Sandra Pérez Guerrero - 1459296

Ismael Sánchez - 1457867

# Hundir la flota

# Juego

**Función: comprobarPosicionCursor**

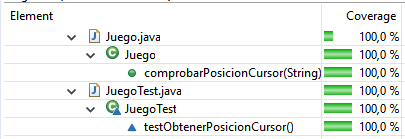
**Funcionalidad**: Método que devuelve si la posición introducida por el jugador es correcta.

**Localización**:

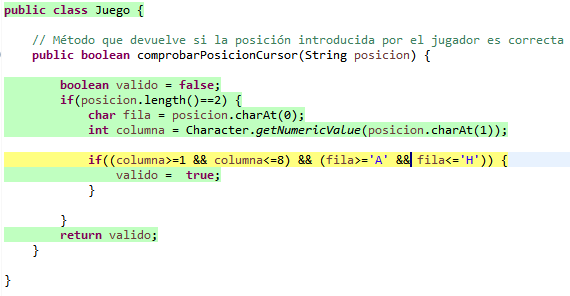
* Archivo: Juego.java
* Clase: Juego
* Método: comprobarPosicionCursor(String posicion)

**Test**:

* Archivo: JuegoTest.java
* Clase: JuegoTest
* Método: testObtenerPosicionCursor()
* Test realizado:
  + **Caja negra**:
    - Particiones equivalentes
    - Valores límite y frontera
  + **Caja blanca:**
    - Statement coverage



* + - Decision coverage/Condition coverage



**Funcion: pedirPosicion (Se ha eliminado el test, ya que era función void)**

**Funcionalidad**: Método lee una posición de Teclado de tipo Carácter + Integer, ej: A1. Se le pasa un parámetro integer longitud que se mostrará en un system out como longitud del barco. Esta función llamará a la del Teclado, que hará un lector.next()

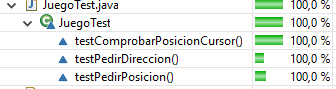
**En las primeras versiones era necesario testearlas, después ya teníamos otros tests que recogían esta información**

**Localización**:

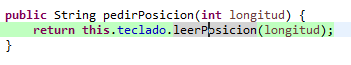
* Archivo: Juego.java
* Clase: Juego
* Método: pedirPosicion(int longitud)

**Test**:

* Archivo: JuegoTest.java
* Clase: JuegoTest
* Método: testPedirPosicion()
* Test realizado:
  + **Caja negra**:
    - Particiones equivalentes
  + **Caja blanca:**
    - Statement coverage



* + - Decision coverage/Condition coverage



**Funcion: pedirDireccion (Se ha eliminado el test, ya que era función void)**

**Funcionalidad**: Método lee una dirección desde Teclado. Esta función llamará a la del Teclado, que hará un lector.next().

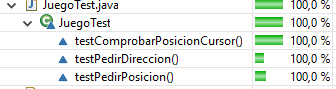
**En las primeras versiones era necesario testearlas, después ya teníamos otros tests que recogían esta información**

**Localización**:

* Archivo: Juego.java
* Clase: Juego
* Método: pedirDireccion()

**Test**:

* Archivo: JuegoTest.java
* Clase: JuegoTest
* Método: testPedirDireccion()
* Test realizado:
  + **Caja negra**:
    - Particiones equivalentes
  + **Caja blanca:**
    - Statement coverage



* + - Decision coverage/Condition coverage



**Funcion: comprobarLimitesTablero**

**Funcionalidad**: Método lee una dirección desde Teclado. Esta función llamará a la del Teclado, que hará un lector.next().

**Localización**:

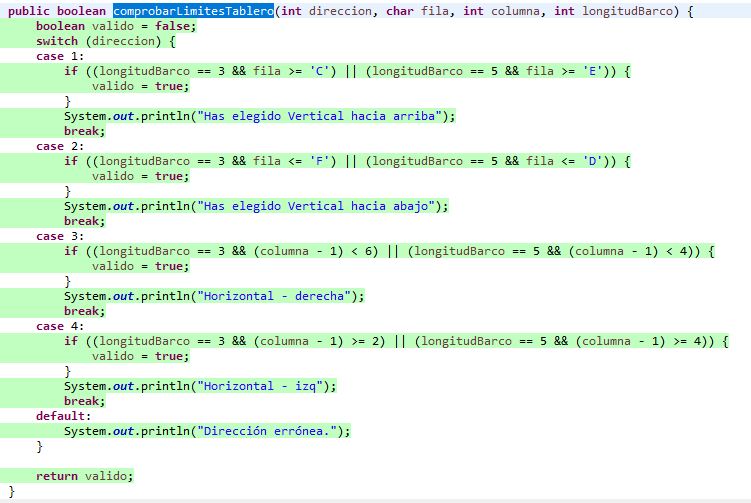
* Archivo: Juego.java
* Clase: Juego
* Método: pedirDireccion()

**Test**:

* Archivo: JuegoTest.java
* Clase: JuegoTest
* Método: testComprobarLimitesTablero()
* Test realizado:
  + **Caja negra**:
    - Particiones equivalentes
    - Valores frontera
  + **Caja blanca:**
    - Statement coverage



* + - Decision coverage/Condition coverage



**Funcion: colocarBarcos**

**Funcionalidad**: Método de alto nivel que, a partir de un jugador, ira pidiendo al usuario todos los tipos de barco a colocar y la direccion en la que colocarlos. Comprobará que la posición es correcta con comprobarPosicionCursor() y comprobarLimitesTablero(). Si todo és correcto, se llamará a la función anadirBarco() de la clase Mapa.

**Localización**:

* Archivo: Juego.java
* Clase: Juego
* Método: colocarBarcos(Jugador jugador)

**Test**:

Funcion de alto nivel, los tests están en cada funciíon a la que llama:

* comprobarPosicionCursor - Juego.java
* pedirDireccion - Juego.java
* anadirBarco - Mapa.java

**Funcion: compararMapas**

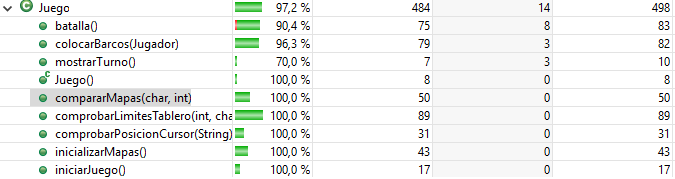
**Funcionalidad**: Método qué, a partir de la fila y la columna, comprobará si ha tocado el mapa del rival. Para ello, mirara el mapa oculto (muestra las posiciones sin desvelar y se tienen que ir abriendo) con las del mapa del rival, y revelará lo que hay en esa posición.

**Localización**:

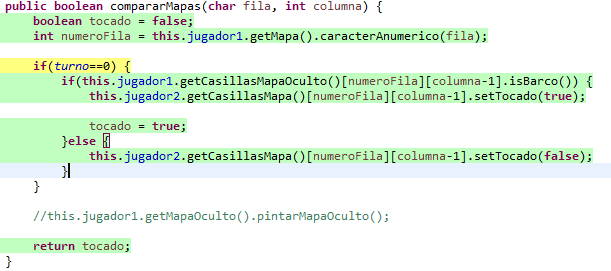
* Archivo: Juego.java
* Clase: Juego
* Método: compararMapas(char fila, int columna)

**Test**:

* Archivo: JuegoTest.java
* Clase: JuegoTest
* Método: testCompararMapas()
* Test realizado:
  + **Caja negra**:
    - Particiones equivalentes
  + **Caja blanca:**
    - Statement coverage



* + - Decision coverage/Condition coverage



# Mapa

**Función: testCaracterAnumerico**

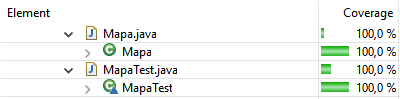
**Funcionalidad**: Método que devuelve si la posición introducida por el jugador es correcta.

**Localización**:

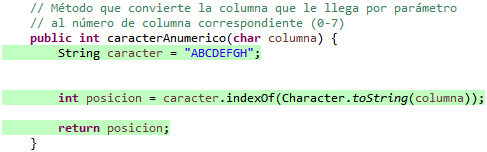
* Archivo: Mapa.java
* Clase: Mapa
* Método: caracterAnumerico(char columna)

**Test**:

* Archivo: MapaTest.java
* Clase: MapaTest
* Método: testCaracterAnumerico ()
* Test realizado:
  + **Caja negra**:
    - Particiones equivalentes
    - Valores límite y frontera
  + **Caja blanca:**
    - Statement coverage



* + - Decision coverage/Condition coverage



**Función: generarMapaVacio**

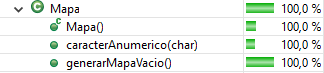
**Funcionalidad**: Método que implementa el mapa que se utilizara como base para los jugadores donde pondrán sus barcos o intentaran hundir los del rival.

**Localización**:

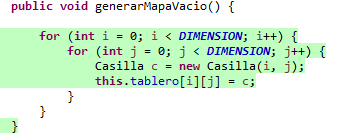
* Archivo: Mapa.java
* Clase: Mapa
* Método: generarMapaVacio(char columna)

**Test**:

* Archivo: MapaTest.java
* Clase: MapaTest
* Método: testGenerarMapaVacio()
* Test realizado:
  + **Caja negra**:
    - Particiones equivalentes
    - Valores frontera (Consideramos los bordes como frontera)
  + **Caja blanca:**
    - Statement coverage



* + - Decision coverage/Condition coverage



**Función: anadirBarco**

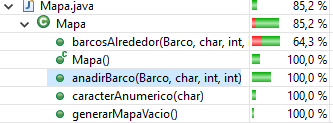
**Funcionalidad**: Método que colocará un barco en el mapa. Los parámetros que le llegan son barco, fila y columna del mapa y dirección y todos ellos son correctos, ya que hay funciones de comprobación y sus tests implementados.

**Localización**:

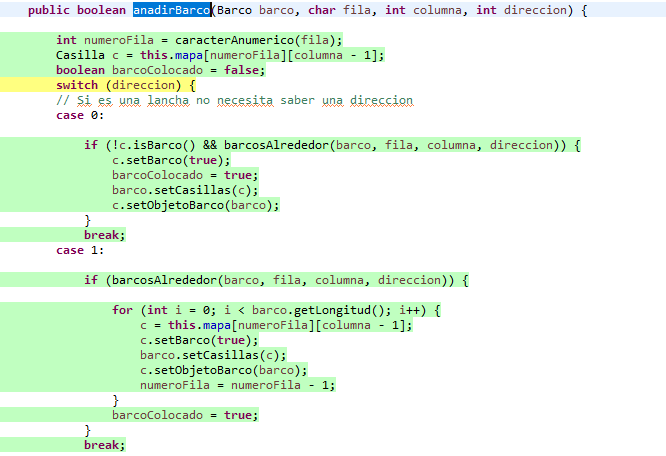
* Archivo: Mapa.java
* Clase: Mapa
* Método: anadirBarco(Barco barco, char fila, int columna)

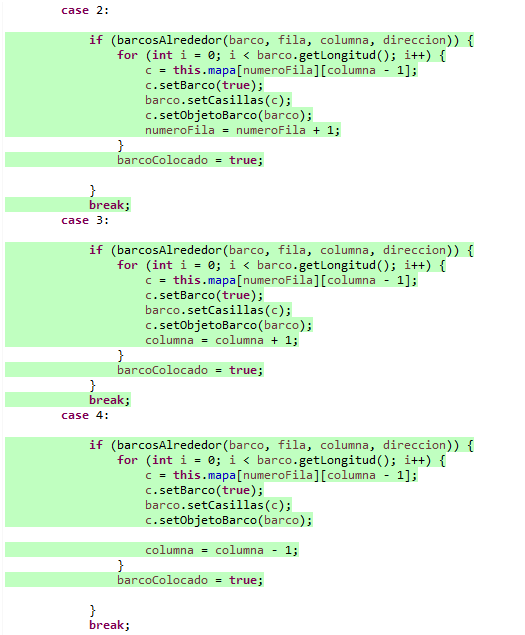
**Test**:

* Archivo: MapaTest.java
* Clase: MapaTest
* Método: testAnadirBarco()
* Test realizado:
  + **Caja negra**:
    - Particiones equivalentes
    - Valores frontera (Consideramos los bordes como frontera)
  + **Caja blanca:**
    - Statement coverage



* + - Decision coverage/Condition coverage







**Función: barcosAlrededor**

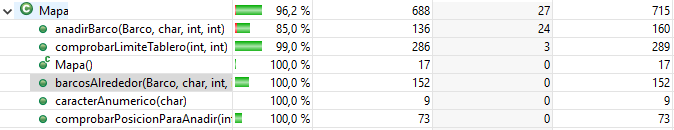
**Funcionalidad**: Método que comprueba que a partir de una posición escogida, no haya ningún otro barco alrededor para poder posicionarlo en el mapa, ya que una de las condiciones del juego és que tiene que haber agua en sus bordes (si puede haber en la diagonal, ya que solo construimos barcos horizontales y verticales).

**Localización**:

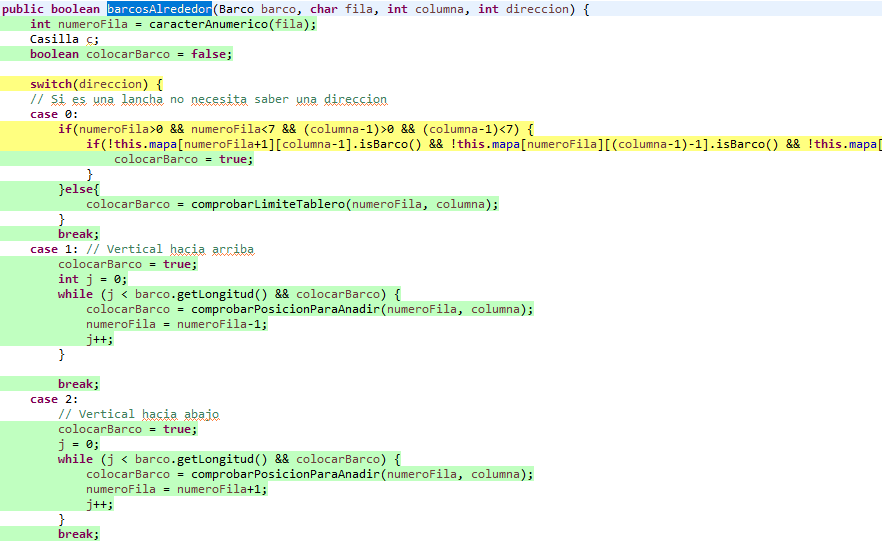
* Archivo: Mapa.java
* Clase: Mapa
* Método: barcosAlrededor(Barco barco, char fila, int columna, int direccion)

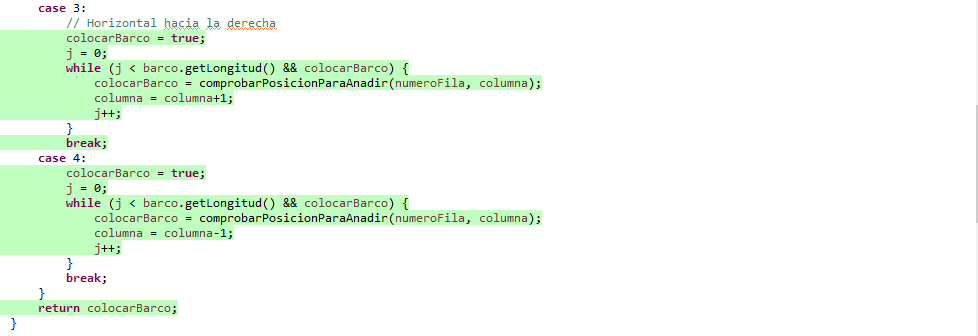
**Test**:

* Archivo: MapaTest.java
* Clase: MapaTest
* Método: testBarcosAlrededor()
* Test realizado:
  + **Caja negra**:
    - Particiones equivalentes
    - Valores frontera (Consideramos los bordes como frontera)
  + **Caja blanca:**
    - Statement coverage



* + - Decision coverage/Condition coverage





**Función: comprobarLimiteTablero**

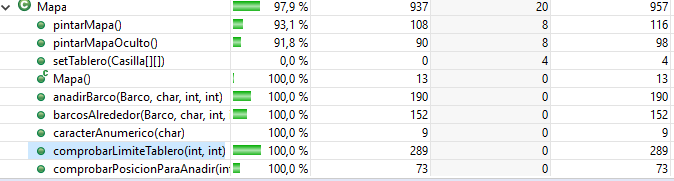
**Funcionalidad**: Método que comprueba que, a partir de una posición escogida, el barco no sobrepase los límites establecidos teniendo en cuenta la dirección en la que se quiere poner el barco. Por ejemplo, si el usuario quiere colocar un barco de longitud tres en A1, y dice que la dirección es hacia arriba, detectaríamos que se saldría del mapa y esta función retornaría false.

**Localización**:

* Archivo: Mapa.java
* Clase: Mapa
* Método: comprobarLimiteTablero(int numeroFila, int columna)

**Test**:

* Archivo: MapaTest.java
* Clase: MapaTest
* Método: testComprobarLimiteTablero()
* Test realizado:
  + **Caja negra**:
    - Particiones equivalentes
    - Valores frontera (Consideramos los bordes como frontera)
  + **Caja blanca:**
    - Statement coverage



* + - Decision coverage/Condition coverage

# 

# Juego

**Unicamente tenemos la función isBarcoHundido, las demás son funciones void i/o de alto nivel.**

**Función: isBarcoHundido**

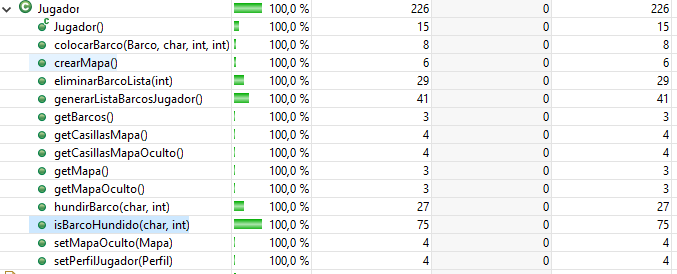
**Funcionalidad**: Método que retornara un booleano comprobando si nuestro barco ha sido hundido a partir de una fila y columna. Miraremos si todas las casillas del barco han sido tocadas o no.

**Localización**:

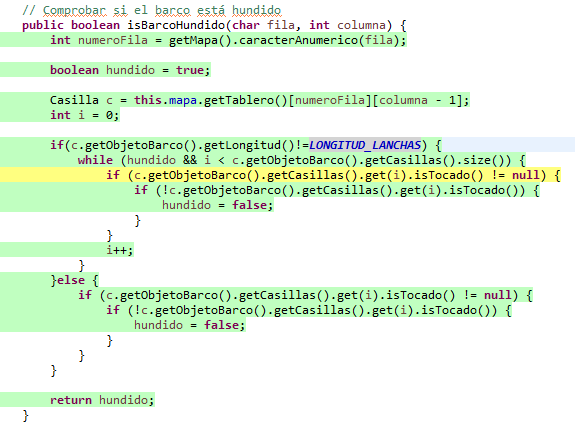
* Archivo: Jugador.java
* Clase: Jugador
* Método: isBarcoHundido(int fila, int columna)

**Test**:

* Archivo: JugadorTest.java
* Clase: JugadorTest
* Método: testIsBarcoHundido()
* Test realizado:
  + **Caja negra**:
    - Particiones equivalentes
  + **Caja blanca:**
    - Statement coverage



* + - Decision coverage/Condition coverage



# MockObject + Teclado - MockTeclado

**Funcionalidad**: Crear un MockObject del Teclado para simular la introducción que hará por un teclado un usuario. Crearemos diferentes registros de posiciones del tablero para los tests.

Crearemos a parte una clase Teclado para llevar a cabo los tests. No tenemos ningún tests sobre estas clases, pero se usarán para realizar otros tests que necesiten de la información que ha de introducir un usuario.

**Localización**:

* Archivo: MockTeclado.java
* Clase: MockTeclado
* Método: comprobarPosicionCursor(String posicion)

# MockObject + Tablero - MockTablero

**Funcionalidad**: Crear un MockObject del tablero para simular la creación de un mapa con barcos ya insertados. La función original crea el mapa vacio que esperará las intrucciones

**Localización**:

* Archivo: MockTablero.java
* Clase: MockTablero
* Método: generarMapaVacio()

# MockObject + Tablero - MockTableroJugador

**Funcionalidad**: Crear un MockObject del tablero para simular la creación de un mapa con barcos ya insertados. Es una versión extendida de nuestro anterior MockTablero pero este añadira todo tipo de barcos y los relacionará con las casillas para hacer las pruebas de posicionar i/o hundir barcos.

**Localización**:

* Archivo: MockTableroJugador.java
* Clase: MockTablero
* Método: generarMapaVacio() y ponerParteBarco(Barco b, int fila, int col, boolean tocado)

# MockObject + Perfil - MockPerfil

**Funcionalidad**: Crear un MockObject del Perfil para simular la creación del mapa de jugador que creara un jugador y en su mapa crearemos un MockTableroJugador

**Localización**:

* Archivo: MockPerfil.java
* Clase: MockPerfil
* Método: generarMapaJugador()