Universidad Rey Juan Carlos

GRADO EN INGENIERÍA INFORMÁTICA

AMPLIACIÓN DE INGENIERÍA DEL SOFTWARE

PRÁCTICA 1: IMPLEMENTACIÓN DE PRUEBAS AUTOMÁTICAS

Alejandro Manuel Pazos Boquete

Javier Martín Torres

Introducción

Se ha pedido llevar a cabo la implementación de distintos tipos de pruebas automáticas para una aplicación correspondiente con el clásico juego de "Tres en raya" (TicTacToe en inglés). Estas pruebas consisten en:

- **Pruebas unitarias de la clase Board:** tienen como objetivo comprobar que susodicha clase detecta cuando se produce una victoria por parte de uno de los jugadores participantes o tiene lugar un empate. En otras palabras, verifica que los métodos getCellsIfWinner(…) y checkDraw() responden como se espera.

- **Pruebas con dobles de la clase TicTacToeGame:** tienen como objetivo comprobar que susodicha clase implementa el juego de forma correcta mediante el uso de dobles para llevar a cabo las simulaciones pertinentes.

- **Pruebas de sistema de la aplicación:** tienen como objetivo verificar si el funcionamiento de la aplicación es el esperado, haciendo uso de Selenium para llevar a cabo las distintas simulaciones.

Como añadido también se ha pedido la creación de un job en Jenkins que ejecute los test descritos y reporte los resultados en el formato adecuado.

Por lo tanto, el objetivo del presente documento es la descripción del proceso de elaboración y toma de decisiones en el mismo, de dichas tareas asignadas.[1]

Pruebas unitarias de la clase Board

A pesar de que la elaboración propiamente dicha de los test ha resultado ser bastante sencilla, el tratamiento de los casos de prueba usados por los mismos, llevado de tal forma que se pueda cumplir con el objetivo de conseguir generalización en el código nos ha hecho emplear más tiempo del que se esperaba.

Esto finalmente se ha conseguido creando para ambos métodos a verifcar (**getCellsIfWinner** y **checkDraw**) las clases **WinParameters** y **DrawParameters** respectivamente. Estas clases consisten básicamente en llevar cabo la parametrización de los test pero de tal forma que, mediante herencia, sea común a todos los test. Destacar

[1] Puesto que lo que se intenta en todo momento es explicar lo que ocurre en los test de manera sencilla, no se entra en muchos detalles técnicos. Por este motivo, no se especifica, por ejemplo, qué asserts u otros recursos del estilo se usan para realizar las comprobaciones. Para ello véase el código fuente.

además que aparte de los parámetros contienen también algunos métodos y atributos extras hechos para facilitar el entendimiento del código.

Ahora bien, en cuanto a los test en sí, se han creado otras dos clases: **BoardWinTest** y **BoardDrawTest**.

La primera, **BoardWinTest** contiene dos test correspondientes a la comprobación de si se ha producido una victoria para el jugador con el primer turno o, por el contrario, se ha producido una derrota para él. Los parámetros usados por estos test son heredados de la clase **WinParameters.**

Por otro lado, **BoardDrawTest** contiene únicamente un test encargado de comprobar si se produce un empate. Los parámetros usados son heredados de la clase **DrawParameters.**

Todos los test (**player1WinTest**, **player2WinTest y drawTest**) tienen una estructura y funcionamiento muy similar:

- En primer lugar, en el caso de **player1WinTest** y **player2WinTest,** mediante un for-each se realiza el marcaje de dichas celdas como si se tratase de los propios jugadores. En un caso con "X" y en el otro con "O".

  En **drawTest** se usa un for normal y se marca "X" o "O" en función de un booleano que almacena e indica el turno de cada jugador.

- Una vez hecho esto, se comprueba si se ha producido un empate mediante el método **checkDraw**.

- Y a continuación se comprueba, mediante **getCellsIfWinner** si se ha producido una victoria o una derrota. Es decir, se observa cuál de los dos jugadores ha ganado.

- Finalmente, la comprobación final consiste en ver si la resolución de las celdas es la esperada.

Cabe destacar que, tal y como se ha visto, en todos los test se hace uso y verificación de los métodos **getCellsIfWinner** y **checkDraw**.

Pruebas con dobles de la clase TicTacToeGame

Para la elaboración de las pruebas con dobles se ha seguido a modo de guion el pseudocódigo aportado en el enunciado de la práctica, haciendo el proceso bastante claro y simple.

Aquí nuevamente se produce herencia de las ya mencionadas clases **WinParameters** y **DrawParameters** para generalizar el código y se han creado otras dos para los test: **TicTacToeGameWinTest** que se encargará de comprobar que el juego está implementado correctamente haciendo las simulaciones pertinentes y en el caso de terminar el juego con una victoria por parte de alguno de los jugadores y **TicTacToeGameDrawTest** que tendrá la misma labor pero en caso de terminar el juego con un empate.

Al igual que antes ocurría con los test unitarios y con el mismo motivo, **TicTacToeGameWinTest** contiene dos test**,** mientras que **TicTacToeGameDrawTest** sólo uno, y todos cuentan con una estructura y funcionamiento muy similar entre ellos:

- En primer lugar se crean los objetos TicTacToeGame (**testedGame**), Connection (**c1** y **c2**) y Player (**p1** y **p2**) indicados y necesarios para realizar las simulaciones correspondientes.

- Acto seguido se añaden los objetos Connection y Player creados a **testedGame.**

- Tras esto, se comprueba que tanto **c1** y **c2** reciben el evento JOIN_GAME con **p1** y **p2** (cada uno de los objetos Connection con ambos objetos Player).

- Tras esto, con ayuda de bucles for se comprueba si tras cada marcaje, se produce el cambio turno entre los jugadores.

- A continuación, se verifica que se produce el envío de los mensajes de cambio de turno y marcado de celdas.

- Y finalmente se comprueba si se produce victoria y si los mensajes de fin de partida con el jugador esperado son enviados.

Pruebas de sistema de la aplicación

Entre las pruebas de sistema se distinguen las de los test de **WinSystemTest** y **DrawSystemTest**. Como sus nombres indican y en la línea de las demás clases del proyecto, la primera presenta los test correspondientes a la victoria y la derrota del jugador con el primer turno y hereda de **WinParameters**; la segunda contiene el test que simula un empate y hereda de **DrawParameters**.

El procedimiento seguido en estos test es simple:

- Se comienza por conectar los dos navegadores utilizados con el puerto 8080 del localhost.

- Accediendo a los elementos HTML de la aplicación por medio de sus identificadores, los dos navegadores introducen el nombre de los jugadores, inician el juego y marcan las casillas del tablero según convenga, dependiendo de si se busca la victoria del primer jugador, del segundo o el empate.

- Por último, se comprueba que el mensaje mostrado por *alert* es el esperado en ambos navegadores. Estos son cerrados después de cada test para eliminar todas las conexiones de la aplicación antes de comenzar el siguiente.

Integración continua

Se adjunta en el proyecto entregado el archivo **Jenkinsfile.txt**, que contiene la configuración del pipeline empleada en el job de Jenkins creado para ejecutar los test de la aplicación.

El funcionamiento del job es el estándar en Jenkins. En primer lugar, se descarga el proyecto del repositorio git especificado en el pipeline. Después se compila y se ejecutan los test que se hallan en él tal y como se haría si se hiciera desde el IDE. Por último, los resultados de los test se almacenan y quedan registrados en Jenkins.

También se envía en el proyecto el fichero de texto **consoleText.txt**, que muestra todo el texto que Jenkins imprime por pantalla durante realización de los test.

Ha sido necesario efectuar el comando *set JAVA_HOME=C:\Progra~1\Java\jdk1.8.0_101* antes de iniciar Jenkins para que la herramienta fuese capaz de localizar el JDK instalado en nuestro equipo.

Comentarios

- Se pretendía utilizar Firefox para establecer la conexión de uno de los dos jugadores para la ejecución de los test de sistema, pero no se ha conseguido implementar con la versión de este navegador que manejamos.

- Se ha intentado introducir un pequeño grado de aleatoriedad en los test de victorias, pero dado que el algoritmo empleado para ello no era totalmente

robusto, se ha descartado; en su lugar, se usan parámetros con una resolución predefinida.