<Iván López Medina>

<Ivan Rubi Lozano>

<Guerra de Barcos>

**Funcionalitat**: Insertar Barco

**Localització**: Tablero.java, setBarco()

**Test**: TestTablero.java. Para partición equivalente se crean los métodos test\_ColocarBarcoOcupadoHorizontal(),test\_ColocarBarcoOcupadoVertical(), test\_ColocarBarcoVacioHorizontal(), test\_ColocarBarcoVacioVertical(). Estos tests definen los criterios de valido e invalido. Con las funciones test\_ColocarBarcoFueraVertical() y test\_ColocarBarcoFueraHorizontal() utilizamos la técnica de valores límites. Todo esto son pruebas de caja negra.

**Funcionalitat**: Crear tablero

**Localització**: Tablero.java, tablero()

**Test**: TestTablero.java. Se realizan dos tests de particion equivalente de caja negra. Test\_insertTableroValido() y test\_insertTableroInvalido(). Estos tests definen los criterios de valido e invalido.

**Funcionalitat**: Crear barco

**Localització**: Barco.java, barco(), setTamano()

**Test**: TestBarco.java. test\_instanciarBarco(). Testeamos que en función del nombre proporcionado podamos instanciar los 4 tipos de barcos. Test\_hundirBarco() en este utilizamos los valores límite para comprobar que no Podemos restar vida negative a un barco.

**Funcionalitat**: Disparar

**Localització**: Jugador.java, disparar()

**Test**: TestJugador.java. test\_disparo(). Función en la que hacemos uso de un mockobject que simula un teclado el cual tendrá un array de filas y columnas a las cuales ir disparando. Realizamos diferentes pruebas de caja negra con particiones equivalentes y valores límite.

**Funcionalitat**: Imprimir Tablero

**Localització**: Tablero.java, usado en main()

**Test**: JuegoTest.java. test\_pintarTablero. Se crea un test el cual inicia una partida y con un mockobject que hace de teclado simulamos una partida para comprobar que imprima bien los valores del tablero (tocado, hundido, agua…)

**Funcionalitat**: Obtener barco

**Localització**: Tablero.java, getBarco()

**Test**: TableroTest.java. test\_loopGetBarco. Se hace loop testing con 0 y 1 iteraciones.

**Funcionalitat**: Insercion aleatoria de barcos

**Localització**: Tablero.java, insertarBarcosAleatorios()

**Test**: test\_insercionAleatoria().Se generan aleatoriamente todos los barcos, no necesita mucho testing ya que se testea a la hora de testear todos los barcos e inserciones pero testeamos que se introduzcan todos de manera correcta.Introducimos 0, 1, 4 y 5. 4 Es el correcto.

**Funcionalitat**: Condición de victoria

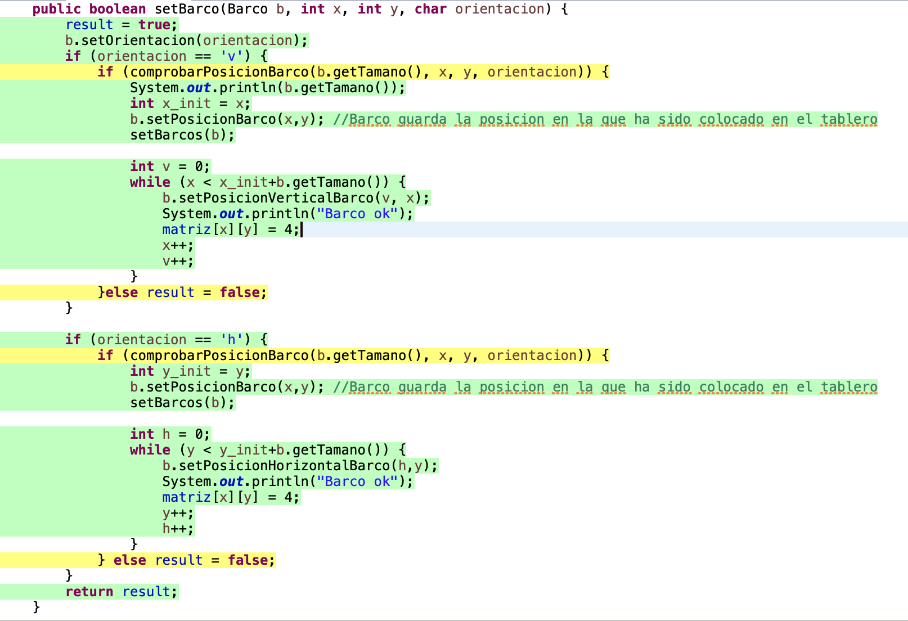
**Localització**: Juego.java, insertarBarcosAleatorios()

**Test**: test\_condicionVictoria()Se testea la condición de victoria, que cuando se hunden los 4 barcos termina el juego. Se testea con diferente número de barcos, ya que es importante que termine.

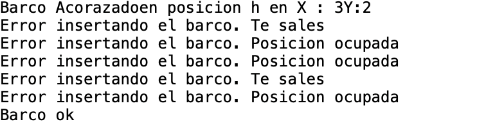
**Funcionalitat**: Code Coverage function setBarco

**Localització**: Clase Tablero método setBarco

**Test**: Comprobamos que pasamos por todas las líneas de código, este es solo un ejemplo para tenerlo documentado, ya que seguimos el Statement Coverage en todo el proyecto



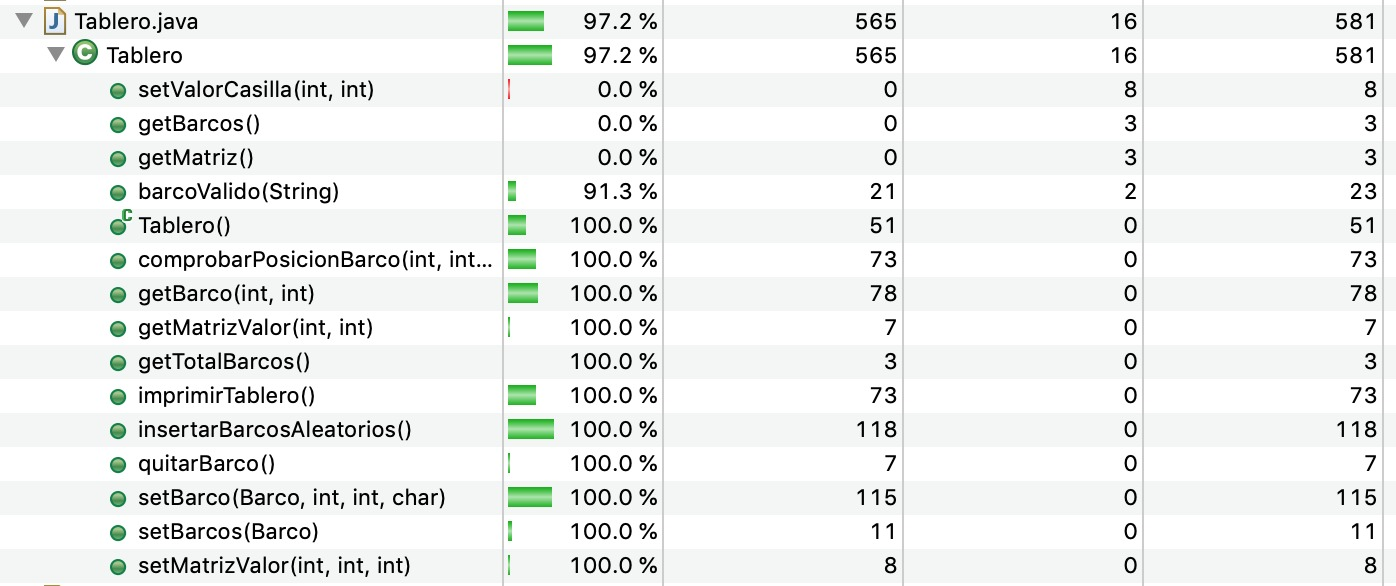
El mismo método setBarco llama al método comprobarPosicionBarco, que revisa que no haya otro barco en esa posición. Como podemos ver hay líneas rojas, esas líneas son necesarias, ya que los barcos se colocan en el tablero de manera ALEATORIA. Pues sin ellas generaríamos un tablero incorrecto.

**Funcionalitat**: Statement coverage de las clases

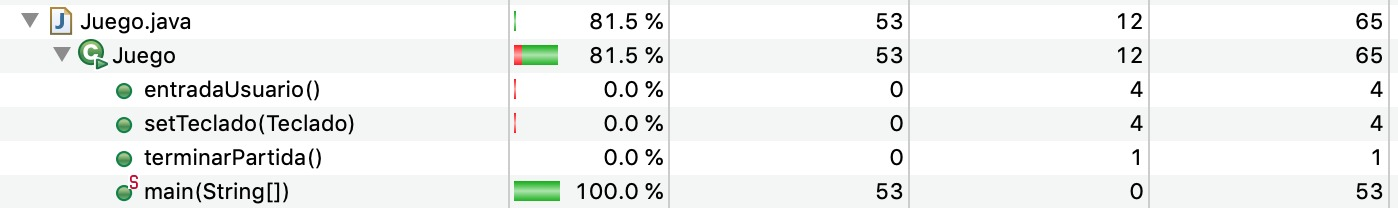
Al haber métodos que se usan en tests no se puede llegar al 100% en esta ejecución ya que la realizamos sobre el main,

**Localització**: Clase Tablero



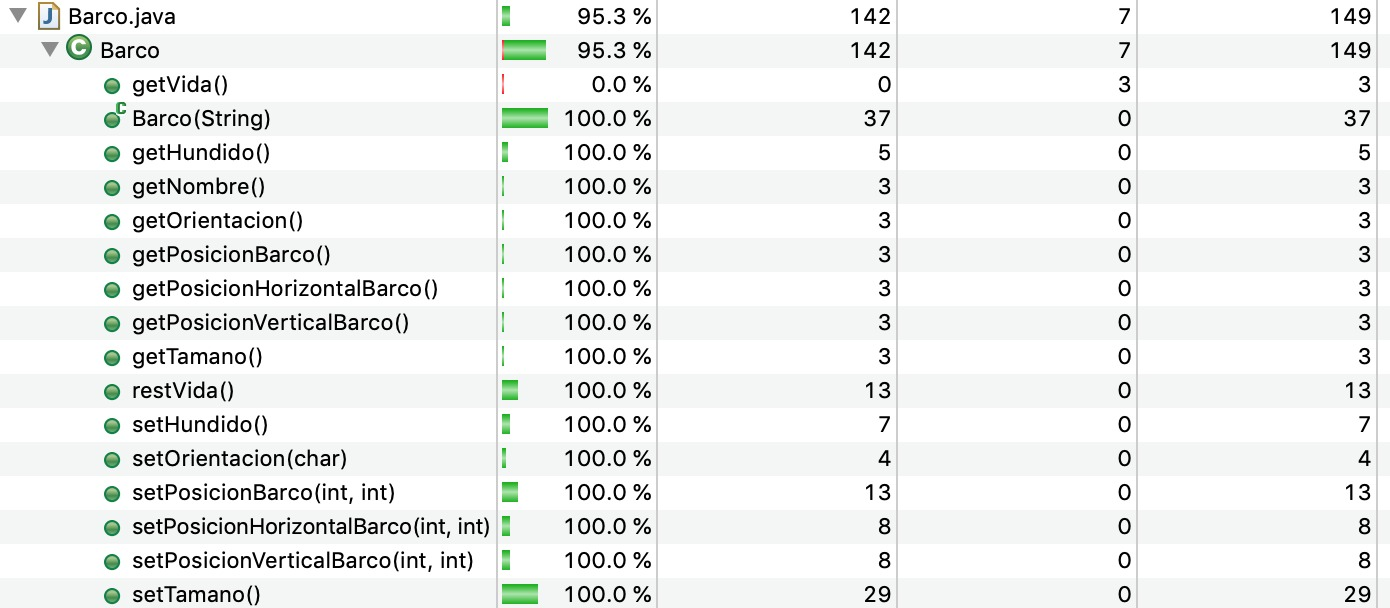
**Funcionalitat**: Statement coverage

**Localització**: Clase Juego



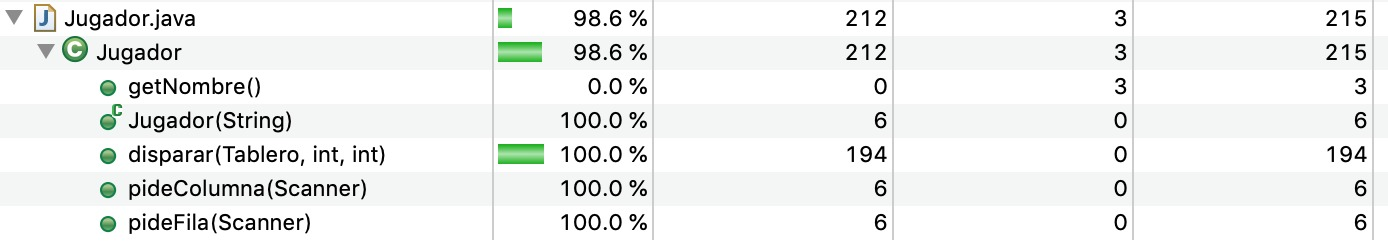
**Funcionalitat**: Statement coverage

**Localització**: Clase Barco



**Funcionalitat**: Statement coverage

**Localització**: Jugador



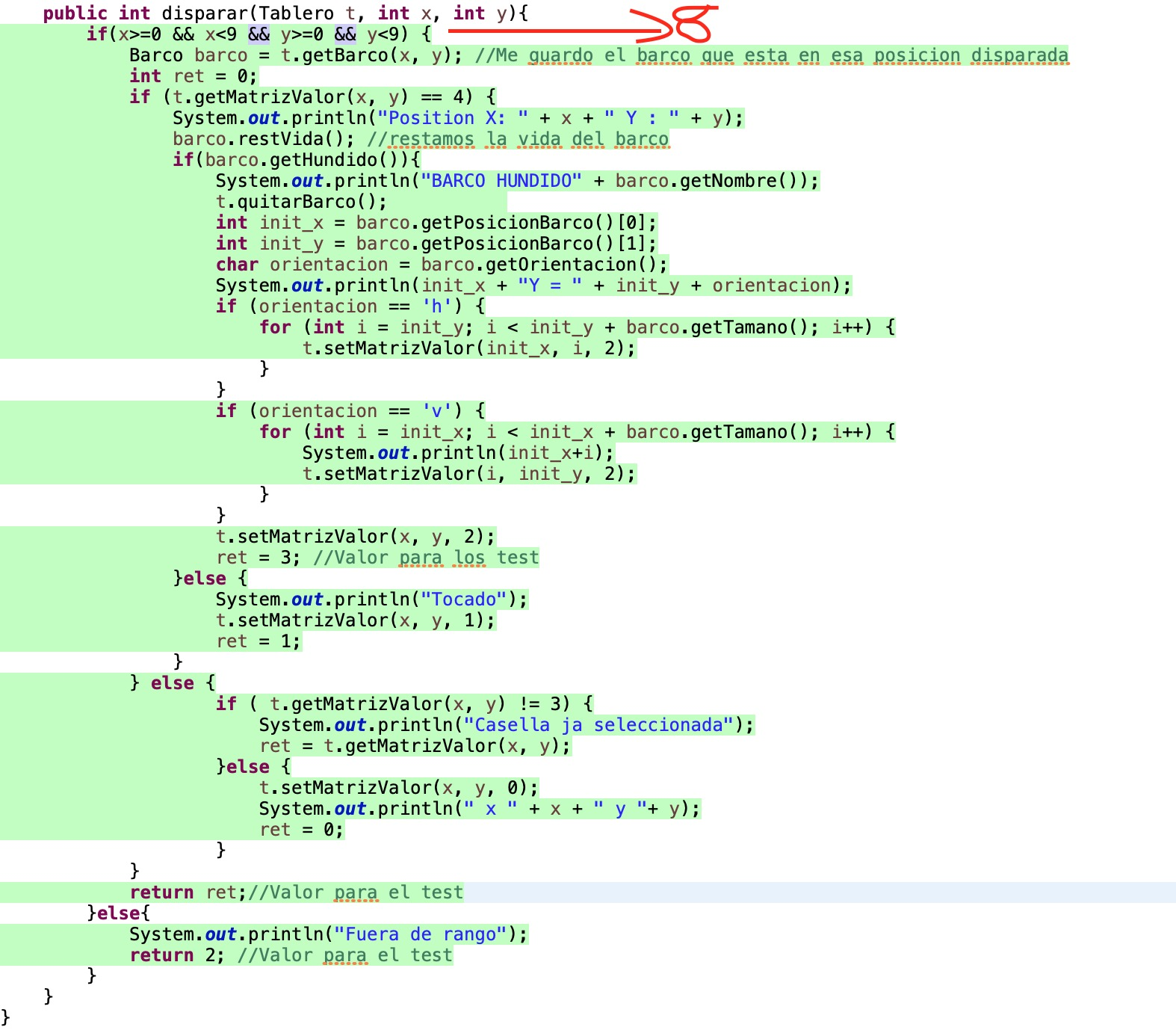
**Funcionalitat**: Condition coverage, convertido a decisión coverage ya que es simple

**Localització**: Disparar

Como el usuario puede disparar donde quiera, se deben tener en cuenta todas esas condiciones, que vienen agrupadas en el MockObject de disparar.

En la condición if (x >=0 && x <9 && y>=0 && y < 9) tenemos 8 posibles casos. Hemos realizado una combinatoria para poder obtener las posibles combinaciones, que las probamos en un mockObject,

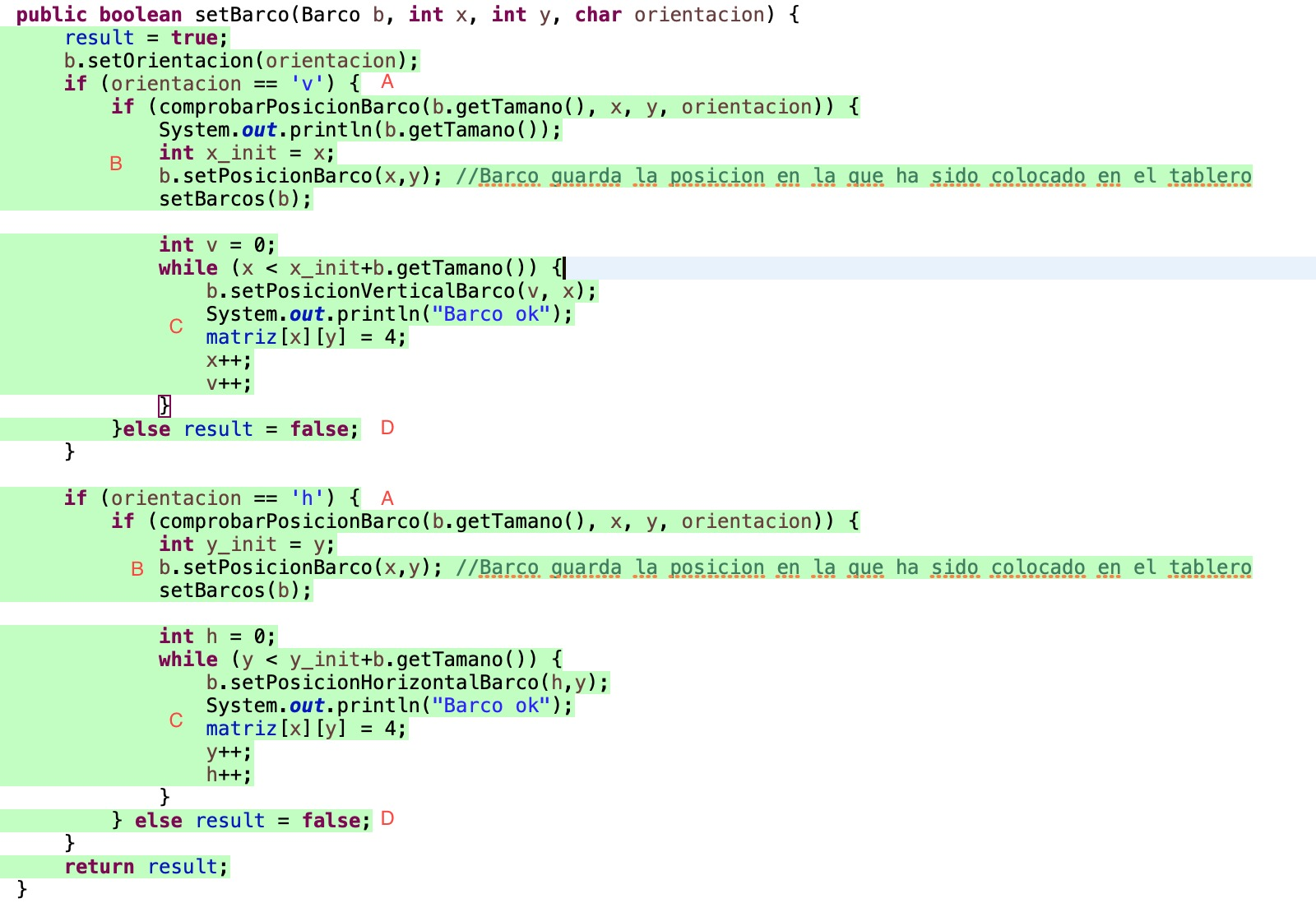


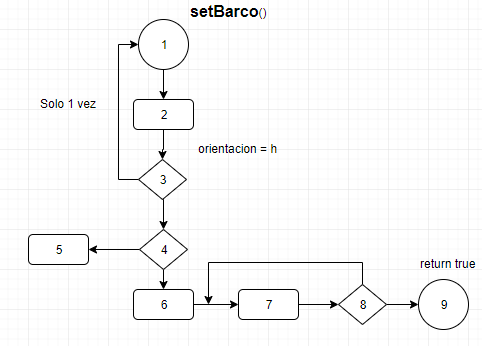


**Funcionalitat**: Path coverage

**Localització**: setbarco()

Path coverage del método setBarco, este método mira primero si es horizontal o vertical, el diagrama muestra solo 1, ya que el segundo es muy similar y se puede interpretar como repetir con orientación ‘h’ y ‘v’





V(G) = 9 - (8 - 2) = 3

Paths:

P1: 1,2,3,4,5,6,5,6,7 ; P2: 1,2,3,8; P3: 1,2,1,2,3,4,5,6,7