Aclaración sobre la exclusión mutua

En la solución que se ha dejado del Juego de Barcos se ha realizado la exclusión mutua de la información compartida por los hilos (por los OOSs correspondiente a cada jugador), optando por realizar ésta mediante regiones críticas.

Aunque la solución propuesta requiere de bastante información compartida, las secciones críticas que es necesario excluir no son tantas. Para determinar correctamente éstas, es conveniente tener en cuentas las distintas fases de ejecución un hilo:

1. Desde el momento que se inicia la ejecución del hilo hasta el momento en que el juego puede dar comienzo.
2. Durante el trascurso del juego
3. Cuando el juego finaliza

Antes de comenzar el juego

En esta fase de la ejecución de los hilos, las operaciones que se pueden invocar y modifican información compartida son: $Servicio(idClient)$, $colocarBarco(str)$ e $iniciarJuego()$.

- En $Servicio(idClient)$ es necesario sincronizar los tres diccionarios que se modifican: $oceanoJugadores$, $turnoJugador$ y $barcosEnOceano$, porque es una modificación estructural (se añade una nueva clave) y $HasMap<K,V>$ no es una estructura sincronizada.
- En $colocarBarco(str)$ no hay sección crítica porque, aunque se accede al diccionario $barcosEnOceano$, no hay modificación estructural (la clave ya está creada) y, además, la clave es el identificador del único hilo que actualizará su valor asociado.
- En $iniciarJuego()$, además de accederse a los diccionarios $oponente$, $mutexTurno$ (en el apartado siguiente se verá para que se utiliza este diccionario) y $turnoJugador$, se accede a la lista compartida $jugadoresEnEspera$ que también es una estructura no sincronizada. En este caso en particular, como en la mayor parte del código está involucrada la lista, se ha optado por sincronizar prácticamente todo él mediante el objeto compartido $mutex$ (también se podría haber utilizado la lista). Por supuesto, es posible hacer una exclusión algo menos grosera, pero dado que es una operación que no se invoca en el transcurso del juego, se ha optado por esta solución más simple.

Durante el juego

Al tratarse de un juego por turnos (entre un par de oponentes) sólo habrá que preocuparse de realizar la exclusión mutua de la información compartida relativa al turno de juego: el diccionario $turnoJugador$. Esto es así porque sólo el jugador que tiene el turno va a jugar y actualizar la información compartida. El resto de los jugadores (para nuestro juego, el oponente) únicamente podrán consultar $turnoJugador$ para saber cuándo les corresponde el turno de juego. Esto será así, si tal y como debe ocurrir, en la implementación de las operaciones que dan lugar a un cambio

de turno, el código relativo a éste (los cambios en $turnoJugador$) es siempre lo último que se ejecuta. Obsérvese que, en caso contrario, habrá ciertos momentos en que el turno corresponderá a más de un jugador.

Para realizar la exclusión mutua de $turnoJugador$ la sincronización debe realizarse con un objeto compartido distinto para cada par de oponentes, de no ser así se producirían esperas innecesarias entre parejas de juego que, en teoría, deberían estar jugando de forma totalmente independiente. Por tanto, aquí sería un error utilizar el propio diccionario ($turnoJugador$) para sincronizar la región crítica, tal y como se hace en el constructor del servicio.

El diccionario compartido $mutexTurno$, que al igual que el resto de los diccionarios tiene como clave el identificador del cliente, es la estructura que va a contener el objeto compartido por cada par de oponentes y que se utilizará para sincronizar los cambios de turno. Como este objeto ha de ser el mismo para cada pareja de oponentes, podrá crearse cuando se determine la pareja de juego; es decir, en la operación $iniciarJuego()$ y, por esta razón, hacia el final el código de esta operación se incluyen las siguientes líneas:

```
// objeto de exclusión mutua para el turno de ambos oponentes
mutexTurno.put(this.idClient, new Object());
mutexTurno.put(idOponente, mutexTurno.get(this.idClient));
```

Las operaciones que pueden llevar a un cambio de turno son: $turno()$ y $coordenadasTiro(str)$, esta última al utilizar la operación interna $actualizarEstado(b)$. Obsérvese que, en ambos casos, la exclusión de la sección crítica correspondiente a la actualización de $turnoJugador$, se realiza con el objeto creado: $mutexTurno.get(idClient)$.

Al final del juego

Cuando finaliza el juego, únicamente se invocan las operaciones: $numBarcosEnOceano()$ y $close()$. La primera para determinar el ganador del juego (el jugador que tiene barcos no hundidos en su océano) y la segunda por la desconexión del cliente (para cerrar el OOS convenientemente).

- En $numBarcosEnOceano()$ no hay sección crítica. Se accede al diccionario $barcosEnOceano$, pero para lectura del valor asociado a la clave. En cualquier caso, tampoco la habría si fuera para escritura porque cada hilo accedería con una clave distinta (el identificador del cliente) y no habría modificación estructural porque las claves ya están en el diccionario.
- En $close()$, por el contrario, cada operación de borrado de la información compartida es una sección crítica (es una modificación estructural). Además, en caso de que el oponente del jugador que sale del juego siga conectado, se le da el turno a éste y al modificar $turnoJugador$ hay que realizar la exclusión mutua como se ha indicado en el apartado previo. Esto garantiza que el oponente también pueda finalizar el juego normalmente, evitando que por alguna circunstancia quede indefinidamente a la espera de su turno de juego.