Máster en Software Craftsmanship

[Módulo II] EACS. Pruebas del Software

Práctica 2

Implementación de pruebas automáticas

1. Pruebas unitarias de la clase Board del juego TicTacToe

Se tienen que implementar varios tests que cambien el estado de las celdas simulando una partida y posteriormente verificar que los métodos getCellsIfWinner(…) y checkDraw() funcionan como se espera. El método getCellsIfWinner(…) devuelve el número de celdas que contienen la "línea" en caso de que el jugador pasado como parámetro haya ganado. Si no, devuelve null. El método checkDraw() devuelve true si el tablero está completo sin ninguna línea con fichas iguales.

Tests propuestos:

- **`BoardTestParamWhenWinner.java:`** Incluye 2 TCs parametrizados que prueban que los resultados de *getCellsIfWinner* y *checkDraw* son los esperados cuando hay un ganador:
  - `givenAPlayerWinsWhenGetCellsIfWinnerThenWinnerPositionIsObtainedTest`
  - `givenAPlayerWinsWhenCheckDrawThenFalseIsObtainedTest`
  - La lógica de los tests y la definición de los parámetros se definen en BoardTest.java.
  - En esta clase están solamente los valores de los parámetros con los que se desean ejecutar los tests. Se ejecutan los dos TCs (*getCellsIfWinner* y *checkDraw*) 16 veces (32 TCs):
    - 8 combinaciones de tablero con cada una de las 8 combinaciones ganadoras (para alguno de los dos tipos de ficha, X o O) sin llegar a llenar el tablero => 8 x 2 = 16 TCs
    - 8 combinaciones de tablero con cada una de las 8 combinaciones ganadoras (para alguno de los dos tipos de ficha, X o O) llenando el tablero. => 8 x 2 = 16 TCs
      NOTA: Ver sección "Resultados" explicando que para estas 8 combinaciones los TCs para la función *checkDraw* fallan por un bug en el código.

- **`BoardTestParamsWhenNoWinner.java:`** Incluye 2 TCs parametrizados que prueban que los resultados de *getCellsIfWinner* y *checkDraw* son los esperados cuando no hay un ganador (draw).
  - `givenNoPlayerWinsWhenGetCellsIfWinnerThenWinnerPositionIsNullTest`
  - `givenNoPlayerWinsWhenCheckDrawThenTrueIsObtainedTest`
  - La lógica de los tests y la definición de los parámetros se definen en BoardTest.java.
  - En esta clase están solamente los valores de los parámetros con los que se desean ejecutar los tests. Se ejecutan los dos TCs (*getCellsIfWinner* y *checkDraw*) 2 veces (4 TCs):
    - 2 combinaciones de tablero sin combinación ganadora (draw), una para cada tipo de ficha, X o O.

- **`BoardTest.java:`** Clase con la lógica común de los tests y la definición de los parámetros:
  - `Label {0}` -> tipo de ficha para la que se desea comprobar si existe ganador, X o O.
  - `boardCells {1}` -> fichas existentes en el tablero cuando se hace la comprobación llamando a las funciones *getCellsIfWinner* y *checkDraw*.
  - `expectedWinPos {2}` -> resultado que se espera recibir de la función *getCellsIfWinner*: las celdas que componen la línea ganadora o null si se espera empate.
  - `expectedCheckDraw {3}` -> resultado que se espera recibir de la función *checkDraw*: true si hay empate y false si no lo hay.
  - La lógica para los TCs de las clases anteriores es común:
    - `givenACertainBoardWhenGetCellsIfWinnerThenTheWinnerPositionOrNullIsObtainedTest`
      - **Given**: se crea un tablero y se rellena con las fichas recibidas en el parámetro `boardCells`.
      - **When**: se llama a la función para el tipo de ficha recibida en `label` y se guarda el resultado.
      - **Then**: se compara el resultado guardado con el esperado en el parámetro `expectedWinPos.`
    - `givenACertainBoardWhenCheckDrawThenCorrectBooleanIsObtainedTest`
      - **Given**: se crea un tablero y se rellena con las fichas recibidas en el parámetro `boardCells`.

- **When**: se llama a la función y se guarda el resultado.
- **Then**: se compara el resultado guardado con el esperado en el paramétro `expectedCheckDraw`.

- **NOTA:** Se podría haber hecho una única clase de test **`BoardTest.java`** con todas las combinaciones a probar (las 16 de la primera clase y las 2 de la segunda). Se ha dividido en dos clases reutilizando código con herencia para asignar nombres significativos a los TCs:
  - Cuando existe ganador el TC tiene título: `"Player {0} wins at {2} with checkDraw = {3}"`
  - Cuando no existe ganador el TC tiene título: `"No winner with checkDraw = {3}"`

Resultados:

Los TCs `givenAPlayerWinsWhenCheckDrawThenFalseIsObtainedTest` en los que hay ganador, pero la línea se consigue al llenar el tablero, **fallan**, ya que esperan que *checkDraw* devuelva false (no hay empate) pero se recibe true. Esto revela un **bug** en el código fuente: la función *checkDraw* devuelve en realidad true siempre que el tablero está lleno, haya empate o haya línea. Una **solución** sería llamar a la función *checkFull*, y esperar true en los tests siempre que se llene el tablero, ya que la lógica para detectar al ganador funciona correctamente como se ve en los siguientes tipos de tests.

2. Pruebas con dobles de la clase TicTacToeGame

En estas pruebas se tiene que comprobar que la clase TicTacToeGame implementa de forma adecuada el juego. Se tendrá que llamar a los métodos de la clase simulando los mensajes que llegan de los navegadores durante el uso normal de la aplicación. Además, se deberán simular las conexiones creando dobles de los objetos Connection. Estas conexiones simuladas se utilizarán para verificar si los eventos que envía TicTacToeGame a los navegadores web son los esperados.

Test propuesto:

- **`TicTacToeGameTest.java`:** Incluye 1 TC parametrizado que prueba que los eventos que envía TicTacToeGame a los navegadores son los esperados cuando hay ganador o en caso de empate.
  - Se definen los siguientes parámetros:
    - `labelJ1 {0}` -> tipo de ficha del jugador 1, X o O.
    - `labelJ2 {1}` -> tipo de ficha del jugador 2, X o O.
    - `playersMovements {2}` -> celdas que irán marcando los jugadores 1 y 2 de manera alternativa
    - `thereIsAnyWinner {3}` -> booleano que indica si se espera ganador con los movimientos de `{2}`
    - `labelWinner {4}` -> tipo de ficha, X o O, que gana la partida.
  - Se ejecuta el TC 3 veces (3 TCs):
    - Realizando movimientos en los que gana el jugador 2 (con ficha O) sin llegar a llenar el tablero.
    - Realizando movimientos en los que gana el jugador 1 (con ficha X) llegando a llenar el tablero.
    - Realizando movimientos en los que hay empate, se llena el tablero sin línea ganadora.
  - La lógica para el TCs es:
    - **Given**: se crea un juego, se añaden dos conexiones mockeadas, y se crean dos jugadores.
    - **When**: se añade el jugador 1 con etiqueta `labelJ1`.
    - **Then**: se chequea que se recibe el evento ***`JOIN_GAME`*** con el jugador1.
    - **When**: se añade el jugador 2 con etiqueta `labelJ2`.
    - **Then**: se chequea que se reciben los eventos ***`JOIN_GAME`*** con el jugador 2 y ***`SET_TURN`*** con el jugador 1.
    - **En un bucle se van marcando las casillas recibidas en el parámetro `playersMovements` (todas menos la última casilla, en la que se decidirá si hay ganador o empate):**

        - **When**: se marca la casilla indicada para el jugador correspondiente (van alternando)
        - **Then**: se chequea que se reciben los eventos ***MARK*** y ***SET_TURN*** con los parámetros correctos (van alternando).
    - **When**: se marca última casilla recibida en el parámetro `playersMovements`.
    - **Then**: el parámetro `thereIsAnyWinner` marca los eventos a chequear:
        - Si es true, chequea que se recibe el evento ***GAME_OVER*** con el jugador ganador y la línea ganadora.
        - Si es false, chequea que se recibe el evento ***GAME_OVER*** vacío.

Resultados:

Los 3 TCs pasan correctamente, se gane llenando el tablero (al marcar la última celda) o no.

3. Pruebas de sistema de la aplicación

Para simular una partida el test iniciará dos navegadores web manejados con Selenium de forma simultánea e irá interactuando con ellos de forma alternativa. De esta forma, puede simular una partida por turnos. 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.

- **`WebAppTest.java`:** Incluye 1 TC parametrizado que prueba que el mensaje de alert es el esperado cuando hay ganador o en caso de empate.
    - Se definen los siguientes parámetros:
        - `nameJ1 {0}` -> nombre del jugador 1.
        - `nameJ2 {1}` -> nombre del jugador 2.
        - `playersMovements {2}` -> celdas que irán marcando los jugadores 1 y 2 de manera alternativa.
        - `expectedFinalMessage {3}` -> mensaje de alert esperado con los movimientos de `{2}`.
    - Se ejecuta el TC 3 veces (3 TCs):
        - Realizando movimientos en los que gana el jugador 2 (con ficha O) sin llegar a llenar el tablero.
        - Realizando movimientos en los que gana el jugador 1 (con ficha X) llegando a llenar el tablero.
        - Realizando movimientos en los que hay empate, se llena el tablero sin línea ganadora.
    - La lógica para el TCs es:
        - **setupClass:** se configura el objeto WebDriver de Selenium para manejar los navegadores Chrome y se arranca la aplicación de TicTacToe.
        - **setupTest:** se crean los dos navegadores Chrome.
        - **Teardown:** se cierran los dos navegadores Chrome.
        - **teardownClass:** se para la aplicación de TicTacToe.
        - **Given**: se conectan los dos navegadores al juego, cada uno con un nombre de jugador según los identificadores `nameJ1` e `nameJ2` recibidos como parámetros.
        - **When**: en un bucle se van marcando las casillas recibidas en el parámetro `playersMovements` para el jugador correspondiente (van alternando).
        - **Then**: se chequea que se recibe el mensaje de alert esperado en `expectedFinalMessage`.

Resultados:

Los 3 TCs pasan correctamente, se gane llenando el tablero (al marcar la última celda) o no.