Pruebas automáticas sobre el juego de Tres en Raya multijugador con tecnología Spring

Autores: Alberto Álvarez García (alberto.alvarez.garcia@alumnos.upm.es)

Mª Esther Terrero Díaz-Chirón (esther.terrero.diaz-chiron@alumnos.upm.es)

Introducción

Esta memoria cubre el análisis de las pruebas automáticas realizadas sobre una aplicación real consistente en el juego de las Tres en Raya (TicTacToe) en formato multijugador.

La aplicación está implementada en Java con la tecnología Spring usando WebSockets.

El juego está implementado con las siguientes clases en la parte servidor:

- **Board**: Representa el tablero del juego. Tiene métodos para determinar si un jugador tiene tres en raya o si el tablero está completo y ninguno tiene tres en raya (empate).
- **Player**: Clase que mantiene el nombre del jugador, el tipo de ficha (label) y su identificador.
- **TicTacToeGame**: Es la clase que gestiona el ciclo de vida del juego gestionando el tablero, el turno del jugador, etc. Los métodos de esta clase serán invocados cada vez que se reciba un mensaje de cualquiera de las conexiones con los navegadores web. Cada vez que haya un cambio en el juego, desde esta clase se envían eventos a los navegadores web usando los objetos Connection.
- **Connection**: Representa la conexión con un navegador web mediante WebSockets.
- **TicTacToeHandler**: Es la clase encargada de procesar los mensajes que llegan de los navegadores web e invocar los métodos de la clase TicTacToeGame.

Se piden las siguiente automáticas:

- Pruebas unitarias de la clase Board
- Pruebas con dobles de la clase TicTacToe
- Pruebas de sistema de la aplicación

En todos los tipos de prueba hay que comprobar la misma funcionalidad, pero el test se implementa a diferentes niveles (unitario, con dobles y de sistema). En concreto hay que implementar al menos los siguientes tests por cada tipo:

- El primer jugador que pone ficha gana.
- El primer jugador que pone ficha pierde.
- Ninguno de los jugadores gana. Hay empate.

En total tienen que implementarse 9 tests (3 de cada tipo representando cada uno de los escenarios anteriores). Como los tests del mismo tipo van a ser muy parecidos entre sí, se pide que se estructure el código de tal forma que se reutilice y no haya duplicaciones.

Las distintas pruebas se han planteado entonces de acuerdo a las siguientes premisas:

1. El sistema siempre asigna la ficha ‘X’ al primer jugador que se une a la partida y la ficha ‘O’ al segundo jugador. En las pruebas no se diferencia entonces quien es el primer jugador que ha puesto ficha, que es siempre el jugador con la ficha ‘X’, sino cuál de los dos jugadores gana la partida o si hay empate.
2. Para facilitar el entendimiento de los distintos casos de prueba se ha creado una clase llamada ‘CellId’ que es un enumerado de las posiciones del tablero, de forma que se vea de forma fácil la fila de la celda (indicada por ‘TOP_’, ‘MIDDLE_’ y ‘BOTTOM’) y la columna de la celda (indicada por ‘_LEFT’, ‘_CENTER’ y

'_RIGHT'). Asimismo, contiene un método que traduce un array de enumerados a su correspondiente array de enteros.

3. Se ha realizado una parametrización de los casos de prueba de forma que se faciliten dos cosas:
    a. El uso de distintos parámetros a utilizar en distintos casos de prueba
    b. El poder diferenciar en el título del caso de prueba cual es la prueba concreta que se está realizando. Hay parámetros que indican por ejemplo si la prueba se está haciendo con el tablero lleno o no, algo que realmente no es una entrada para la prueba, pero permite diferenciar que es lo que se está probando.
4. En los distintos tipos de prueba se han planteado cuatro escenarios, no tres:
    1. El jugador con la ficha 'X' (el que empieza), es el que gana con el tablero sin estar completo
    2. El jugador con la ficha 'X' (el que empieza), es el que pierde con el tablero sin estar completo
    3. Ninguno de los dos jugadores gana (y por lo tanto el tablero está completo y hay empate).
    4. El jugador con la ficha 'X' (el que empieza), es el que gana en la última jugada (y por lo tanto completa con el tablero).

Pruebas unitarias de la clase Board

En estas pruebas se pide comprobar que la clase implementa correctamente la detección de que un jugador ha ganado o de que se ha empatado. Concretamente se pide probar los métodos getCellsIfWinner(…) y checkDraw() funcionan como se espera:

- El método getCellsIfWinner(…) debe devolver el número de celdas que contienen la "línea" en caso de que el jugador pasado como parámetro haya ganado y si no, devolver null.
- El método checkDraw() debe devolver true si el tablero está completo sin ninguna línea con fichas iguales.

Los parámetros de input contienen la siguiente información:

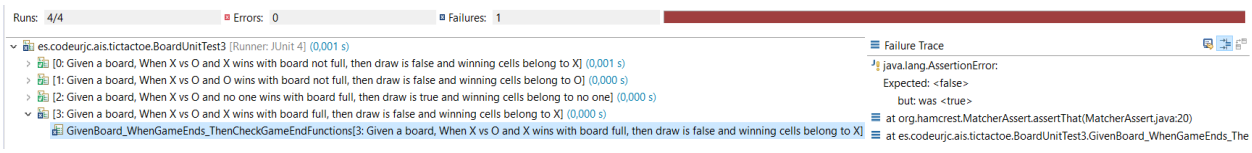
1. El token que usa el jugador que empieza.
2. El token que usa el segundo jugador.
3. Qué jugador gana la partida.
4. Si al final de la partida el tablero está completo o no.
5. Si se espera que la salida del método que calcula si hay empate sea 'true' o 'false'.
6. Las jugadas que introduce el primer jugador.
7. Las jugadas que introduce el segundo jugador.
8. La posición de celdas ganadora (si hay alguna) o null en caso de que haya empate.

Las pruebas se han hecho con JUnit4. Se consideró hacer las pruebas con JUnit5 para hacer casos de prueba anidados pero dado que se han parametrizado los casos de prueba esto añadía una complejidad que no se estimó que no aportaba realmente nada, descartando por tanto el uso de JUnit5.

Para comprobar el correcto funcionamiento de los métodos se ha hecho uso de 'assertThat' comprobando:

1. Que el método getCellsIfWinner() aplicado al jugador que no ha ganado devuelve null.
2. Que el método getCellsIfWinner() aplicado al jugador ha ganado devuelve la secuencia que se ha definido en los parámetros de entrada como secuencia ganadora.
3. Que el método checkDraw() se corresponde con lo que dice el parámetro de entrada (y depende de si hay ganador o no).

Lo que se ha visto es que el método 'getCellsIfWinner' funciona correctamente pero el método 'checkDraw()' solo comprueba si el tablero está lleno o no. Hay un error ya que cuando el tablero está lleno pero la jugada ganadora ha sido la última, el método devuelve que ha habido empate, lo que no es correcto. El incorrecto funcionamiento de este método lo podemos ver en la siguiente captura, donde el jugador A gana la partida en la última jugada y sin embargo el método devuelve que hay empate.



Pruebas con dobles de la clase TicTacToeGame

En estas pruebas se pide comprobar que la clase TicTacToeGame implementa de forma adecuada el juego. Para ello, durante las pruebas, se llama a los métodos de la clase simulando los mensajes que llegan de los navegadores durante el uso normal de la aplicación. Se crean dobles de la clase Connection (un doble por cada conexión/jugador), que se utilizan para verificar si los eventos que envía TicTacToeGame a los navegadores web (representados por los dobles de Connection) son los esperados.

Al igual que en los casos de prueba unitarios estas pruebas también se han parametrizado, pero simplificando los parámetros de entrada, que contienen la siguiente información:

1. Qué jugador gana la partida (si el primero o el segundo).
2. Las jugadas que introduce el primer jugador.
3. Las jugadas que introduce el segundo jugador.
4. La posición de celdas ganadora (si hay alguna) o null en caso de que haya empate.

Para cada caso de prueba el desarrollo es:

1. Se crea un objeto de la clase TicTacToeGame y se le añaden las dos conexiones mockeadas.
2. Se crean dos usuarios, y se añaden al juego. Tras esto se verifica que por cada conexión se ha mandado el evento EventType.JOIN_GAME con los dos jugadores que están en la partida y se resetean las conexiones con el método 'reset()' para borrar el registro de llamadas a los métodos de los mocks correspondientes.
3. Se simula la alternancia de las jugadas dentro del método 'play()' definido en el caso de prueba, de forma que se van cogiendo de forma alterna de los parámetros de entrada una jugada de cada jugador y marcando en el tablero. Como resultado se comprueba que por las dos conexiones se notifica que se manda un evento indicando el marcado de la jugada, EventType.MARK. Además, la alternancia de los turnos se comprueba verificando que por cada conexión se recibe un evento de tipo SET_TURN, que indica el jugador contrario al último que realizó la jugada, usando para ello la clase ArgumentCaptor que permite ver el jugador que se pasó como parámetro a la conexión mockeada.
4. Tras cada jugada se comprueba si ha habido una jugada ganadora, y si la ha habido, se comprueba si por las dos conexiones se manda el evento de fin de juego, EventType.GAME_OVER, indicando el jugador ganador (utilizando también la clase ArgumentCaptor), si lo ha habido o empate en caso contrario.
5. Para finalizar el caso de prueba, en caso de que haya habido un ganador se comprueba que la jugada ganadora es la especificada en los parámetros de entrada.

Nota: En estas pruebas no falla ningún caso de prueba porque el método checkDraw() solo se comprueba en caso de que el jugador que acaba de introducir la jugada no haya conseguido una combinación ganadora, con lo que el resultado en conjunto es correcto, aunque la salida de checkDraw() si se comprobase aisladamente no fuese realmente correcta.

Pruebas de sistema de la aplicación

Las pruebas de sistema verifican que la aplicación completa funciona correctamente. Como el juego está implementado de forma que al finalizar el mismo el resultado aparece en un cuadro de diálogo (alert). El objetivo de los tests consiste en verificar que el mensaje del alert es el esperado cuando gana cada uno de los jugadores y cuando quedan empate.

Para realizar las pruebas de sistema se utiliza Selenium, de forma que para simular una partida el test iniciará dos navegadores web de forma simultánea e irá interactuando con ellos de forma alternativa. De esta forma, puede simular un partida por turnos.

Al igual que en las otras pruebas también los casos de prueba de sistema se han parametrizado, de forma que los parámetros de entradas son:

1. El identificador del jugador que empieza (el primero registrado)
2. El identificador del segundo jugador.
3. Qué jugador gana la partida.
4. Las jugadas que introduce el primer jugador.
5. Las jugadas que introduce el segundo jugador.

En este caso no se comprueba la jugada ganadora ya que no es el objetivo de la prueba, sino comprobar el mensaje de alert.

Se ha tenido que modificar el POM.xml del proyecto para cambiar la versión del 'springframework.boot' a la versión 2.1.3.RELEASE, ya que si no daba error en la compilación.

El desarrollo del caso de prueba es el siguiente:

1. Como test fixture antes de cualquier prueba de la clase, se hace el setup de los dos WebDriverManagers, uno para cada jugador. Se ha optado por probar con un driver de Chrome y otro de Firefox. A continuación, se lanza la aplicación.
2. Por cada caso a prueba a ejecutar se define como test fixture, la creación de los browsers a usar (uno Chrome y otro Firefox).
3. Dentro de cada caso de prueba se simula en el método 'startGame()', el login de cada usuario en su browser.
4. Con el método 'play()' se simula la alternancia de jugadas entre los dos jugadores, pasándosele la jugada a aplicar al método 'click' así como el browser donde se tiene que aplicar esta jugada.
5. Cuando el juego ha terminado, dependiendo de cual haya sido el jugador que se haya definido como ganador en los parámetros de entrada se construye el mensaje de alert esperado.
6. Finalmente se comprueba con un AssertEquals que el mensaje esperado y el obtenido coinciden. Para conseguir el mensaje obtenido se hace uso del método 'browser<X>.switchTo().alert().getText()', donde <X> es 1 o 2.
7. Como test fixture después de cada caso de prueba se cierran los browsers.
8. Finalmente, como test fixtura al final de la clase de prueba se para la aplicación.