Práctica POE. Juego de barcos

La *batalla naval* (*juego de barcos* o *hundir la flota*) es un juego tradicional de estrategia y algo de suerte, que involucra a dos participantes.

En la práctica a realizar, varios usuarios deberán poder conectarse mediante un programa *cliente* a un programa *servidor* para jugar en red al *Juego de Barcos*. Para ello, el *servidor* deberá ofrecer un *servicio* que empareje los usuarios y controle el transcurso del juego de cada pareja de jugadores.

Los programas *cliente* y *servidor* se implementarán utilizando la biblioteca `POE_library.jar`, tal y como se ha visto en las sesiones de prácticas previas. Pero, el programa *cliente* no podrá limitarse a un simple menú para que el usuario lance una operación cualquiera del servicio, siendo necesario adaptar éste a las exigencias que impone el propio desarrollo del juego.

Para facilitar la realización del *servicio* se proporciona su interfaz (`JuegoBarcos.java`) un esqueleto de su implementación (`Servicio.java`) con toda la información que es necesario mantener (exceptuando la requerida para resolver problemas de concurrencia) y la biblioteca documentada `componentes.jar` que proporciona varias clases para crear y manejar objetos fundamentales para el juego, como, por ejemplo: los *tableros*, las *celdas* que componen un tablero o los *barcos*.

Descripción de una partida entre dos jugadores

Los dos jugadores manejan un tablero océano (donde colocan sus barcos) y un tablero de tiro (donde anotan los tiros efectuados al océano del oponente y los barcos alcanzados). Por defecto, un tablero es un cuadrado de 10x10 celdas (o casillas), identificándose la posición de estas en el tablero por su fila (una letra de la 'A' a la 'J') y su columna (un número del 0 al 9). Para la práctica, el tamaño del tablero es el valor de la constante *DIMENSION* dado en la interfaz.

Inicialmente los jugadores colocarán un mismo número de barcos de ciertos tamaños (número de celdas que ocupa a lo largo, la anchura siempre será 1; es decir, el barco se ubica en una misma fila o columna de la tabla) sin que éstos se toquen. Para la práctica, el número y tamaño de los barcos a colocar se establece en la lista constante *BARCOS* declarada en la interfaz.

La partida se juega por turnos, el turno inicial normalmente se establece por acuerdo entre ambos jugadores. Para la práctica, el *turno inicial* corresponderá al jugador que, como consecuencia de lanzar el evento $iniciarJuego()$, lleva al servicio a poder realizar un emparejamiento de jugadores y que, por tanto, el juego entre ambos pueda comenzar.

El juego finaliza cuando uno de los contendientes se queda sin barcos (todos están hundidos), resultando ganador su oponente.

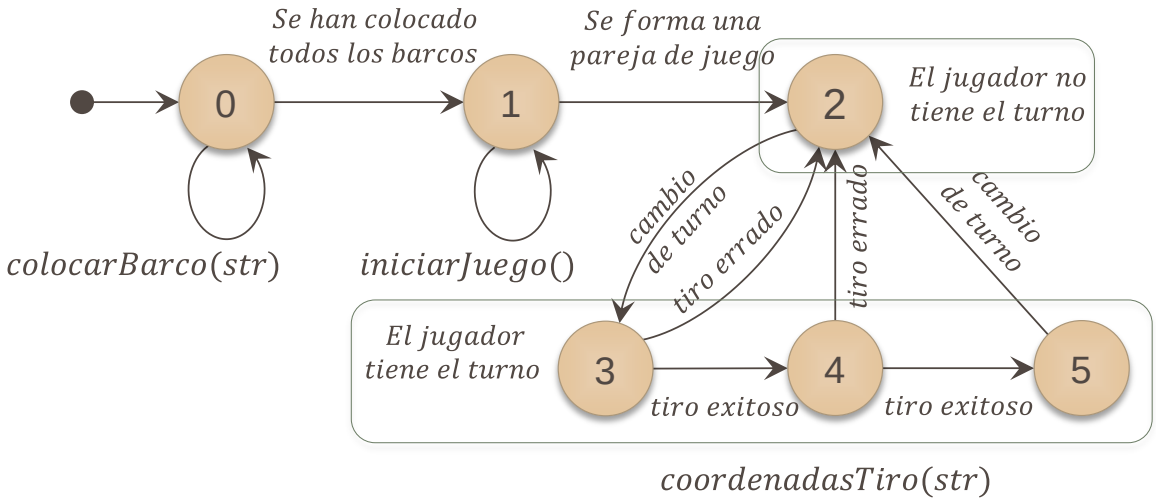
En lo que respecta al *cambio de turno de juego*, hay diversas variantes. Para la práctica, el jugador que tiene el turno realizará un tiro al océano de su oponente, si falla (*agua*) *cambia el turno* (pasa a su oponente), pero si da en un blanco (barco *tocado* o *hundido*) podrá realizar un nuevo tiro y así sucesivamente hasta un máximo de tres tiros consecutivos, produciéndose entonces un *cambio de turno* con independencia del resultado del último tiro realizado. En cualquier caso, no se tendrán en cuenta los tiros que provoquen alguna excepción como, por ejemplo, proporcionar unas coordenadas no válidas.

Grafo de estados del OOS de un jugador

Cuando un jugador se conecta al servicio se crea el OOS correspondiente y éste entra al estado inicial (0), en este estado el jugador debe colocar todos sus barcos en su tablero océano. Una vez completado este proceso el OOS pasa al estado 1 y podrá dar comienzo el juego cuando haya un par de contrincantes.

Cuando un jugador ha colocado todos sus barcos, el cliente interrogará al servicio para saber si es posible comenzar el juego, mediante el evento $iniciarJuego()$. Si existe la posibilidad, el servicio le asignará un oponente pasando al estado 2 para comenzar la partida. Si no hubiera emparejamiento posible, el OOS quedaría en el estado 1 y el jugador a la espera de contrincante.

A continuación, se proporciona el grafo de estados. Obsérvese que las transiciones entre estados se producen por eventos internos del sistema (del juego)[1] y que, en general, éstos son simples condiciones que ha de cumplir el *estado de datos* (ciertas variables de éste). Por supuesto, esos eventos internos van a ser consecuencia de los eventos que puedan lanzar los clientes y, los más relevantes, se han indicado en el grafo de estados.



El evento $turno()$ es fundamental porque es el que establece si el jugador tiene o no el turno de juego. En el grafo de estados se establece cómo y cuándo se produce un *cambio de turno*. Por supuesto, al tratarse de un juego entre un par de contendientes, si un jugador no tiene el turno es porque le corresponde a su oponente.

Por último, el estado del OOS de ambos contrincantes al final de la partida deberá ser el valor de la constante *FINAL_JUEGO* declarado en la interfaz. Este estado lo alcanzara el jugador que tiene el turno cuando hunde todos los barcos de su oponente (desde cualquiera de los estados 3, 4 o 5), pero para su oponente ocurre sin disponer del turno (desde el estado 2).

---

[1] En un sistema distribuido el *cliente* no tiene control sobre el estado del sistema, lo tiene el servidor y, por lo general, es el servidor el que debería informar de alguna forma al cliente de lo que puede o no puede hacer en un momento dado, por ejemplo, avisar a un cliente de que le corresponde el turno de juego. Dado que en una comunicación petición-respuesta es el *cliente* el que inicia el intercambio de mensajes realizando una petición al servidor y no a la inversa, en bastantes ocasiones el cliente ha de realizar un *pooling* de peticiones al servidor para saber qué puede hacer, por ejemplo: "*¿tengo el turno?*" o *"¿tengo oponente y ya se puede jugar?"*