AMPLIACIÓN DE INGENIERIA DEL SOFTWARE

ENTREGA 2

Nombre: Ariana González Garzón

Grupo: GII

TEST 1 – PRUEBAS UNITARIAS DE LA CLASE BOARD

En primer lugar, para cada test inicializamos un tablero y los jugadores.

1- testJugadorUnoWinner(), testJugadorDosWinner()

Crearemos un array con una posición esperada en la que gane el jugador:

```
int[] posicion = {0, 1, 2};
```

Después de realizar los correspondientes movimientos comprobaremos si la posición inicial es igual a la obtenida para el jugador uno:

```
assertArrayEquals(posicion, tablero.getCellsIfWinner(jugadorUno.getLabel()));
```

Verificamos que el jugador dos no es el ganador:

```
assertNull(tablero.getCellsIfWinner(jugadorDos.getLabel()));
```

Finalmente comprobamos que no se ha producido un empate:

```
assertEquals(false, tablero.checkDraw());
```

Sería similar para el jugador dos.

2- testJuegoEmpatado()

Inicialmente lo que vamos a hacer es comprobar que no hay un empate de la partida ya que no hemos empezado a movernos.

```
assertFalse(tablero.checkDraw());
```

Después de realizar los correspondientes movimientos hasta lograr un empate comprobaremos que se da un empate en el tablero (no hay un ganador).

```
assertNull(tablero.getCellsIfWinner(jugadorUno.getLabel()));

assertNull(tablero.getCellsIfWinner(jugadorDos.getLabel()));

assertTrue(tablero.checkDraw());
```

TEST 2 – PRUEBAS DOBLES CON LA CLASE TICTACTOEGAME

Ayudándonos de los pasos del enunciado hemos realizado los siguientes pasos.

En primer lugar, inicializaremos la clase TicTacToeGame con game, los jugadores y las conexiones que se usaran.

A continuación, creamos los dobles de los objetos Connection:

```
conexionUno= mock(Connection.class);
conexionDos = mock(Connection.class);
```

Primero añadimos al jugador uno.

Verificamos que las conexiones reciben el evento JOIN_GAME. Lo verifican las dos conexiones.

```
verify(conexionUno).sendEvent(eq(EventType.JOIN_GAME), argThat(hasItems(jugadorUno)));
verify(conexionDos).sendEvent(eq(EventType.JOIN_GAME), argThat(hasItems(jugadorUno)));
```

Borramos el registro de llamadas a los métodos del mock con **reset**.

```
reset(conexionUno);
reset(conexionDos);
```

Despues añadimos al jugador dos y comprobamos que las conexiones reciben el evento JOIN_GAME.

```
verify(conexionUno).sendEvent(eq(EventType.JOIN_GAME), argThat(hasItems(jugadorUno, jugadorDos)));
verify(conexionDos).sendEvent(eq(EventType.JOIN_GAME), argThat(hasItems(jugadorUno, jugadorDos)));
```

Borramos el registro de llamadas a los métodos del mock con **reset**.

Establecemos el turno inicial que le corresponde al jugador Uno y empezamos a probar el juego:

```
verify(conexionUno).sendEvent(eq(EventType.SET_TURN), eq(jugadorUno));
verify(conexionDos).sendEvent(eq(EventType.SET_TURN), eq(jugadorUno));
```

testJugadorUnoWinner(), testJugadorDosWinner() y testJuegoEmpatado():

En primer lugar, al haber comprobado ya anteriormente que es el turno del jugador uno realizamos el movimiento con **mark**.

Lo que haremos durante este test es ir comprobando que los diferentes jugadores Uno y Dos reciben el evento SET_TURN y realizan un movimiento.

Según los movimientos que se realicen ganara el jugador uno, el jugador dos o se dará un empate.

Finalmente recuperamos el WinnerValue y verificamos (según el test) que jugador ha ganado y cual ha perdido.

La prueba desea verificar que el argumento pasado al método de sendEvent es un objeto WinnerValue donde para la prueba 1 será igual al jugador uno, prueba 2 jugador dos y prueba 3 no seria ninguno.

En el caso del jugador uno ganador seria:

```
ArgumentCaptor<WinnerValue> argument = ArgumentCaptor.forClass(WinnerValue.class);
verify(conexionUno).sendEvent(eq(EventType.GAME_OVER), argument.capture());
WinnerValue event = (WinnerValue) argument.getValue();

// Comprobamos que la conexion 2 recibe el GAME_OVER
verify(conexionDos).sendEvent(eq(EventType.GAME_OVER), eq(event));

// Comprobamos quien ha ganado y quien ha perdido

assertThat(event.player.equals(jugadorUno));
assertThat(!event.player.equals(jugadorDos));
assertNotNull(event);
```

En el caso de empate seria:

```
ArgumentCaptor<WinnerValue> argument = ArgumentCaptor.forClass(WinnerValue.class);
verify(conexionUno).sendEvent(eq(EventType.GAME_OVER), argument.capture());
Object event = (WinnerValue) argument.getValue();

// Comprobamos que la conexion 2 recibe el GAME_OVER

verify(conexionDos).sendEvent(eq(EventType.GAME_OVER), eq(event));

assertNull(event);
```

TEST 3 – PRUEBAS DE SISTEMA DE LA APLICACIÓN

En primer lugar, ejecutamos los drivers del navegador Firefox.

```
protected WebDriver driver1, driver2;

@BeforeClass
public static void setupClass() {
    WebDriverManager.firefoxdriver().setup();
    WebApp.start();
}

@AfterClass
public static void teardownClass() {
    WebApp.stop();
}
```

Iniciamos el jugador uno. Usamos el método sendKeys para darle un “nickname2 al jugador. Realizamos lo mismo con el jugador dos.

Haremos click en ambos para iniciar la partida.

A continuación, para cada test se realizaran movimientos que llevan a una victoria, derrota o empate por parte de los jugadores. Es decir, localizamos los elementos con findElement e interactuamos con ellos (click).

La aserción será exitosa si el mensaje recibido coincide con el esperado.

```
assertEquals(driver1.switchTo().alert().getText(), "JugadorUno wins! JugadorDos looses.");
assertEquals(driver2.switchTo().alert().getText(), "JugadorUno wins! JugadorDos looses.");
```

Al finalizar cada test se cierra el browser con el método quit().

USO DE JENKINS EN LAS PRUEBAS:

Despues de configurar Jenkins tal cual viene en el manual de la asignatura del profesor hemos procedido a la creación de tareas con el fin de comprobar el correcto funcionamiento de los test creados.

Jenkins en primer lugar descarga un repositorio git, compila el proyecto y ejecuta los test sobre el proyecto.