificando cada tablero**, como se indica en la figura 4, rellenando cada casilla con el símbolo correspondiente según se indica en la tabla 1, y separando las casillas por una barra horizontal |.

La siguiente tabla muestra los **caracteres para mostrar en el tablero**. Nótese que hay diferentes caracteres para cada tipo de barco

Displayed square	Description
BBBB	Battleship (cada casilla a B)
CCC	Cruiser (cada casilla a C)
DD	Destroyer (cada casilla a D)
S	Submarine
*****	Hit
Ø	Missed shot (\u00F8, conjunto vacío)
Espacio en blanco	En el tablero de la izquierda, Casillas agua; en el de la derecha, aquellas que no se descubrieron todavía

*Tabla 1: Caracteres **para visualizar el contenido** de las cuadrículas*

3.2 Mostrar tablero *myShips*

Usa B, C, D y S para mostrar barcos y blancos para mostrar casillas que indican agua.

Cuando el jugador *computer* dispara a uno de los barcos del usuario, se mostrará el carácter '*' en lugar de la letra correspondiente). Los disparos perdidos hechos por el jugador *computer*, se mostrarán con el carácter 'Ø'. Imprime '|' para separar columnas. Imprime una cabecera con las letras para reconocer cada columna. Recuerda también imprimir el número de fila al comienzo de la impresión de cada fila.

3.3 Mostrar tablero *opponentShips* en modo normal

La salida del tablero del oponente depende de un parámetro del juego (modo de juego).

Por defecto, jugando en **modo NORMAL**, **solo se muestran casillas que el usuario ha disparado**: si contuvieron un barco, usando el carácter '*' mientras que disparos perdidos serán mostrados usando símbolo 'Ø'. El resto de las casillas, es decir **aquellas que aún no se han disparado, serán mostradas como blancos**.

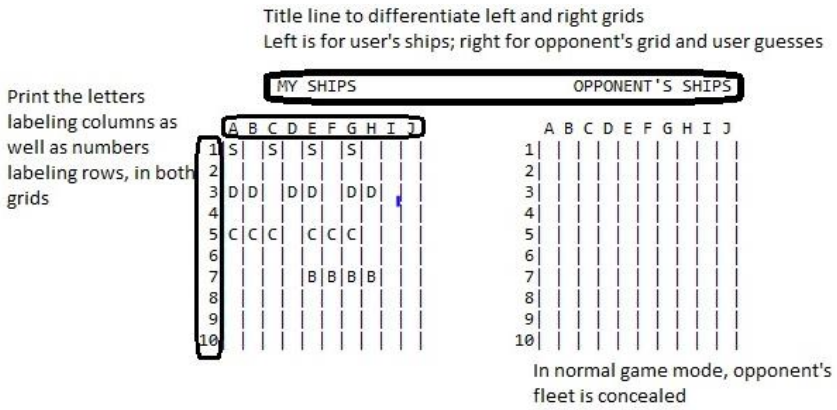


*Figura 4: Pantalla inicial del juego en **modo Normal***

3.4 Mostrar tablero *opponentShips* en modo Depuración

Si el **modo** de juego es **DEPURACIÓN**, entonces **se mostrarán también los barcos del oponente** del mismo modo que los barcos del propio usuario.

El modo de juego afecta únicamente a la manera de mostrar, no al procedimiento del juego en sí mismo.

En cada paso del juego, serán siempre mostrados los barcos del oponente (flota de la máquina). Así que, después de algunos turnos, este podría ser un ejemplo de visualización:

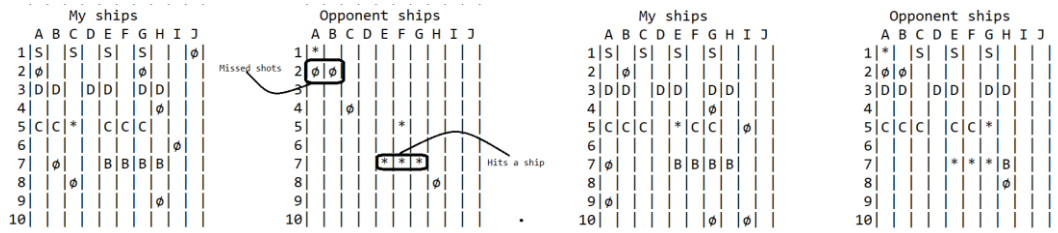


*Figura 5: Ejemplo de salidas después de algunos disparos en modo **normal** (izquierda) y modo **depuración** (derecha)*

3.5 Entrada del usuario

Cuando se pide al usuario que introduzca las coordenadas, se referirá a la fila y columna objetivo, utilizando la letra y el número que se visualiza en pantalla (figura 6). **Puedes asumir, en este sprint, que el usuario nunca tecleará coordenadas fuera del tablero**.

Nótese que hay correspondencia entre las letras mostradas y las columnas indexadas del array, así como los números escritos por el usuario y las filas indexadas del array. Así cuando el usuario escribe columna A fila 4, se traduce en la columna 0 del array (letra A) la fila 3 del array (fila 4 en pantalla).

3.6 Gestión de la interacción con el usuario

Aparte de mostrar los tableros del usuario habrá mensajes para gestionar la interacción con el usuario.

- Imprime un mensaje mostrando el **nombre del jugador que tiene el turno**.
- Cuando es el turno del Usuario, **pregúntale por las coordenadas del siguiente disparo**. Pregunta por la columna (letra) primero; después la fila (número).
- Cuando es el turno de la máquina, únicamente **imprime las coordenadas generadas aleatoriamente**, en lugar de leer la entrada.
- Finalmente imprime **HIT** o bien **MISS** (dependiendo del resultado del disparo).

Por favor, intenta ceñirte a las siguientes figuras.

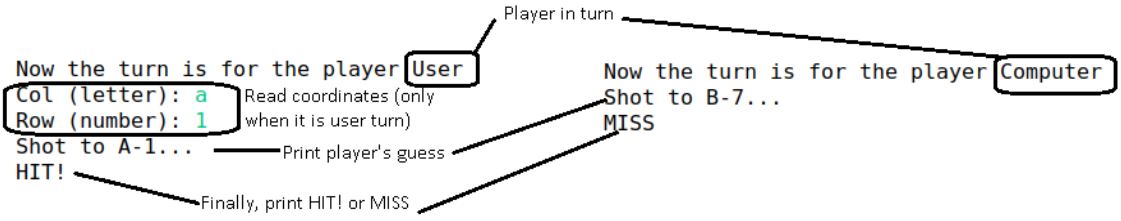


Figura 6: Interacción con el Usuario utilizando entrada/salida estándar

La aplicación se comporta igual cuando es el turno de la máquina, excepto que genera una coordenada de disparo (**no se generará nunca una coordenada de disparo ya realizado en otra jugada**) en lugar de leer la entrada desde teclado.

4 Qué hacer

4.1 UML

Después de leer las secciones anteriores un par de veces y antes de implementar, dibuja un diagrama de clases UML que modele las relaciones entre las siguientes clases del proyecto: *Main, Game, Board, HumanPlayer, ComputerPlayer y Coordinate*.

Para cada clase muestra también atributos y métodos públicos. Intenta dibujar el diagrama sin líneas cruzadas, cuando sea posible. Las líneas pueden ser horizontales, verticales o combinar ambas con ángulo de 90 grados. **Puedes usar *Visual Paradigm*** para el diseño u otra herramienta.

4.2 Proyecto inicial proporcionado

Junto con este documento se proporciona un proyecto Java mínimo (esqueleto). Contiene la clase Main con el método main() para iniciar el juego.

Cambia el nombre del proyecto como *apellido1_apellido2_nombre _battleship_sprint1* e implementa las siguientes clases como se describe a continuación. Lo que sigue es la descripción de los únicos métodos públicos para cada clase, pero puedes (y debes) usar tantos métodos y campos privados como sea necesario.

4.3 Ejecución del proyecto

El esqueleto proporciona el contenido completo del método main() que no tendrás que modificar.

El método main recibe una cadena como parámetro. Este parámetro se incluirá a partir del menú Run/Configurations/JavaApplication/Arguments. Si el valor del argumento es la cadena "debug" en mayúsculas, minúsculas o cualquier combinación, el juego se iniciará en modo de depuración. Si es "normal" se iniciará en modo normal. Ten en cuenta que, si no incluyes el argumento, el programa dará un error y finalizará.

4.4 Clase Game

Ejecuta el bucle principal del juego y es la única clase responsable de la interacción con el usuario.

Métodos públicos:

`Constructor Game(HumanPlayer human, ComputerPlayer computer)`

Se asignan los dos jugadores recibidos, se crean dos tableros de tamaño 10x10, y se les asignan a ambos jugadores. Cada jugador tendrá su tablero y el tablero del oponente. En este primer Sprint del juego, ambos tableros serán iguales inicialmente, es decir, tendrán su flota localizada en las mismas posiciones.
Además, se debe crear un objeto selector del turno, y un objeto *presenter* de tipo ConsoleWriter que será el único encargado de imprimir cualquier información en consola.
El juego se inicia en modo Normal.

`void setDebugMode(boolean mode)`

Asigna el modo de juego. Si el argumento es false, se coloca a modo normal (la flota de la máquina estará oculta). Si es true, el modo de juego será depuración que permitirá en su momento mostrar las coordenadas de la flota del oponente.

`void play()`

Es el único responsable de la interacción con el usuario, así como la gestión del bucle principal. En este bucle, los jugadores alternan turnos de disparo mientras no haya ganador.
No importa el turno del jugador, la aplicación se comporta de la misma forma. Cada turno de juego hará lo siguiente:

- Se muestran los dos tableros como se muestra en la figura 4.
- Se muestra mensaje indicando a quién corresponde el turno (figura 6)
- Se generan nuevas coordenadas (automáticamente cuando es el turno de la máquina o se recoge desde la entrada estándar cuando es el turno del usuario)
- Se muestran las coordenadas generadas (figura 6)
- Se realiza el disparo en esas coordenadas en el tablero del jugador oponente.
- Se muestra el resultado del disparo. Se imprime un mensaje con la información apropiada ("Hit" si hay un barco en esas coordenadas del oponente, o "Miss" cuando no hay barco en ellas.
- Se comprueba si hay ganador, en cuyo caso se imprime mensaje de felicitación, o bien se cambia el turno de juego si no hay ganador todavía.

4.5 Class Board

Esta clase representa un tablero cuadrado donde la aplicación colocará los barcos. Para ello, almacena la **flota**, una lista de barcos, y la localización de los barcos en un array de dos dimensiones de enteros (la cuadrícula (**grid**)). El contenido de cada celda o elemento de la matriz da información sobre el estado del juego en esa coordenada.

0: casilla agua no disparada; no hay barcos colocados en ella y no ha sido disparada.

Desde 1 a 4: hay un barco o pieza de barco en esta casilla. Si es 1, el barco es un Submarino. En otro caso es parte de un barco mayor. (se describen todos en el apartado 2.2).

De -1 a -4: esta pieza de barco fue disparada.

-10: un disparo perdido (un disparo a casilla con agua).

Métodos públicos:

`Constructor Board()`

Este constructor crea un tablero de 10*10 con la flota de 10 barcos (**un Barco de Batalla** (longitud cuatro), **dos Cruceros** (longitud tres), **tres Destructores** (longitud dos) y **cuatro Submarinos** (longitud uno) usando BoardBuilder.

`int getSize()`

Devuelve un número de filas y columnas de la cuadrícula (la dimensión).

`boolean shootAt(Coordinate coordinates)`

Guarda un disparo en esas coordenadas. Si la coordenada contiene un barco (un valor mayor que 0), el método cambia el contenido al mismo valor en negativo (para indicar tocado) y devuelve true. Posteriores disparos sobre esta posición dejarán el mismo valor negativo en la casilla y devuelve true.
Si en la posición no hay barco (contendrá el valor 0), el método devuelve false y cambia el valor a -10. Posteriores disparos sobre esa misma casilla devolverán también false y dejarán el valor -10.

`boolean isFleetSunk()`

Decide si todos los barcos de la flota están hundidos, en cuyo caso devuelve true; en caso contrario devuelve false.

`char[][] getFullStatus()`

Devuelve un array de 2D de caracteres representando el estado del tablero. Un carácter en las coordenadas (x,y) representa el estado de esa casilla, como se describe en la tabla 1 y los apartados para mostrar tableros.

`char[][] getMinimalStatus()`

Como el anterior, devuelve un array 2D de caracteres. Sin embargo, devuelve el valor actual de aquellas casillas en la cuadrícula que han sido ya disparadas. Aquellas que aún no han sido elegidas, devuelven un carácter blanco, independientemente de que contengan un barco o no.

`int[][] getInnerArray()`

Método **protegido** (usado para realizar los test). Devuelve una copia del propio array (grid). El tipo Array tiene un **método público clone()** que devuelve una copia del mismo array.

4.6 Class HumanPlayer

Se trata de la clase que almacena el estado del juego del jugador humano. Contiene referencias a dos tableros, objetos de tipo *Board* (*myShips* y *myOpponentShips*).

Métodos públicos:

`Constructor HumanPlayer(String name)`
Crea un nuevo objeto para un jugador humano y guarda el nombre del jugador. Este nombre debe siempre ser diferente de *null*, y de cadena vacía; **en otro caso, lo denominaremos "*user*"**.

`String getName()`
Devuelve el nombre del jugador

`void setMyShips(Board board)`
Asigna el parámetro recibido al tablero *myShips* que contendrá sus barcos y los disparos del enemigo, con el parámetro que recibe.

`void setOpponentShips(Board board)`
Asigna el parámetro recibido al tablero *opponentShip* que almacena los disparos propios y los barcos del oponete.

`Board getMyShips()`
Devuelve el tablero que guarda *myShips* (con mis barcos y los disparos del oponente).

`Board getOpponentShips()`
Devuelve el tablero *opponentShips* (con los barcos del oponente y mis disparos).

`boolean hasWon()`
Devuelve true si la flota del oponente ha sido hundida; false en otro caso.

`boolean shootAt (Coordinate Coordinate)`
Dispara sobre el tablero del oponente. Devuelve true si el disparo toca un barco y false en otro caso.

`Coordinate makeChoice()`
Lee las coordenadas por teclado y devuelve un objeto de tipo *Coordinate*. Debes suponer que el usuario nunca introducirá valores incorrectos de las coordenadas.

4.7 Class ComputerPlayer

Es casi lo mismo que la clase anterior (*HumanPlayer*). Las únicas diferencias son en el constructor y el método *makeChoice*.

`Constructor ComputerPlayer(String name)`
Crea un nuevo objeto y guarda el nombre del jugador. Si el argumento es *null* o cadena vacía o en blanco lo denominaremos "***computer***".

`Coordinate makeChoice()`
Genera aleatoriamente coordenadas **que no hayan sido disparadas anteriormente**. Se eligen entre la existentes en la lista de coordenadas que no hayan sido disparadas aún.

4.8 Class TurnSelector

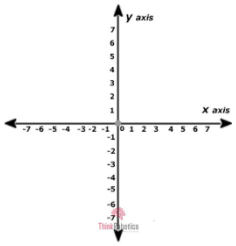
Maneja los turnos de juego. La primera vez siempre será turno del Usuario. A partir de ahí, los turnos se alternarán, aun cuando un jugador toca o hunde a un barco del oponente.

Métodos públicos:

`Constructor TurnSelector()`
Crea un nuevo objeto para alternar los turnos de juego entre dos jugadores.

`int next()`
Devuelve alternado 1 o 0. Cada número representa un turno de jugador diferente: 1 es para el usuario, 0 es para la máquina. La primera vez, el turno es siempre para el Usuario.

4.9 Class Coordinate



Esta clase representa una posición en el tablero en un array de dos dimensiones.

Métodos públicos:

`Constructor Coordinate (int x, int y)`
Crea un objeto de tipo *Coordinate*, a partir de la referencia **x para la columna, e y para la fila**. Esta clase está pensada para cualquier valor de x e y. Por tanto no se debe validar la entrada.

`int getCol()`
Devuelve el valor de la columna.

`int getRow()`
Devuelve el valor de la fila.

`String toString()`
Sobreescribe el método *toString* y devuelve el valor de las coordenadas con el formato: Coordenada [x = 0, y = 1] (por ejemplo)

`String toUserString()`
Devuelve el valor de las coordenadas en un formato más adecuado para el usuario. En el caso de la Coordenada [x = 2, y = 0] este método devuelve "C-1".

`boolean equals(Object o)`
Sobreescribe el método equals y devuelve verdadero si el argumento es una coordenada con los mismos valores para x e y que este objeto. Es muy recomendable utilizar Eclipse para generar esta función (Fuente → Generar código hash y es igual…).
Este método podría ser necesario en las pruebas cuando se necesita comprobar si alguna coordenada es igual a otra.

4.10 Class BoardBuilder

Esta clase construye un array cuadrado de dos dimensiones de enteros, y lo rellena colocando números del 0 al 4 en, en posiciones fijas.

Está completamente implementada en el esqueleto.

4.11 Class ConsoleWriter

Esta clase contiene los métodos para visualizar el estado del juego.

Métodos públicos:

`public void showGameStatus(Board left, Board right, boolean debugmode)`
Muestra los tableros (el del usuario a la izquierda, la de la máquina a la derecha) de acuerdo con el modo de juego. Intenta ceñirte a la descripción que aparece en la sección 3.3 tanto como te sea posible.

`public void showWinner (String name)`
Imprime un mensaje con el nombre del ganador. **Los mensajes esperados están comentados en la clase ConsoleInteraction**.

`public void showGameOver()`
Imprime un mensaje para la finalización del juego.

`public void showTurn (String name)`
Imprime un mensaje con el nombre del jugador que tiene el turno.

`public void showShootingAt (Coordinate position)`
Imprime una version de las coordenadas más apropiada para el usuario.

`public void showShotMessage (Boolean damage)`
Imprime el mensaje HIT! o MISS, dependiendo del parámetro.

5 Tests

A continuación, se describen los test JUnit que debes incluir. Los casos de uso serán comprobados como se describe más abajo. Implementa métodos de test preparando el contexto, ejecutando los métodos correspondientes y comprobando los resultados esperados, en cualquier caso.

5.1 Clase Board

`Método shootAt`
Casos de uso:
1. Después de disparar una casilla aún no disparada que no contienen barco, el método devolverá falso y la casilla en cuestión se marcará como disparada (-10).
2. Después de disparar una casilla ya disparada que originalmente no contenía un barco, el método debe devolver false y la casilla permanecerá como disparada (-10).
3. Después de disparar una casilla aún no disparada que contiene un barco, el método devolverá true y la casilla cambiará a disparada (su valor en negativo).
4. Después de disparar una casilla disparada que originalmente contenía un barco, el contenido permanecerá igual y el método devolverá true.

`Método isFleetSunk`
Casos de uso:
1. Todos los barcos están hundidos. Devuelve true.
2. Todos los barcos, salvo uno que le falta una posición sin disparar, están hundidos. Devuelve false.
3. Varios barcos no han sido disparados. Devuelve false.

`Método getNotFiredCoordinates`
Casos de uso:
1. Un tablero sin haber sido aún usado, devuelve todas las casillas.
2. Un tablero que ha sido disparado completamente, devuelve la lista vacía
3. Tablero en el que han sido disparadas varias casillas, devuelve el resto de casillas.

Para facilitar la realización de los test, podría resultar útil lo siguiente:

- El tipo Array tiene un método público clone() que devuelve una copia del mismo array. Puedes usarlo para crear un método que devuelva una copia de grid.
- Usa assertArrayEquals() para comprobar si dos arrays son iguales.
- Para implementar los test de getNotFiredCoordiantes es necesario implementar el método equals en la clase Coordinate. Eclipse lo hará por tí. Usa el menu de Eclipse IDE para generar el equals() y hashCode() usando 'Source->Generate hashCode()and equals().
- Puedes comparar el contenido de dos ArrayLists, ignorando el orden, usando:
  `assertTrue(first.size() == second.size() && first.containsAll(second) && second.containsAll(first));`

6 Requerimientos mínimos para calificar tu trabajo

Antes de entregar, asegúrate de que **tu proyecto se ejecuta sin errores.** Es decir, debes desarrollar una versión simple pero que funcione.

Aparte de estos requisitos mínimos, ten en cuenta lo siguiente:

- Debes escribir Junit para todos los métodos descritos en la sección de test y **deben todos deben pasar la ejecución correctamente**.
- Tu programa **no debe producir ningún tipo de mensaje** o texto a través de la salida estándar o de error estándar **más que los indicados en el enunciado**.
- **Todas las clases deben estrictamente encajar con las interfaces públicas escritas en el esqueleto**: no debes cambiar la signatura de ningún método (nombre del método, número, tipo y orden de los parámetros, y tipo de datos devuelto) o su comportamiento.
- Debes respetar los **nombres de los paquetes, jerarquía de paquetes y las clases que pertenecen a cada paquete**.
- A menos que se indique lo contrario, la clase principal no debe cambiarse.
- A menos que se indique explícitamente lo contrario, la validación de argumentos debe implementarse para métodos públicos. En esta etapa, simplemente lanza **IllegalArgumentException** junto con un **mensaje adecuado** como:
  - `Null is an invalid value for the argument`
  - `Empty String is an invalid value for the argument`
  - `Blank String is an invalid value for the argument`
  - `<valor de argumento> is an invalid value for the argument`

7 Envío del trabajo

No tienes que entregar este primer sprint para calificación, pero será la base para el segundo y subsecuentes sprints. Es altamente aconsejable implementarlo en 2 semanas, antes de que se publique sprint2 (ver calendario).