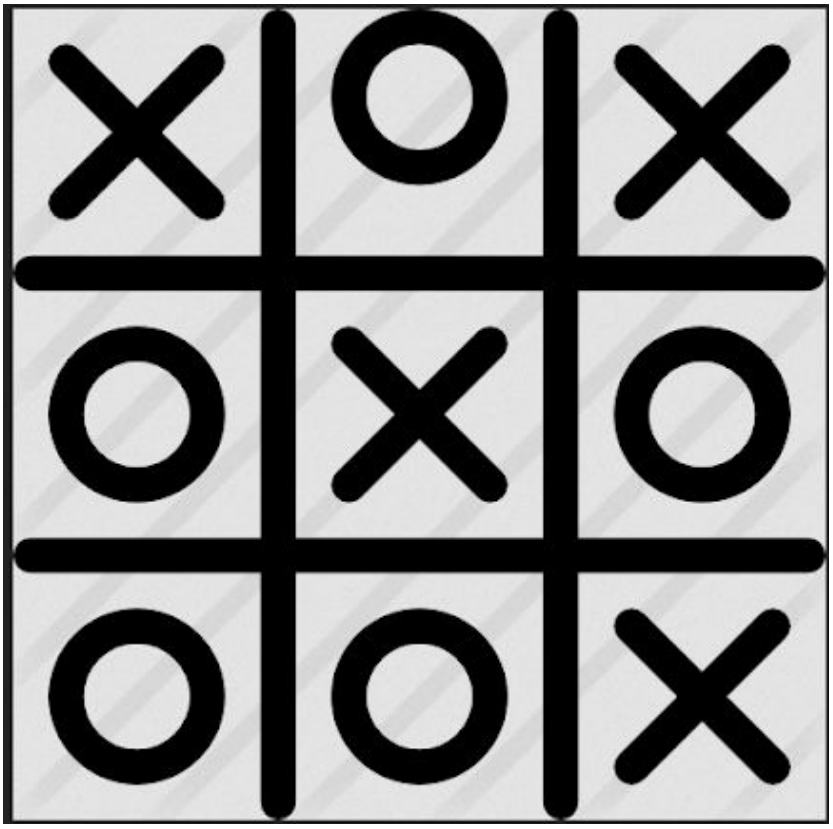
Memoria del Proyecto

Asignatura Ampliación de Ingeniería del Software, Curso 2019-20



Integrantes de la pareja:

- Smolarek Cobollo, Alejandro
- Villaverde Chavez, Joel

Funcionamiento de los tests

- **BoardTest**: Para estos test hemos utilizado simplemente el tablero de la aplicación junto con sus métodos. Antes de cada prueba creamos un tablero nuevo al que llamamos board. Una vez en los test, se simulará una partida real. En un array de enteros llamado *movimientos* metemos las celdas que serán seleccionadas. En el método **setValorCeldas** con el parámetro *movimientos* se asignará a cada celda el jugador correspondiente. Esto se hará recorriendo con un for el array de *movimientos* devolviendo un elemento *i* del array en cada ciclo. Antes de empezar a recorrerlo inicializamos un contador a 0, y en cada ciclo le sumamos una unidad. Si el contador es par esa celda *i* la escoge el J1, en caso de ser impar, la escoge el J2. Para esa asignación simplemente se usa el método **getCell** usando como parámetro la celda que especifique el array en ese ciclo. Entonces asignamos al atributo value de esa celda el jugador que sea.
  En el caso de que un jugador gane, se meterán en un array de enteros llamado *celdasganadoras* las tres celdas que le han dado la victoria a un jugador. Entonces en un **assertThat** se comprueba si el array de enteros que te devuelve el método de **getCellsIfWinner**, con uno de los jugadores como parámetro, es el mismo que el ya citado. Además con otro **assertThat** comprobaremos que lo que nos devuelve el método **checkDraw** es false puesto que no ha habido empate en el juego.
  En el caso de que haya empate se comprueba por cada jugador en un **assertThat** que el método **getCellsIfWinner** devuelve null debido a que ninguno ha ganado. Por tanto ahora el siguiente **assertThat** comprobará que lo que devuelve **checkDraw** es true.

- **TicTacToeGameTest**: Estos tests utilizarán: Un TicTacToeGame(g), dos Connection(c1, c2), dos Player(p1, p2) y una lista de Players(players). Antes de cada test se ejecutarán una serie de sentencias. Se inicializa el juego g, las dos conexiones c1 y c2(que en este caso son dos dobles de Connection). Se añaden ambas conexiones al juego con **addConnection**.También se crean dos jugadores p1 y p2, e igualmente se añaden con **addPlayer** al juego. Por último se inicializa la lista de players y se añade a la lista a los dos creados con **add.**
  En cada test, primero se verifica que el juego ha empezado (**verificarJuegoEmpezado**). Este método comprueba que en ambas conexiones se haya enviado dos veces el evento **JOIN_GAME,** ya que son dos jugadores y el parámetro devuelto sea players. Tras el método se inicializa un array de enteros *movimientos* con todas las celdas que se van a

escoger. Se llama entonces a **marcarYcomprobar** con el parámetro antes dicho. Aquí se verifica que se haya enviado en ambas conexiones el evento **SET_TURN** y el parámetro devuelto sea el primer jugador que entró.
Después se inicializa *cont* a 1 y se recorre el array de movimientos como en BoardTest. Primero g utiliza **mark** con el elemento *i* que toque. Se vuelve a verificar que se haya enviado en ambas conexiones el evento **SET_TURN** y el parámetro devuelto sea el jugador al que le toque el turno(elegido con *cont*%2). Después se resetean los mock de las conexiones y se suma 1 al contador. Al volver al test, g utilizará el método **mark** para la última jugada del juego. Inicializamos dos ArgumentCaptor<WinnerValue> para que al verificar en ambas conexiones que se ha enviado el evento **GAME_OVER** los argumentcaptor recuperen el parámetro devuelto.
Si el primero en jugar ha ganado se hará un **assertThat** en cada conexión que recupere el valor del argumento y el atributo player de ese argumento y se compruebe que son iguales al primer jugador. En caso de que el primero en jugar ha perdido se hará lo mismo comprobando que los atributos player sean iguales al segundo jugador.
Si los jugadores empatan no son necesarios los argumentcaptor porque en la verificación de **GAME_OVER** se comprobará en ambas conexiones que devuelve null.

- **SystemTest**: Estos tests utilizarán: dos WebDriver(browser1, browser2) que serán los navegadores, dos String con los nombres de los jugadores(nombreJ1, nombreJ2), varios WebDriverWait, para esperar al tablero(waitBoard1, waitBoard2) y esperar a la alerta de fin del juego(waitAlert1, waitAlert2), y por último dos String que tendrán como contenido la información de las alertas(alert1, alert2).
  Antes de todos los test a partir de la clase WebDriverManager se configuraran los drivers de Chrome y Firefox, y se inicia la aplicación con **start** de WebApp.
  Después de todos los test la aplicación termina con **stop** de WebApp.
  Antes de cada test se crean los drivers de cada browser y con **get** se conectan a un servidor local. Se da nombre a cada jugador. Una vez en la página en ambos navegadores se localiza el hueco en el que escribir el nombre, identificado por “nickname”, además de localizar el botón para empezar el juego, identificado por “startBtn”. A cada hueco se le manda con **sendKeys** el nombre de jugador correspondiente y se pasa al juego con el método **click** de los botones. Entonces se inicializan los WebDriverWait de espera del tablero. Después de cada test, se hará un **quit** de cada navegador en caso de no ser nulos.
  En cada test primero se inicializan en un array de enteros las celdas que serán escogidas(*movimientos*) y se llama al método **jugar** con ese array. En

este método primero cada waitBoard esperará que la celda que le toque esté presente(primer elemento del array para uno, el segundo para el otro). Se realiza lo ya explicado en otros test con la variable *cont* y recorrer el array de enteros con un for. Según sea par o impar la variable *cont* se seleccionará una celda en un navegador u otro. Para hacer esto cada navegador ejecuta **findElement** con un identificador “cell-” mas la celda que le especifique el elemento del array. Por último se hace **click** a esa celda. Después de este método se ejecuta **esperarCuadrosDeDialogo.** Aquí se inicializan los waitAlert. Se esperará hasta que la alerta esté presente. Se ejecuta entonces el método **guardarCuadrosDeDialogo** en el test. En este método se da valor a alert1 y alert2 a partir de la concatenación en cada navegador de **browser.switchTo().alert().getText()**. Después en el test se inicializa el String alertContent con la información que debería devolver la alerta según un jugador gane o empaten ambos. Por último se realizan dos **assertThat** comprobando que alert1 y alert2 son iguales a alertContent.

Job de Jenkins

Primero se configura jenkins con la versión de Maven y Java que se van a utilizar. Una vez hecho esto se crea un job, se le da un nombre y se escoge el tipo pipeline. En tools se declara “M3” de tipo Maven, tal como se ha configurado. Agent es de tipo any con lo que se escogerá un agente cualquiera para ejecutar el job. En este caso solo tiene una etapa a la que llamamos “Test”. Dentro de esta etapa se ejecuta un sh(dado que son comandos que se ejecutarán en el terminal). Dentro del shell habrá tan solo dos pasos que consisten en posicionarse en el directorio en el que se encuentra el proyecto ejecutando el comando cd con la ruta absoluta de donde se encuentre el proyecto y a continuación ejecutar el comando mvn test.
Hecho esto guardamos los cambios y ejecutamos la tarea. Una vez acabe se especificará si ha sido un éxito o ha habido errores. Para comprobar su resultado pinchamos en el número de ejecución, después a la salida del comando y podremos ver que todo se ha ejecutado según lo esperado.
Cabe destacar que solo se ha probado con linux debido a que es más amigable a la hora de realizar estos test.