Máster Software Craftmanship

Práctica 2: Implementación de pruebas automáticas

[Módulo II] EACS. Pruebas del Software

Autores:

Pablo Jaramillo
pablo.jaramillo@ericsson.com

María Huertas Martín
maria.huertas.martin@ericsson.com

Introducción

Se han implementado 10 tests en total, 3 tests por cada tipo: unitario, usando dobles y de sistema, y un unit test parametrizado que engloba todos los escenarios.

En cada uno de los tipos de test se comprueba que:

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

La memoria se ha estructurado en tres secciones, una por cada nivel de test elaborado.

Pruebas unitarias de la clase Board

Para las pruebas unitarias se ha utilizado Junit 4, y se han implementado 3 test en los que se prueban los métodos getCellsIfWinner() y checkDraw(), simulando partidas entre dos jugadores.

Para las pruebas unitarias se han entregado dos ficheros:

- BoardTest.java

  Se prueba el funcionamiento del método getCellsIfWinner() introduciendo una combinación ganadora en un objeto de la clase Board y llamando al método a continuación. Se comprobará que, para el jugador perdedor, el array devuelto es nulo y que, para el ganador, el array devuelto no es nulo ya que contendrá las posiciones de las celdas de la línea ganadora.

  El método checkDraw() se prueba simulando una partida empatada, llenando el tablero sin que haya ninguna línea ganadora. A continuación se llama a checkDraw() y se comprueba que su valor de retorno sea true, lo que indica que hay empate entre ambos jugadores.

- BoardParameterizedTest.java

  Al tratarse de test donde se prueban las diferentes combinaciones de posiciones de celda elegidas por los jugadores, resulta útil parametrizar dichos valores. De este modo, se podrían probar diferentes posiciones en un único test, sin repetir código.

  De este modo, se han parametrizado las posiciones para que una vez resulte ganador el jugador X, en otra ocasión Y, y finalmente, empate. En todos los casos se estarían probando también los métodos previamente mencionados, pero rompiendo en cierto modo los requisitos de los 3 escenarios con un test para cada uno de ellos.

Pruebas con dobles de la clase TicTacToeGame

En estas pruebas se ha utilizado JUnit4 y Mockito para probar la clase TicTacToeGame con dobles de los objetos Connection. Estos dobles de los objetos se utilizarán para comprobar si los eventos que envía TicTacToeGame son los correctos en función del momento en el que se encuentre la partida. Se han implementado tres test, como se requiere en la práctica, creando dos dobles de los objetos Connection y comprobando no solo el evento que se manda sino también el argumento de tipo Object, que cambia con cada tipo de evento. Se ha utilizado la clase ArgumentCaptor (incluida en Mockito), la cual permite capturar los argumentos con los que se llama a un determinado método para luego realizar comprobaciones sobre ellos. Del mismo modo se ha hecho uso del método reset(mock), el cual permite resetear el conteo de llamadas a métodos del mock, proporcionando mucha más flexibilidad a la hora de escribir los test. Los test siguen el siguiente esquema de llamadas y comprobaciones:

- Se crean objetos de la clase TIcTacToeGame y de la clase Player.

```
game = new TicTacToeGame();

player0 = new Player(0, "X", "Pikachu");
player1 = new Player(1, "O", "Charmander");
```

- Se añade un jugador a la partida y se comprueba el evento y la lista en los argumentos. A continuación, se añade el segundo jugador y se repite la operación.

```
game.addPlayer(player0);

verify(connection1, times(1)).sendEvent(eq(EventType.JOIN_GAME), argumentCaptor.capture());
verify(connection2, times(1)).sendEvent(eq(EventType.JOIN_GAME), argumentCaptor.capture());

List<Player> values = argumentCaptor.getValue();
assertEquals(values, list);
```

- Se llama repetidamente al método mark() para simular una partida y se comprueba el número de eventos SET_TURN así como el evento GAME_OVER y el jugador ganador enviados a los objetos Connection.

```
game.mark(0);
game.mark(3);
game.mark(1);
game.mark(5);
game.mark(2);

verify(connection1, times(4)).sendEvent(eq(EventType.SET_TURN), argumentCaptor.capture());
verify(connection2, times(5)).sendEvent(eq(EventType.SET_TURN), argumentCaptor.capture());

verify(connection2, times(1)).sendEvent(eq(EventType.GAME_OVER), winnerArgumentCaptor.capture());
verify(connection2, times(1)).sendEvent(eq(EventType.GAME_OVER), winnerArgumentCaptor.capture());

WinnerValue capturedWinner = winnerArgumentCaptor.getValue();
```

Pruebas de sistema de la aplicación

Para las pruebas de sistema se ha hecho uso de Selenium WebDriver, ya que nos permite un control automatizado de navegadores web locales. El navegador web utilizado en los test es Google Chrome.

Para hacer uso de Selenium WebDriver, en primer lugar, se necesita importar en el pom.xml la dependencia externa necesaria:

```
<dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>3.141.59</version>
        <scope>test</scope>
</dependency>
```

```
<dependency>
        <groupId>io.github.bonigarcia</groupId>
        <artifactId>webdrivermanager</artifactId>
        <version>3.3.0</version>
        <scope>test</scope>
</dependency>
```

Para comenzar, se debe ejecutar la aplicación en dos navegadores, para conocer así cuáles son las zonas con las que el usuario interaccionará. Después, se deberán inspeccionar los elementos de la página para conocer el lugar exacto donde el test debe de hacer click, o introducir datos. De esta forma, Selenium podrá simular una partida real.

Para la escritura de este tipo de tests, donde se necesitan dos usuarios jugando, por lo que es importante crear dos WebDriver, uno por cada jugador. Según se indica en el enunciado de la práctica, ambos usuarios deben estar apuntando a localhost:8080.

```
public void GIVEN_twoPlayers_WHEN_firstOneStarts_THEN_Win() {
    driver0.get("http://localhost:8080/");
    driverX.get("http://localhost:8080/");
```

Antes de empezar la partida, el juego permite a cada uno de los usuarios introducir su nombre de usuario, y hacer click en “PLAY!”.

```
driver0.findElement(By.id("nickname")).sendKeys("Pablo");
driver0.findElement(By.id("startBtn")).click();
driverX.findElement(By.id("nickname")).sendKeys("Maria");
driverX.findElement(By.id("startBtn")).click();
```

Cuando ambos jugadores han hecho esto, la partida puede dar comienzo, y podrían empezar a hacer click en las celdas del juego donde quieran colocar la ficha. En este caso, el jugador mete la combinación “0, 1, 2”, que hace referencia a la primera fila del tablero. De esta forma nos aseguramos de que sea él quien ganará.

```
driver0.findElement(By.id("cell-0")).click();
driverX.findElement(By.id("cell-3")).click();
driver0.findElement(By.id("cell-1")).click();
driverX.findElement(By.id("cell-5")).click();
driver0.findElement(By.id("cell-2")).click();
```

Una vez uno de los jugadores gana, aparece un mensaje en la pantalla de cada uno de ellos, impidiendo el siguiente movimiento. En la pantalla del jugador que gana, aparece el mensaje mostrado en el test. Para acabar, se deberá verificar que la web bajo pruebas cumple las condiciones esperadas comparando el texto proporcionado con el que le aparece al jugador en tiempo de ejecución.

```
String body = driverX.switchTo().alert().getText();
String newBody = "Pablo wins! Maria looses.";
Assert.assertEquals(body, newBody);
```

Para el resto de test se ha procedido de la misma forma, cambiando las celdas de los jugadores para así poder comprobar el resto de los escenarios necesarios.