

## Enunciado Práctica

# Evolución del software e Implementación de pruebas automáticas

### Objetivo

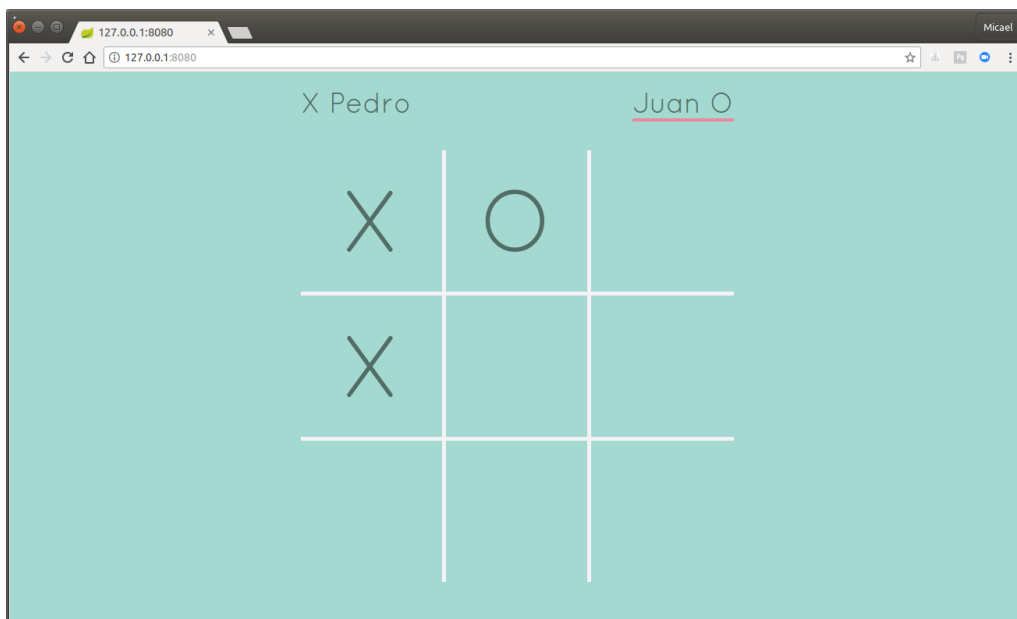
---

El objetivo de la práctica se divide en dos partes claramente diferenciadas: por un lado, el alumno tiene que añadir funcionalidad a una aplicación web real ya existente (ver apartado ENTREGA1) y por otro lado el alumno debe implementar diferentes tipos de pruebas automáticas sobre dicha aplicación web real (ver apartado ENTREGA2). De esta forma, el alumno podrá practicar los conceptos presentados en la asignatura.

### Aplicación Web Real

---

La aplicación proporcionada al alumno es el juego de las Tres en Raya en formato multijugador vía web. En inglés este juego se llama TicTacToe. Como se trata de un juego multijugador, para jugar una partida será necesario abrir dos navegadores (o dos pestañas del mismo navegador) apuntando a la dirección web <http://localhost:8080>.



Esta aplicación está implementada en Java con la tecnología Spring usando WebSockets. La aplicación se ejecuta pulsando el botón derecho sobre la clase WebApp y seleccionando la opción “Run As > Java Application”.

Además, está preparada para que se pueda arrancar y detener al ejecutar los tests de sistema con Selenium. Para ello, la clase WebApp tiene los métodos estáticos, start() y stop().

El juego está implementado con las siguientes clases en la parte servidor:

- **Board:** Representa el tablero del juego. Tiene métodos para determinar si un jugador tiene tres en raya o si el tablero está completo y ninguno tiene tres en raya (empate).
- **Player:** Clase que mantiene el nombre del jugador, el tipo de ficha (label) y su identificador.
- **TicTacToeGame:** Es la clase que gestiona el ciclo de vida del juego gestionando el tablero, el turno del jugador, etc. Los métodos de esta clase serán invocados cada vez que se reciba un mensaje de cualquiera de las conexiones con los navegadores web. Cada vez que haya un cambio en el juego, desde esta clase se envían eventos a los navegadores web usando los objetos Connection.
- **Connection:** Representa la conexión con un navegador web mediante WebSockets.
- **TicTacToeHandler:** Es la clase encargada de procesar los mensajes que llegan de los navegadores web e invocar los métodos de la clase TicTacToeGame.

La aplicación estará disponible en el aula virtual de la asignatura, en la sección de Evaluación.

Como se ha mencionado con anterioridad esta práctica consta de dos entregas diferenciadas: ENTREGA1 y ENTREGA2.

## ENTREGA 1: Evolución del Software

---

En esta primera entrega de la práctica trabajaremos las actividades relativas a gestión de proyectos, gestión de la configuración y evolución del software.

### Descripción

---

Para llevar a cabo estas actividades, se va a añadir nueva funcionalidad al software ya existente: Una página que muestre unas estadísticas sobre las partidas jugadas.

Para ello será necesario añadir: (a) una estructura de datos (en memoria) que recoja los resultados de las partidas jugadas y (b) una página HTML donde visualizar su información. Por ello tenemos un proyecto con dos tareas (a) y (b).

- Para llevar a cabo la actividad de gestión de proyectos, habrá que descomponer al menos tanto la tarea (a) como la (b) en subtareas, de forma que se puedan planificar adecuadamente. El plan de gestión de proyectos habrá de describirse y supervisarse utilizando Microsoft Project (disponible en la plataforma myapps).
- Para llevar a cabo la gestión de la configuración del software, habrá que utilizar una herramienta como Git, durante todo el proceso asociado a la ENTREGA1.

Desde un punto de vista técnico, la implementación se realizará siguiendo estos aspectos:

- La página de las estadísticas será accesible cuando se use la URL: <http://localhost:8080/stats>
- Se implementará con un controlador StatisticsController (encargado de generar la página HTML usando un fichero template) y un servicio StatisticsService (encargado de mantener la estructura de datos en memoria).
- La página de estadísticas mostrará una tabla en la que aparecerá, por cada usuario, el número de partidas ganadas, el número de partidas perdidas y el número de partidas empatadas.
- Se deberá modificar la clase TicTacToeHandler, en el método handleMessage(...). Cada vez que se aplica un nuevo movimiento en el juego (acción MARK), se deberá verificar si la partida ha terminado y si ha ganado algún jugador o ha habido empate (métodos checkDraw(...) y checkWinner(...)) de la clase TicTacToeGame). Con esa información, se actualizarán las estadísticas almacenadas en el StatisticsService.

### Procedimiento y fecha de entrega de la parte ENTREGA1

---

Esta parte de la práctica debe realizarse en **grupos de 4-5 personas**.

La **ENTREGA1** tiene como **fecha límite de entrega el día 20 de abril a las 23:55**, entrega que habrá de realizarse en el enlace incorporado a tal fin en la sección de Evaluación de Aula Virtual.

Se deberá entregar un **.zip cuyo nombre será la concatenación, en orden alfabético, de los primeros apellidos de los componentes del grupo, cuyo contenido** serán los tres directorios siguientes con el contenido que se especifica a continuación **(en caso contrario se suspenderá la parte práctica de la asignatura respecto de la ENTREGA1)**:

- **Directorio ES (Evolución del software):**

Debe contener un documento (la portada debe indicar Evolución del Software, Asignatura

Ampliación de Ingeniería del Software, Curso 2019-20, y los nombres (apellidos + nombre) de los componentes del grupo en orden alfabético por apellidos)) con la propuesta de modificación/ampliación con los detalles sobre el desarrollo de dicha modificación/ampliación (actividades de requisitos –diagrama de casos de uso-, análisis –diagrama de clases de análisis- y diseño –diagrama de clases de diseño- del Proceso Unificado de Desarrollo del Software). Las diferentes versiones de estas actividades estarán accesibles a partir del sistema de control de versiones Git.

- **Directorio GCS (Gestión de la configuración del software):**

Debe contener un documento del plan de la gestión de la configuración (la portada debe indicar Gestión de la Configuración del Software, Asignatura Ampliación de Ingeniería del Software, Curso 2019-20, y los nombres (apellidos + nombre) de los componentes del grupo en orden alfabético por apellidos)) detallando el proceso de control de cambios según los principios de la GCS (esto es, las actividades de control de la configuración tal como se han visto en teoría). El documento contendrá además en alguna de las secciones del documento, la siguiente información:

1. Detalle de cómo se ha utilizado el proceso de control de cambios en dos cambios consensuados por vuestro grupo en el seno del proyecto.

Estos procesos de control de cambios han de ser coherentes con la historia del repositorio tal como se muestra en Git. Para mayor claridad, se ha de indicar la fecha en la que se consensúa un cambio para poder verificar la evolución posterior en el repositorio. En particular, se pide que se indique cuál de los *commits* posteriores se considera como la “línea base” correspondiente en cada caso.

- **Directorio GP (Gestión de proyecto):**

Debe contener un único documento (la portada debe indicar Gestión de Proyectos Software, Asignatura Ampliación de Ingeniería del Software, Curso 2019-20, y los nombres (apellidos + nombre) de los componentes del grupo en orden alfabético por apellidos)) que contenga al menos la siguiente información:

- Diagrama de Gantt con la enumeración de tareas y subtareas del desarrollo, correspondientes a las actividades mencionadas en el punto Evolución del Software, y sus tiempos correspondientes.
- Red de Precedencia (información análoga a PERT)

---

## ENTREGA 2: Pruebas automáticas e Integración Continua

---

### Pruebas automáticas

---

El alumno tiene que implementar pruebas automáticas para asegurarse de que el juego funciona correctamente. Concretamente, tiene que implementar pruebas de diferentes tipos:

- **Pruebas unitarias de la clase Board:** En estas pruebas se tiene que comprobar que la clase implementa correctamente la detección de que un jugador ha ganado o de que se ha empatado. Concretamente, 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.

- **Pruebas con dobles de la clase `TicTacToeGame`:** En estas pruebas se tiene que comprobar que la clase `TicTacToeGame` implementa de forma adecuada el juego. Es decir, 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` (que representan una comunicación WebSockets). Estas conexiones simuladas se utilizarán para verificar si los eventos que envía `TicTacToeGame` a los navegadores web (representados por los dobles de `Connection`) son los esperados. El flujo de eventos entre los navegadores y el servidor es el siguiente:
  - Cuando los navegadores web se conectan al servidor, se crean objetos `Connection` y se dan de alta en el juego llamando al método “`addConnection(...)`”
  - Cuando los jugadores ponen su nombre se envía un mensaje al servidor que se traduce en la creación de los objetos `Player` y la invocación del método `addPlayer(...)` por cada jugador.
  - Cada vez que se añade un jugador, el juego genera un evento de tipo `JOIN_GAME` que se envía a los navegadores web con la lista de jugadores que se han registrado hasta ese momento.
  - Una vez que se han registrado los dos jugadores, se envía un evento `SET_TURN` indicando el jugador que tiene el turno. De esta forma, los navegadores web pueden actualizar su interfaz en consecuencia.
  - Cuando en el navegador se marca una casilla, se invoca el método `mark(...)` en el juego indicando el identificador de la celda que ha sido marcada. El propio juego sabe qué tipo de ficha poner en esa celda por el turno actual.
  - Cada vez que se marca una celda, se vuelve a enviar el evento de `SET_TURN`.
  - Cuando alguno de los dos jugadores gana o se detecta el empate, el servidor envía el evento de `GAME_OVER`. Si hay empate, el parámetro del evento se envía vacío. Si gana alguno de los jugadores, se envía su identificador y las celdas que contienen las fichas de ese jugador que forman la línea.

En pseudocódigo, el test procedería de la siguiente manera:

- 1) Crear el objeto `TicTacToeGame`
- 2) Crear los dobles de los objetos `Connection`
- 3) Añadir los dobles al objeto `TicTacToeGame`
- 4) Crear los dos jugadores (objetos `Player`)
- 5) Añadir los jugadores al objeto `TicTacToeGame`
- 6) Comprobar que la conexión 1 recibe el evento `JOIN_GAME`, con ambos jugadores
- 7) Comprobar que la conexión 2 recibe el evento `JOIN_GAME`, con ambos jugadores

- 8) Por turnos, cada jugador va marcando una casilla invocando el método mark de TicTacToeGame, comprobando que el turno cambia
- 9) Al final se comprueba que el juego acaba y que dependiendo de las casillas marcadas uno de los jugadores gana o hay empate.

Para la implementación de estos tests puede ser interesante el uso de algunas funcionalidades de Mockito no vistas en clase:

- **argThat(...):** Este método permite usar un matcher de hamcrest en un verify de Mockito. Por ejemplo, para verificar que el valor del evento que recibe una conexión es una lista que contiene elementos, se puede usar el siguiente esquema (siendo hasItems un matcher de hamcrest):

```
Connection c1 = mock(Connection.class);
```

```
...
```

```
verify(c1).sendEvent(  
    eq(EventType.JOIN_GAME), argThat(hasItems(p0, p1)));
```

- **reset(mock):** El método reset permite borrar el registro de llamadas a los métodos del mock. Es ideal para verificar las llamadas a los métodos del mock que se han producido únicamente desde que se ha hecho reset. Si no se usa reset, habría que verificar el número de veces que ha sido llamado un método durante todo el test.

```
reset(c1);
```

- **ArgumentCaptor:** Esta clase permite recuperar los valores pasados como parámetro a los métodos del mock. Es ideal cuando los matchers no tienen expresividad suficiente para verificar el parámetro.

```
ArgumentCaptor<WinnerValue> argument =
```

```
    ArgumentCaptor.forClass(WinnerValue.class);
```

```
verify(c1).sendEvent(eq(EventType.GAME_OVER), argument.capture());
```

```
Object event = argument.getValue();
```

- **Pruebas de sistema de la aplicación:** Para verificar que la aplicación completa funciona correctamente se implementarán pruebas de sistema con Selenium. Para simular una partida el test iniciará dos navegadores web 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.

Para obtener el mensaje del alert se utiliza el código:

```
browser1.switchTo().alert().getText()
```

En todos los tipos de prueba hay que comprobar la misma funcionalidad, pero el test se implementa a diferentes niveles (unitario, con dobles y de sistema). En concreto hay que implementar al menos

los siguientes tests por cada tipo:

- El primer jugador que pone ficha gana.
- El primer jugador que pone ficha pierde.
- Ninguno de los jugadores gana. Hay empate.

En total tienen que implementarse 9 tests (3 de cada tipo representando cada uno de los escenarios anteriores). Como los tests del mismo tipo van a ser muy parecidos entre sí, se pide que se estructure el código de tal forma que se reutilice y no haya duplicaciones. Eso requiere un esfuerzo de generalización del código, pero merece la pena porque favorece la comprensión del código y añadir más casos de prueba es bastante sencillo.

### Integración continua

---

Además de la implementación de los tests, se pide crear un job en Jenkins que ejecutará todos los tests del proyecto y reportará los resultados en el formato adecuado para que lo pueda capturar Jenkins.

La configuración de este job debe guardarse en un fichero Jenkinsfile en la raíz del proyecto Eclipse entregado.

La aplicación web se puede arrancar directamente desde los tests, sin necesidad de generar un fichero .jar y arrancar el fichero de forma independiente a los tests.

Para evitar las incompatibilidades entre diferentes tipos y versiones del sistema operativo, el job de Jenkins deberá funcionar correctamente en un sistema linux. Además, para evitar problemas con el repositorio de código, se asumirá que el código fuente estará almacenado en disco en la carpeta /ais/<nombre-practica>/. Siendo <nombre-practica> el nombre del fichero .zip.

### Procedimiento y fecha de entrega de la parte ENTREGA2

---

Esta parte de la práctica debe realizarse de **forma individual** o por **parejas**. Los tests se pueden implementar sobre el código original propuesto como enunciado o sobre el código entregado en la ENTREGA1.

La **ENTREGA2** tiene como **fecha límite de entrega el día 22 de mayo a las 23:55**, entrega que habrá de realizarse en el enlace incorporado a tal fin en la sección de Evaluación de Aula Virtual.

Se deberá entregar un **.zip cuyo contenido** será la carpeta src y el fichero pom.xml. Además, deberán estar en la raíz del fichero comprimido (no debe haber una carpeta “practica” o similar).

El nombre del fichero .zip tiene que ser igual al identificador del alumno en la URJC (la parte antes de la @ del correo electrónico del alumno en la URJC). En caso de dos alumnos, se incluirán ambos nombres separados por “-” y sólo uno de ellos deberá realizar la entrega.

En el fichero pom.xml se deberá incluir el siguiente nombre del proyecto (donde nombre.alumno corresponde con el identificador del alumno o alumnos):

```
<groupId>es.codeurjc.ais</groupId>
<artifactId>nombre.alumno</artifactId>
```

`<version>0.0.1-SNAPSHOT</version>`

Además del código fuente, se deberá elaborar una memoria explicativa del mismo en formato PDF. La memoria deberá guardarse en la carpeta `src/main/resources` y tendrá el nombre “`memoria.pdf`”.

En la memoria se deberá describir el funcionamiento de los tests implementados, de forma que un desarrollador que no conozca la aplicación pueda entender qué hacen los tests. También se deberá describir el funcionamiento del job de Jenkins. La memoria deberá tener una longitud de 4 páginas (incluyendo la página de la portada).

### **Evaluación de la ENTREGA2**

---

La ENTREGA2 se evaluará en base a los siguientes criterios:

- Que los tests sean correctos y verifiquen realmente que el comportamiento obtenido es el esperado en los supuestos indicados en el enunciado.
- Que no haya código duplicado.
- Que la memoria explique con claridad los tests desarrollados y el job de Jenkins.
- Que los tests se ejecuten con el comando “`mvn test`” en un sistema con Maven instalado.
- Que el job de Jenkins se ejecute correctamente en un sistema linux.