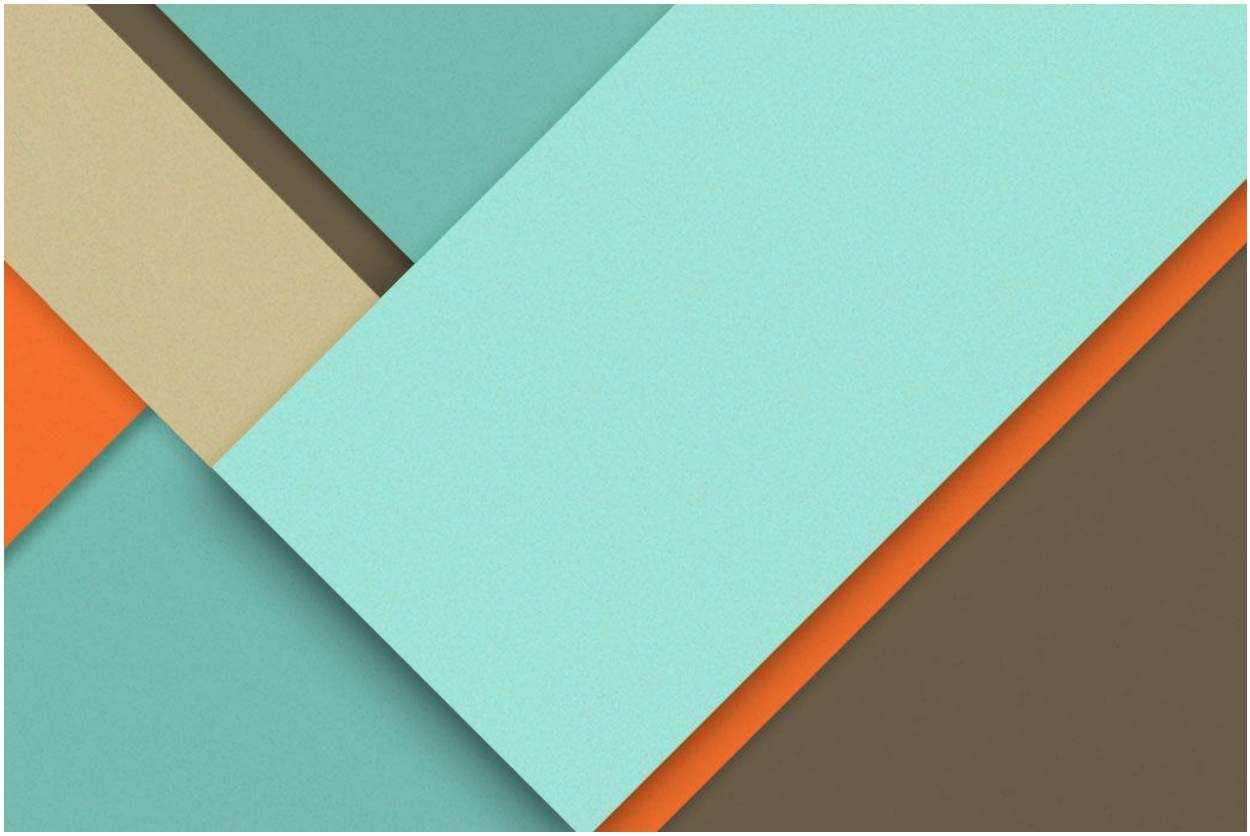

Pruebas automáticas de SW

24/05/2018

Javier Redondo Antón

Guillem García Subies

Ampliación de Ingeniería del Software

Tests unitarios

Estos tests han sido los más fáciles de realizar, aunque al principio tuvimos bastantes problemas con los atributos protegidos de las clases.

Básicamente hemos probado uno a uno los métodos de la clase *Board.java*. Para ello, hemos creado un objeto de tipo *Board* y, a través del método, *getCell* (obteníamos la celda del array privado de la clase y así podíamos modificar su contenido) hemos ido rellenando las celdas que nos interesaban con los resultados de las partidas para hacer las pruebas.

Sin embargo nos ha sido imposible cumplir con lo que pedía el enunciado de que en una prueba gane el primer jugador que pone ficha y en otra el segundo ya que estábamos haciendo pruebas a más bajo nivel, al nivel del tablero. En otras palabras, lo que hemos hecho ha sido rellenar el tablero con dos identidades distintas y comprobar que los métodos *getCellsIfWinner* y *checkDraw* funcionaban cuando ganaba alguien o cuando había empate.

Tests dobles

Para realizar estos tests hemos seguido la guía que hay en el enunciado de la práctica. Primero hemos creado un metodo *@Before* para inicializar todos los tests.

En él, creamos el objeto de la clase *TicTacToeGameTest,* de manera que podamos simular una partida del tres en raya. A continuación creamos los mocks de las conexiones ya que el juego es en línea. Después creamos los dos jugadores, de la clase *Player* y hemos creado un método para iniciar el juego. Seguimos añadiendo a la partida las conexiones y los jugadores.

Seguidamente comprobamos que ambas conexiones reciben el evento *JOIN_GAME* con ambos jugadores. Finalmente rellenamos el tablero con algunas jugadas que nos servirán para las tres partidas de prueba.

Para finalizar, hacemos los tres tests que se proponen en el enunciado de la práctica.

Selenium

Para la realización de los tests de sistema con Selenium, hemos seguido las transparencias de la asignatura y hemos decidido utilizar *WebDriverManager*. Al principio nos ha sido complicado usar Selenium, pero una vez hemos entendido su funcionamiento, todo era muy intuitivo.

La única diferencia ha sido que hemos tenido que instanciar dos veces el *ChromeDriver()* para tener los dos navegadores jugando el uno contra el otro y tener en cuenta quién mueve cuándo.

Hemos creado dos métodos *@BeforeClass* y *@Before* para configurar las partidas antes de empezarlas. En el primero solo hemos preparado el *WebDriverManager* y hemos arrancado el programa. En el segundo hemos creado las variables necesarias para la ejecución de cada test particular y hemos abierto los navegadores.

También hemos creado métodos *@AfterClass* y *@After.* En la primera, apagamos el programa principal, ya que la ejecutamos al final de todos los tests y en la segunda apagamos los navegadores.

Para las acciones en el navegador, hemos inspeccionado el html de la página para saber cómo se llamaban los identificadores y usarlos en los tests.

Respecto a las dificultades a las que nos hemos enfrentado en esta parte de la prueba, hay que destacar que había un error bastante raro en los permisos y que para solucionarlo había que añadir una dependencia al *POM.xml*

Jenkins

Esta ha sido, sin duda, la parte más complicada de la práctica ya que hemos tenido varios problemas. Estos se han debido a que teníamos Jenkins instalado desde hace tiempo y estaba configurado el puerto 8080, que entra en conflicto con el programa que estamos testeando.

A la hora de crear el Jenkinsfile, simplemente insertamos el código que habiamos implementado en la versión web de Jenkins en una directiva node,

añadiendo además el comando checkout smc como se indica en la documentación (https://jenkins.io/doc/book/pipeline/jenkinsfile/)

En el apartado de preparation, obtenemos el código fuente del repositorio de github que estamos usando. En el apartado de test, ponemos los comandos de shell o cdm (dependiendo de nuestro sistema operativo) para llegar a la carpeta test y ejecutarlos. Para finalizar, always envía los resultados de los tests a un archivo .xml.

Conclusiones

Queremos destacar que nuestro proyecto es de IntelliJ, ligeramente distinto a los proyectos de Eclipse.

Nos ha sido bastante complicado reutilizar el código de unos tipos de tests a otros porque cada uno tenía su sintaxis propia y poco o nada tenían que ver los tests de Selenium con los tests unitarios. La única forma de reutilizar el código ha sido crear funciones genéricas para poner las fichas y así los tests han tenido todos una apariencia casi idéntica, favoreciendo que terceras personas puedan entenderlos. Sin embargo, dichas funciones son distintas de test a test.

Además se ha intentado poner métodos *@Before* y *@BeforeClass* que sean útiles y eviten repetir código en los tests. De este modo, hemos conseguido implementar todos los tipos de tests, por lo que la aplicación está bien cubierta.

Para concluir, nos gustaría reportar un bug en el programa ya que si dos jugadores con el mismo nombre se enfrentan, no se puede jugar ninguna partida.