

Buscaminas TQS

Angel García Calleja 1490917

Daniel Calvo Ramos 1494116

Primero, para dar un poco de contexto sobre las funciones y tests que hemos hecho, vamos a explicar un poco cómo funciona el juego y la vista, ya que no había que hacer test de ésta.

Nuestro juego implementado ha sido el buscaminas. En nuestro caso hemos decidido realizar el juego utilizando una interfaz gráfica en lugar de mostrar el juego por consola.

Tenemos organizado el juego en diferentes carpetas siguiendo la arquitectura modelo-vista-controlador.

En el controlador tenemos principalmente una clase Juego que es la que se encarga del flujo de ejecución principal del juego. Esta clase es la encargada de combinar la vista y el modelo, es decir, se encarga básicamente de crear objetos de ambas clases e ir llamando a las funciones en orden cronológico para el correcto desarrollo del juego.

En la vista tenemos tanto aquellos elementos visuales como los que interactúan directamente con estos (el uso de los clicks):

Tenemos primero de todo una interfaz VistaVentanaAux, donde tenemos los mock objects declarados. Estos métodos se implementarán en vistaVentanaAuxMock.

En la vista también tenemos la clase VistaVentana, que se encarga de crear todo tipo de gráficas JFrame: ya sea para mostrar el menú de inicio de partida, el tablero con sus respectivas casillas, o las ventanas que hay que mostrar en caso de ganar/perder.

La estructura visual principal se compone de un JFrame sobre el cual tenemos un JPanel que contiene el JPanel con GridLayout del tablero el cual contiene el JLabel del Sprite de cada casilla.

Otra cosa importante que muestra es todo lo relacionado con el click en la pantalla.

Dicho de otra forma, esta clase va a pasarle al controlador el número de filas y columnas que tendrá nuestro tablero, el nivel seleccionado por el usuario, si es click derecho o izquierdo, y también en que casilla se ha clicado.

En la carpeta Modelo, tenemos el núcleo, del programa, el lugar donde se implementan todas las funcionalidades del juego y el lugar sobre el que hemos realizado los tests.

El modelo se compone de dos interfaces, necesarias para crear los dos mock objects extras estipulados para llegar al 10, una clase Casilla, en la cual tenemos la información correspondiente a una casilla y tablero, donde se tratan las casillas y donde se crean todas las funcionalidades del juego.

Problemas durante la práctica

A lo largo de la práctica hemos tenido varios problemas los cuales mencionaremos a continuación.

- Github: Al principio, no entendimos que teníamos que trabajar los dos sobre el mismo repositorio, así que por eso solo hay commits de un integrante en los primeros commits realizados. Desde el primer momento hemos tenido varios problemas para trabajar los dos integrantes sobre el mismo repositorio, ya que el 'invitado' al repositorio tenía problemas para hacer commits de manera remota a través de comandos. Tanto problemas relacionados con los commits como con los pulls etc. Así que se optó por probar de realizar los commits desde la propia web. Ahí, el integrante invitado podía realizar los commits, pero en el momento en el que el integrante 'dueño' del repositorio intentaba realizar un pull no tenía los cambios actualizados, y en el momento que hacía un push force, se eliminaban los commits anteriores realizados desde la web.

Es debido a esto que solo hay commits de un integrante del grupo. La alternativa que hemos adoptado ha sido realizar comentarios explicativos en los commits indicando quien ha hecho cada cosa o en su defecto si no hay comentario. En nuestro caso tampoco le damos mucha importancia a los commits, ya que siempre hemos trabajado de forma conjunta en el proyecto.

- El main del programa avanzaba de forma asíncrona: Al realizar el flujo principal del juego, tuvimos un problema en el que, una vez introducidos los datos por pantalla en la ventana previa a cargar el tablero, se ejecutaban las siguientes líneas del main sin esperar a que se terminara de inicializar el tablero, lo que provocaba que al ejecutar una función del tablero, diera el error de que tablero era null. La solución fue añadir tiempos de espera de microsegundos simplemente para que la ejecución del programa se desarrollara de forma correcta.

- MouseListener no detectaba sprites: El problema que tuvimos aquí fue que el mouse registraba clicks de forma correcta hasta que añadíamos sprites al JFrame. Esto era debido a que asignábamos el mouse listener al JFrame. La solución fue poner el mouse listener al JPanel en vez de al JFrame.

- Problema al convertir de coordenadas a fila/columna seleccionada: Teníamos un pequeño problema que consistía en que nosotros hacíamos click en una determinada casilla, pero se abría la casilla adyacente. Esto era porque no teníamos en cuenta el margen superior y lateral que de por sí tiene la ventana del JFrame. La solución fue tener en cuenta dichos márgenes a la hora de repartir el tamaño de las casillas.

A partir de aquí comenzaremos a usar la plantilla para hablar sobre las 2 clases que tenemos: Casilla y Tablero.

CASILLA

Comenzaremos hablando de Casilla para que se entienda mejor después el Tablero, ya que es el que maneja el tablero.

Funcionalidad: Constructor Casilla. Función que simplemente crea una Casilla con sus atributos inicializados a unos ciertos valores.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Casilla.java, clase: Casilla, método: Casilla()

Test: Archivo: src/test/java/PRAC/TQS/CasillaTest.java, clase: CasillaTest, método: testConstructor()

Test realizados: TDD.

Simplemente comprobamos que se ha creado una Casilla con todos sus atributos inicializados a sus correspondientes valores.

Funcionalidad: Constructor por parámetros de Casilla. Función que crea una Casilla dentro de una Tablero ya existente. Inicializa sus atributos a los valores correspondientes.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Casilla.java, clase: Casilla, método: Casilla(int fila, int columna, int h, int w, int filas, int columnas)

Test: Archivo: src/test/java/PRAC/TQS/CasillaTest.java, clase: CasillaTest, método: testConstructorPar()

Test realizados: TDD, caja negra. Técnicas utilizadas: particiones equivalentes y valores límite y frontera.

Comprobamos que si creando una Casilla correctamente (valores dentro sus particiones equivalentes válidas, es decir, filas entre 3-20, etc) asigne esos valores a los atributos.

Si la creamos con algún valor fuera de su partición válida, creamos una Casilla con unos valores predeterminados.

Funcionalidad: Modificar casilla. Función que modifica los atributos de Casilla.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Casilla.java, clase: Casilla, método: modCasilla(int[] datos)

Test: Archivo: src/test/java/PRAC/TQS/CasillaTest.java, clase: CasillaTest, método: testmodCasilla()

Test realizados: TDD, caja negra. Técnicas utilizadas: Mock Object, particiones equivalentes y valores límite y frontera.

Comprobamos que si modificamos una Casilla correctamente con unos valores que nos llegan del Tablero (cada valor dentro de su partición equivalente válida) éstos se asignen correctamente. Si la intentamos modificar con algún valor fuera de su partición válida, modificamos la Casilla con unos valores predeterminados.

Funcionalidad: Comprobar si dos casillas son iguales. Función que comprueba si los atributos de 2 casillas son los mismos, devolviendo true/false. Este método se utiliza más de ayuda para realizar tests.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Casilla.java, clase: Casilla, método: equals(Object anObject)

Test: Archivo: src/test/java/PRAC/TQS/CasillaTest.java, clase: CasillaTest, método: testCasillaEquals()

Test realizados: TDD.

Comprobamos que si tenemos 2 casillas iguales nos devuelva true.

Funcionalidad: Asignar bomba. Función que asigna una bomba a una casilla.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Casilla.java, clase: Casilla, método: setBomba().

Test: Archivo: src/test/java/PRAC/TQS/CasillaTest.java, clase: CasillaTest, método: testsetBomba()

Test realizados: TDD.

Simplemente comprobamos que el atributo bomba pase de false a true.

Funcionalidad: Modificar estado. Función que modifica el estado de una casilla.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Casilla.java, clase: Casilla, método: setEstado(int x)

Test: Archivo: src/test/java/PRAC/TQS/CasillaTest.java, clase: CasillaTest, método: testsetEstado()

Test realizados: TDD.

Comprobamos que el atributo estado tome sus 3 valores posibles: 0=cerrado, 1=abierto y 2=bandera.

Funcionalidad: Modificar número de vecinos. Función que modifica el atributo vecinos de una casilla. Éste se usa para indicar el número de bombas que tiene alrededor. Este método solo lo asigna, se calculará en la clase Tablero más adelante.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Casilla.java, clase: Casilla, método: setVecinos(int v)

Test: Archivo: src/test/java/PRAC/TQS/CasillaTest.java, clase: CasillaTest, método: testsetVecinos()

Test realizados: TDD.

Comprobamos que el atributo vecinos tome el valor que le pasamos.

Funcionalidad: Asignar como primera. Función que modifica el atributo primera de una Casilla. Utilizamos esta función porque la primera casilla que abres en un Buscaminas es "especial", ya que a partir de esa casilla repartes las bombas. Esta casilla será especial, no puede caer una bomba ahí.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Casilla.java, clase: Casilla, método: setPrimera()

Test: Archivo: src/test/java/PRAC/TQS/CasillaTest.java, clase: CasillaTest, método: testsetPrimera()

Test realizados: TDD.

Comprobamos que el atributo primera tome el valor true.

TABLERO

Una vez explicado cómo funcionan las casillas, veamos cómo las administra el Tablero.

Funcionalidad: Constructor Tablero. Función que simplemente crea un Tablero con sus atributos inicializados a unos ciertos valores.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: Tablero(int filas, int columnas, int alto, int ancho, int nivel)

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest, método: testTablero()

Test realizados: TDD.

Simplemente comprobamos que se ha creado un Tablero con todos sus atributos inicializados a sus correspondientes valores (0).

Funcionalidad: Constructor por parámetros de Tablero. Función que crea un Tablero con los atributos correspondientes.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: Tablero()

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest, método: testTableroPar()

Test realizados: TDD, caja negra. Técnicas utilizadas: particiones equivalentes y valores límite y frontera.

Comprobamos que si se crea un Tablero correctamente (valores dentro sus particiones equivalentes válidas, es decir, filas entre 3-20, nivel entre 1-3, etc) se asignen esos valores a los atributos correspondientes.

Si lo creamos con algún valor fuera de su partición válida, creamos un Tablero con unos valores predeterminados: 3x3 de 300x300px de nivel 1.

Funcionalidad: Modificar Tablero. Función que modifica los atributos de Tablero. Esta función la utilizamos mucho en los test para básicamente reiniciar el tablero con otros atributos, por ejemplo, cambiándole el nivel o las filas/columnas. Como si fuera otro "constructor".

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: modTablero(int[] datos)

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest, método: testmodTablero()

Test realizados: TDD, caja negra. Técnicas utilizadas: Mock Objects, particiones equivalentes y valores límite y frontera.

Comprobamos que si se modifica un Tablero correctamente (valores dentro sus particiones equivalentes válidas, es decir, filas entre 3-20, nivel entre 1-3, etc) se asignen esos valores a los atributos correspondientes que obtenemos.

Si lo modificamos con algún valor fuera de su partición válida, creamos un Tablero con unos valores predeterminados: 3x3 de 300x300px de nivel 1.

Funcionalidad: Crear Tablero. Función que crea las casillas que conforman el Tablero.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: crearTablero()

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest, método: testcrearTablero()

Test realizados: TDD.

Comprobamos que se han creado las Casillas bien igualando una de ellas a una que creamos nosotros con unos ciertos valores.

Funcionalidad: Acceder a una Casilla. Función que accede a una Casilla, desde la cual podemos acceder a sus métodos. Esta función es muy importante ya que para todas las comprobaciones necesitamos valores de Casilla.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: getCasilla(int fila, int columna)

No hay test de los métodos set/get, pero tenemos que mencionar que si hacemos getCasilla(x,y) de una Casilla que no está en el Tablero(fuera de bordes), como por ejemplo getCasilla(-1,2) o getCasilla(filas+1,columnas+1), esta función nos devolverá la casilla[0][0].

Funcionalidad: Función que calcula el número de bombas que va a tener el tablero a partir del nivel, y las dimensiones del tablero pasadas por teclado. Según el nivel seleccionado se asignará un porcentaje de bombas diferente.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: calculaNumBombas()

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest. Test realizados: TDD, caja negra, caja blanca. Técnicas utilizadas: Mock Object, particiones equivalentes, valores límite y frontera, decision coverage, condition coverage, path coverage.

Test1: método: testCalculaNumBombas . Test realizados: TDD, caja negra. Técnicas utilizadas: Mock Object, particiones equivalentes, valores límite y frontera.

Para cada partición equivalente comprobamos que el número de bombas de los valores límite y frontera coincidan con el esperado. Por ejemplo, si introduces un nivel negativo, estarás en la partición equivalente que trata valores menores que 1.

Test2: método testDecisionCoveragecalculaNumBombas. Test realizados: caja blanca. Técnicas utilizadas: Decision Coverage, mock object.

Comprobamos que las 3 decisiones que tiene esta función tomen los valores true y false. También hemos realizado su correspondiente statement coverage. (FOTO 4)

Test3: método testConditionCoveragecalculaNumBombas. Test realizados: caja blanca. Técnicas utilizadas: Condition Coverage, mock object.

Comprobamos que todas las condiciones que forman la expresión lógica de las decisiones tengan los valores true y false. Este test es muy parecido al anterior, salvo por el hecho de que tenemos una decisión con dos condiciones (comprobamos nivel>3 y nivel<1). También hemos realizado su correspondiente statement coverage (FOTO 7)

Test4: método testPathCoveragecalculaNumBombas. Test realizados: caja blanca. Técnicas utilizadas: Path Coverage, mock object.

Hemos testeado los posibles caminos del método. Para hacerlo hemos puesto prints de letras en cada uno de los estados posibles del método. Para obtener el número de paths posibles hemos recurrido a la fórmula de la complejidad ciclomática: En nuestro

caso ha sido: $E=10-8+2$. Y esos 4 caminos han sido: A->B->H // A->C->D->H // A->C->E->F->H // A->C->E->G->H (FOTO 13). También hemos realizado un diagrama de flujo de dicho método (FOTO 11) y su correspondiente statement coverage (FOTO 12)

Funcionalidad: Colocar de forma aleatoria las bombas obtenidas mediante la función calculaNumBombas en todo el tablero, siempre evitando colocar bomba en aquellas casillas que ya tienen una o en aquella casilla que haya sido el primer click del usuario.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: repartirBombas().

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest. Test realizados: TDD, caja blanca. Técnicas utilizadas: Loop Testing, mock object

Test1: método: testrepartirBombas. Test realizados: TDD. Técnicas utilizadas: mock object.

Al ser una función que trabaja con valores aleatorios, no hemos realizado test de valores límite y frontera. Para ello hemos creado una función alternativa que presentaremos más adelante llamada repartirBombasManual. En esta función testeamos que se hayan repartido el número de bombas correcto, es decir si calculaNumBombas nos da 6 comprobamos que tras realizar los aleatorios, se han colocado exactamente 6 bombas en el tablero.

Test2: método testLoopRepartirBombas Test realizados: caja blanca. Técnicas utilizadas: Loop Testing.

Para poder realizar esta técnica hemos añadido un pequeño if en la función desarrollada. Para testear el bucle correctamente realizamos las siguientes particiones: evitar loop, 1 pasada, 2 pasadas, M pasadas y N-1 pasadas. Para realizar esto igualamos la variable que se incrementa en el for a una variable seteada por nosotros. Si queremos que el bucle se ejecute 1 vez pues seteamos la función bombasAColocar a 1 y de esta forma solo realizaremos la última iteración del bucle. Al tratarse de una prueba de caja blanca, hemos realizado su equivalente statement coverage. (FOTO 14)

Funcionalidad: Esta función pone bombas en las casillas especificadas en el array de coordenadas, el cual es pasado por parámetro y contiene diferentes posiciones del tablero. El número de bombas a colocar vendrá dado por el 2do parámetro de la función. En caso de que se pidan colocar más bombas de las que contiene el array, simplemente se pondrán todas.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: repartirBombasManual(int [][] coords, int total)

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest. Test realizados: TDD, caja negra, caja blanca. Técnicas utilizadas: loop testing, mock object.

Test1: método: testrepartirBombasManual. Técnicas utilizadas: TDD, caja negra. Test realizados: particiones equivalentes, mock object.

La forma de hacer las particiones equivalentes y los valores límite y frontera siguen el esquema completo (FOTO 1). Primero se realizan las pruebas con las filas. Para ello creamos un array de arrays y ponemos bomba en aquellas casillas que queremos testear y llamamos a nuestra función desarrollada. Posteriormente recorreremos todo el tablero y comprobamos si en las casillas donde habíamos puesto bomba realmente la contienen, en caso positivo incrementamos el contador. Finalmente comprobamos que el contador sea igual al número de bombas insertadas.

Realizamos el mismo proceso para las columnas y para unos valores interiores al tablero (aleatorios).

Test2: método: testLoopRepartirBombasManual. Técnicas utilizadas: caja blanca. Test realizados: loop testing.

Para realizar este test realizamos los mismos procedimientos que utilizamos para el test testLoopRepartirBombas: Para testear el bucle correctamente realizamos las siguientes particiones: evitar loop, 1 pasada, 2 pasadas, M pasadas y N-1 pasadas. Para realizar esto vamos variando el número de bombas que queremos que se coloquen. De esta forma vamos pasando por todos los estados mencionados anteriormente. Al ser un loop testing, mostramos también su respectivo statement coverage(FOTO 15).

Funcionalidad: Método que coloca 1 bomba (pone setBomba a true) en la posición pasada por parámetro. Esta función se utiliza principalmente para realizar test (en muchas ocasiones queremos insertar una bomba en un determinado lugar) para probar otros métodos.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: colocar1bomba(int x, int y).

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest. Test realizados: TDD,caja blanca. Técnicas utilizadas: Loop Testing, mock object

Test1: método: testcolocar1bomba. Test realizados: TDD. Técnicas utilizadas:mock object.

Para realizar este test usamos 2 mock objects, uno para simular un tablero y otro para simular una casilla. Posteriormente comprobamos que dicha casilla esté vacía (no tiene bomba) y tras llamar al método colocar1bomba en dicha casilla contenga una bomba.

Test2: método: testDecisionCoveragecolocar1bomba. Test realizados: caja blanca. Técnicas utilizadas: mock object, decision coverage

. En este caso, como únicamente tenemos una decisión, únicamente tendremos que testear una rama. (FOTO 3)

Test3: método: testConditionCoveragecolocar1bomba. Test realizados: caja blanca. Técnicas utilizadas: mock object, condition coverage.

Esta función será igual a la anterior ya que la decisión está formada por una única condición, por lo que el testeo será el mismo. (FOTO 6)

Funcionalidad: Marcar Casilla. Función que pone una bandera o quita una bandera dependiendo del estado de la casilla. Si no tiene bandera la pone y si ya la tiene la quita. Al final hacemos actualizarCasilla() para cambiar el sprite de ésta.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: marcarCasilla(int x, int y)

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest.

Test realizados: TDD, caja negra, caja blanca. Técnicas utilizadas: Mock Objects, particiones equivalentes, valores límite y frontera, decision coverage, condition coverage, path coverage

Test1: método: testmarcarCasilla()

Test realizados: TDD, caja negra. Técnicas utilizadas: Mock Objects, particiones equivalentes y valores límite y frontera.

Comprobamos que las casillas se marcan correctamente.

Hemos contemplado el rango [0,NUM_FILAS-1]. De aquí nos salen las siguientes particiones equivalentes: $x < 0$ (inválido), $0 \leq x \leq \text{num_filas}-1$ (válidos ya que existen en el Tablero) y $x > \text{num_filas}-1$ (inválido). Las 2 particiones inválidas las usamos para

“mostrar” que te estás saliendo de los límites del Tablero. No existen la Casilla[-1][0] y no existe la Casilla[5][0] en un Tablero 5x5.

En base a esto y a un Tablero 5x5, contemplamos los siguientes valores de número de fila a marcar:

- * Valores interiores: 2
- * Valores Frontera: 0,4 --> (num_filas-1)
- * Valores interior Frontera: 1,3
- * Valores exterior Frontera: -1,5

Por tanto, marcaremos y comprobaremos que se marque las siguientes casillas:

[-1][0], [0][0], [1][0], [2][0], [3][0], [4][0], [5][0]. Para los casos inválidos, se marcará la Casilla [0][0], como ya hemos explicado en getCasilla().

Para las columnas (p.e casilla [0][2]), seguimos el mismo patrón.

Y como extra también haremos estas comprobaciones en la casilla de la esquina abajo derecha [4][4] y una casilla por el centro [2][2].

Este esquema de comprobaciones lo usaremos para más métodos, como por ejemplo abrirCasilla(). Lo llamaremos EsquemaCompleto. (FOTO 1)

Test2: método: testDecisionCoverageMarcarCasilla()

Test realizados: caja blanca. Técnicas utilizadas: Decision Coverage, mock object

Comprobamos que las 2 decisiones que tiene esta función tomen los valores true y false. También hemos realizado su correspondiente statement coverage. (FOTO 2)

Test3: método: testConditionCoverageMarcarCasilla()

Test realizados: caja blanca. Técnicas utilizadas: Condition Coverage, mock object.

Este test es igual que el anterior ya que en este método no hay una condición con 2 expresiones lógicas. También hemos realizado su correspondiente statement coverage (FOTO 3)

Test4: método testPathCoverageMarcarCasilla. Test realizados: caja blanca, caja negra. Técnicas utilizadas: mock object, path coverage.

Este test consiste en recorrer todos los posibles paths de la ejecución. Para hacerlo hemos puesto prints de letras en cada uno de los estados posibles del método. Para obtener el número de paths posibles se ha utilizado la fórmula de la complejidad ciclomática: En el diagrama de flujo de la función hemos obtenido: $E=6-5+2$. Y los 3 caminos resultantes son: A->B->E // A->C->D->E // A->C->E (FOTO 10). También hemos realizado un diagrama de flujo de dicho método (FOTO 8) y su correspondiente statement coverage (FOTO 9).

Funcionalidad: Abrir Casilla. Función recursiva que abre una casilla, es decir, le cambia su estado a ABIERTO. Si vemos que la Casilla no tiene bombas alrededor (vecinos=0), abriremos las 8 casillas de su alrededor llamando a abrirAlrededor(). Si abrimos una casilla con bomba, lo marcamos poniendo el atributo explosión en true. Al final hacemos actualizarCasilla() para cambiar el sprite de ésta.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: abrirCasilla(int x, int y)

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest, método: testabrirCasilla()

Test realizados: TDD, caja negra. Técnicas utilizadas: Mock Objects, particiones equivalentes y valores límite y frontera.

Para este test seguimos el EsquemaCompleto y comprobamos que las casillas se abran correctamente. Hemos modificado el tablero de cierta manera para que no se abra ninguna casilla alrededor, ya que eso se probará en el siguiente test.

Funcionalidad: Abrir alrededor. Función que es llamada por abrirCasilla() si la casilla en cuestión no tiene ninguna bomba alrededor. Sirve para abrir sus 8 (o menos) casillas alrededor.

Comprobamos los casos en los que estamos en esquinas, bordes o por el centro para abrir o no abrir ciertas casillas (no pasarse de los límites del tablero).

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: abrirAlrededor(Casilla c)

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest, método: testabrirAlrededor()

Test realizados: TDD, caja negra. Técnicas utilizadas: Mock Objects, particiones equivalentes y valores límite y frontera.

Para este test seguimos el EsquemaCompleto y comprobamos que las casillas se abran correctamente. En este caso hemos modificado el tablero y asignado ciertas bombas con repartirBombasManual para que se nos abran algunas casillas alrededor, dependiendo de la zona donde esté esa casilla, por ejemplo, una casilla en una esquina solo abre 3.

Tuvimos problemas para hacer este test ya que primero, si no pones bombas a la que abras una se abrirá todo el tablero y segundo algunas casillas se quedaban abiertas y nos salía mal el test, por ejemplo, la casilla [0][0] abre [0][1] y luego otras comprobaciones daban mal porque esa ya estaba abierta. Decidimos que después de cada comprobación simplemente asignamos el estado de las casillas que se abrían a 0, para ponerlas “cerradas” otra vez.

Funcionalidad: Insertar jugada. Función que dependiendo de la jugada que le pasamos hace una cosa u otra. Jugada es un array de 3 posiciones donde [0] y [1] son filas y columnas donde hacer la [2] acción. La acción viene del mouseListener, 0 si click izquierdo y 1 si click derecho.

Realmente engloba abrir y marcar casilla, haciendo abrirCasilla() si recibe jugada[2]=0 (click izq.) y marcarCasilla() si jugada[2]=1 (click derecho).

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: insertarJugada(int[] jugada)

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest, método: testinsertarJugada()

Test realizados: TDD, caja negra. Técnicas utilizadas: Mock Objects, particiones equivalentes y valores límite y frontera.

Realmente lo comprobamos haciendo el mismo test que el abrirCasilla(), ya que no ofrece ninguna funcionalidad nueva.

Funcionalidad: Comprobar victoria. Función que comprueba que queden cerradas el mismo número de bombas tiene y el tablero y que las casillas que quedan cerradas correspondan a esas donde hay bomba.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: checkWin()

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest Test realizados: TDD, caja negra, caja blanca. Técnicas utilizadas: Mock Objects y particiones equivalentes, loop testing

Test1: método: `testcheckWin()`. Test realizados: TDD, caja negra. Técnicas utilizadas: Mock Objects y particiones equivalentes.

Modificamos un Tablero 3x3 donde repartimos manualmente 7 bombas. Comprobamos el caso de no ganar, abriendo 1 casilla donde no hay bomba, y después comprobamos que se gane si abrimos la última casilla sin bomba que hay.

Test2: método: `testLoopCheckWin`. Test realizados: caja blanca. Técnicas utilizadas: loop testing.

Para poder testear el bucle anidado, tratamos el doble bucle `for` como si fueran dos bucles simples: primero realizamos los tests en el bucle interior, manteniendo el bucle exterior constante, y posteriormente realizamos el procedimiento a la inversa; mantenemos estable el bucle interior y realizamos los tests con el bucle exterior. Al haber realizado una técnica de loop testing, también mostramos su respectivo statement coverage (FOTO 17)

Funcionalidad: Comprobar derrota. Función que simplemente devuelve el atributo explosión, si es true es que has abierto una casilla con bomba.

Localización: Archivo: `src/main/java/PRAC/TQS/Modelo/Tablero.java`, clase: `Tablero`, método: `checkLose()`

Test: Archivo: `src/test/java/PRAC/TQS/TableroTest.java`, clase: `TableroTest`, método: `testcheckLose()`

Test realizados: TDD, caja negra. Técnicas utilizadas: Mock Objects y particiones equivalentes.

Modificamos un Tablero 3x3 donde repartimos manualmente 2 bombas. Comprobamos que no se pierde si abrimos una casilla sin bomba y después que sí porque abrimos una casilla con bomba.

Funcionalidad: Descubrir todo el Tablero. Función que simplemente pone todas las casillas con estado=abierto. Esta función solo la usamos en el main para que si pierdes/ganas se revele todo el tablero.

Localización: Archivo: `src/main/java/PRAC/TQS/Modelo/Tablero.java`, clase: `Tablero`, método: `descubrirTablero()`

Test: Archivo: `src/test/java/PRAC/TQS/TableroTest.java`, clase: `TableroTest`. Test realizados: TDD, caja blanca. Técnicas utilizadas: Mock Objects, Loop Testing.

Test1: método: `testdescubrirTablero`. Test realizados: TDD. Técnicas utilizadas: Mock Objects.

Simplemente comprobamos que en un principio todas las casillas estén cerradas, luego al descubrirlas estén todas abiertas.

Test2: método: `testLoopDescubrirTablero`. Test realizados: caja blanca. Técnicas utilizadas: Loop testing.

Para poder testear el bucle anidado, tratamos el doble bucle `for` como si fueran dos bucles simples: primero realizamos los tests en el bucle interior, manteniendo el bucle exterior constante, y posteriormente realizamos el procedimiento a la inversa; mantenemos estable el bucle interior y realizamos los tests con el bucle exterior. Al haber realizado una técnica de loop testing, también mostramos su respectivo statement coverage (FOTO 16).

Funcionalidad: Esta función se encarga de calcular el número de bombas que tiene alrededor una casilla pasada por parámetro. Para que este método funcione correctamente es importante tener en cuenta los 3 posibles 'lugares' donde puede estar una casilla: en una esquina, en un borde o sin bordes.

Localización: Archivo: src/main/java/PRAC/TQS/Modelo/Tablero.java, clase: Tablero, método: getNumVecinos(Casilla c).

Test: Archivo: src/test/java/PRAC/TQS/TableroTest.java, clase: TableroTest método: testNumVecinos Test realizados: TDD, caja negra. Técnicas utilizadas: particiones equivalentes, valores límite y frontera, mock object.

Para este test seguimos el Esquema Completo: realizamos particiones equivalentes tanto para las filas como para las columnas, y para cada eje, comprobamos los vecinos de los valores frontera, límites exteriores a la frontera, valores interiores a la frontera, y valores interiores (extras).

Para acabar, hemos hecho el statement coverage de todo el código.

Si lo hacemos para los test, podemos observar que sale un 100% (FOTO 18).

Si lo hacemos para el main, nos sale un 76% (FOTO 19).

Esto es debido a que, si nosotros jugamos normal, hay algunas condiciones en las que nunca entrarás, por ejemplo, si ganas se mostrará la pantalla de victoria, lo que implica que NO se mostrará la de derrota por lo tanto esas líneas no se harán.

También algunos de nuestros métodos tanto en Casilla como en Tablero los usamos únicamente para test, no se ejecutarán en una partida normal.

También gran parte de métodos get/set en Casilla y en Tablero no se ejecutan. Estos métodos tienen sentido que existan, aunque luego no se utilicen.

También ya que el programa tiende a crear el Tablero correctamente, las líneas de comprobar si se crean Tableros y Casillas mal y asignar unos ciertos valores tampoco son ejecutadas.

COMPROBACIONES REALIZADAS

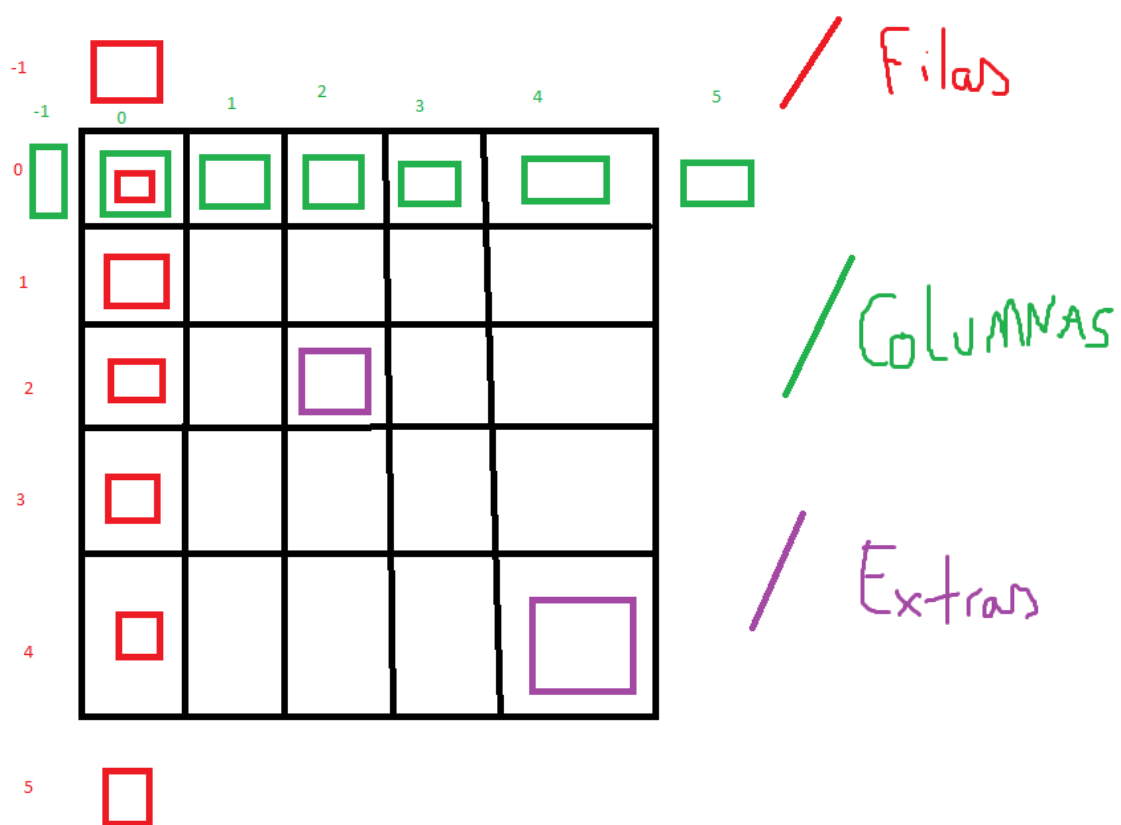


Foto 1 EsquemaCompleto

```
public void testDecisionCoverageMarcarCasilla() {
    VistaVentanaAuxMock mockVentana=new VistaVentanaAuxMock();
    Tablero tabrirAl=new Tablero();
    tabrirAl.setVentana(mockVentana);
    int[] dat=mockVentana.pasarDatos();
    dat[0]=2;
    dat[1]=2;
    dat[4]=1;
    tabrirAl.modTablero(dat);

    int [][] cas_sel= {{0,1},{1,0},{1,1}};
    tabrirAl.repartirBombasManual(cas_sel);

    tabrirAl.marcarCasilla(0, 0);

    tabrirAl.marcarCasilla(0, 0);

    tabrirAl.getCasilla(0, 0).setEstado(1);
    tabrirAl.marcarCasilla(0, 0);
}
```

Foto 2 Decision coverage marcarCasilla()

```

public void testDecisionCoveragecolocar1bomba() {
    CasillaAuxMock mockCasilla=new CasillaAuxMock();
    VistaVentanaAuxMock mockVentana=new VistaVentanaAuxMock();

    Tablero tmock=new Tablero();
    tmock.setCasilla(mockCasilla);
    tmock.setVentana(mockVentana);

    int[] dat=mockVentana.pasarDatos();
    dat[0]=5;
    dat[1]=5;
    dat[4]=1;
    tmock.modTablero(dat);

    int []coords=mockCasilla.pasarCoordenadas();

    int bombas=tmock.getNBombas();
    //PONER BOMBA DONDE NO HABIA ANTES --> se anade 1 bomba
    tmock.colocar1bomba(coords[0], coords[1]);

    //PONER BOMBA DONDE SI HABIA ANTES --> no se anade 1 bomba, no bombas+2
    tmock.colocar1bomba(coords[0], coords[1]);
}

```

Foto 3 Decision coverage colocar1bomba()

```

public void testDecisionCoveragecalculaNumbombas() {
    VistaVentanaAuxMock resultado=new VistaVentanaAuxMock();
    Tablero t5=new Tablero();
    t5.setVentana(resultado);
    int[] dat=resultado.pasarDatos();
    int expected1=8;
    dat[0]=8;
    dat[1]=8;
    dat[4]=1;
    t5.modTablero(dat);

    t5.setNivel(4);
    t5.calculaNumbombas();

    t5.setNivel(1);
    t5.calculaNumbombas();

    t5.setNivel(2);
    t5.calculaNumbombas();

    t5.setNivel(3);
    t5.calculaNumbombas();
}

```

Foto 4 Decision coverage calculaNumbombas()

```

public void testConditionCoverageMarcarCasilla() {
    VistaVentanaAuxMock mockVentana=new VistaVentanaAuxMock();
    Tablero tabrirAl=new Tablero();
    tabrirAl.setVentana(mockVentana);
    int[] dat=mockVentana.pasarDatos();
    dat[0]=2;
    dat[1]=2;
    dat[4]=1;
    tabrirAl.modTablero(dat);

    int [][] cas_sel= {{0,1},{1,0},{1,1}};
    tabrirAl.repartirBombasManual(cas_sel);

    tabrirAl.marcarCasilla(0, 0);

    tabrirAl.marcarCasilla(0, 0);

    tabrirAl.getCasilla(0, 0).setEstado(1);

    tabrirAl.marcarCasilla(0, 0);

}

```

Foto 5 Condition coverage marcarCasilla()

```

public void testConditionCoveragecolocar1bomba() {
    CasillaAuxMock mockCasilla=new CasillaAuxMock();
    VistaVentanaAuxMock mockVentana=new VistaVentanaAuxMock();

    Tablero tmock=new Tablero();
    tmock.setCasilla(mockCasilla);
    tmock.setVentana(mockVentana);

    int[] dat=mockVentana.pasarDatos();
    dat[0]=5;
    dat[1]=5;
    dat[4]=1;
    tmock.modTablero(dat);

    int []coords=mockCasilla.pasarCoordenadas();

    int bombas=tmock.getNBombas();
    //Condicion if se cumple --> se anade 1 bomba
    tmock.colocar1bomba(coords[0], coords[1]);

    //Condicion if no se cumple--> no se anade 1 bomba, no bombas+2
    tmock.colocar1bomba(coords[0], coords[1]);

}

```

Foto 6 Condition coverage colocar1bomba()

```

public void testConditionCoveragecalculaNumBombas() {
    VistaVentanaAuxMock resultado=new VistaVentanaAuxMock();
    Tablero t5=new Tablero();
    t5.setVentana(resultado);

    int[] dat=resultado.pasarDatos();
    int expected1=8;
    dat[0]=8;
    dat[1]=8;
    dat[4]=1;
    t5.modTablero(dat);

    t5.setNivel(0);
    t5.calculaNumBombas();

    t5.setNivel(4);
    t5.calculaNumBombas();

    t5.setNivel(1);
    t5.calculaNumBombas();

    t5.setNivel(2);
    t5.calculaNumBombas();

    t5.setNivel(3);
    t5.calculaNumBombas();
}

```

Foto 7 Condition coverage calculaNumBombas()

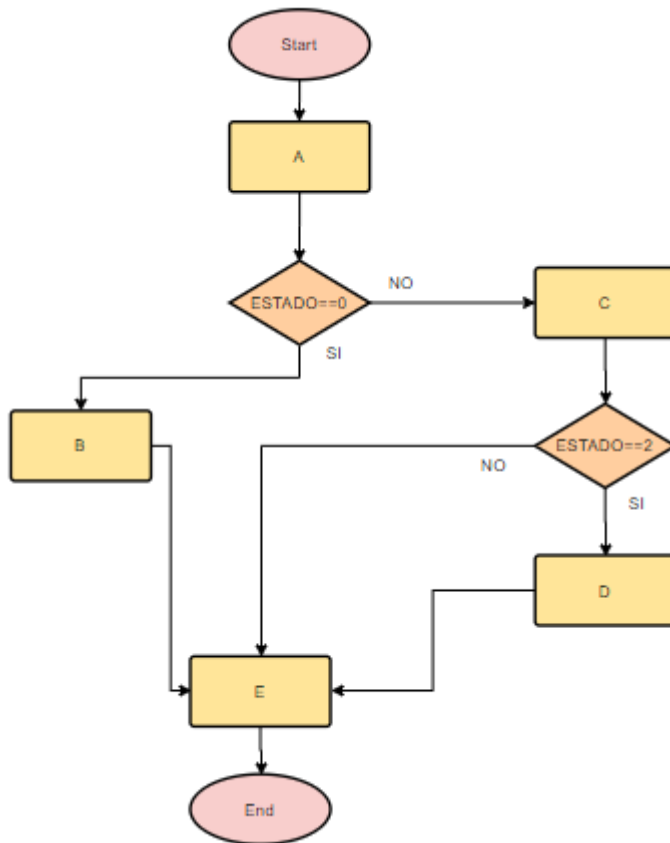


Foto 8 Diagrama flujo marcarCasilla()

```

public void testPathCoverageMarcarCasilla() {
    VistaVentanaAuxMock mockVentana=new VistaVentanaAuxMock();
    Tablero tabrirAl=new Tablero();
    tabrirAl.setVentana(mockVentana);
    int[] dat=mockVentana.pasarDatos();
    dat[0]=2;
    dat[1]=2;
    dat[4]=1;
    tabrirAl.modTablero(dat);

    int [][] cas_sel= {{0,1},{1,0},{1,1}};
    tabrirAl.repartirBombasManual(cas_sel);
    System.out.println("INCIO TEST Path COVERAGE MARCAR CASILLA")
    System.out.println("-----");
    System.out.println("Path estado=CERRADO");
    tabrirAl.marcarCasilla(0, 0);
    System.out.println("-----");
    System.out.println("Path estado=MARCADO");
    tabrirAl.marcarCasilla(0, 0);
    System.out.println("-----");
    System.out.println("Path estado=ABIERTO");
    tabrirAl.getCasilla(0, 0).setEstado(1);
    tabrirAl.marcarCasilla(0, 0);
    System.out.println("-----");
    System.out.println("FIN TEST Path COVERAGE MARCAR CASILLA");
}

```

Foto 9 Path coverage marcarCasilla()

```
INCIO TEST Path COVERAGE MARCAR CASILLA
-----
Path estado=CERRADO
A
B
E
-----
Path estado=MARCADO
A
C
D
E
-----
Path estado=ABIERTO
A
C
E
-----
FIN TEST Path COVERAGE MARCAR CASILLA
```

Foto 10 Paths resultado marcarCasilla()

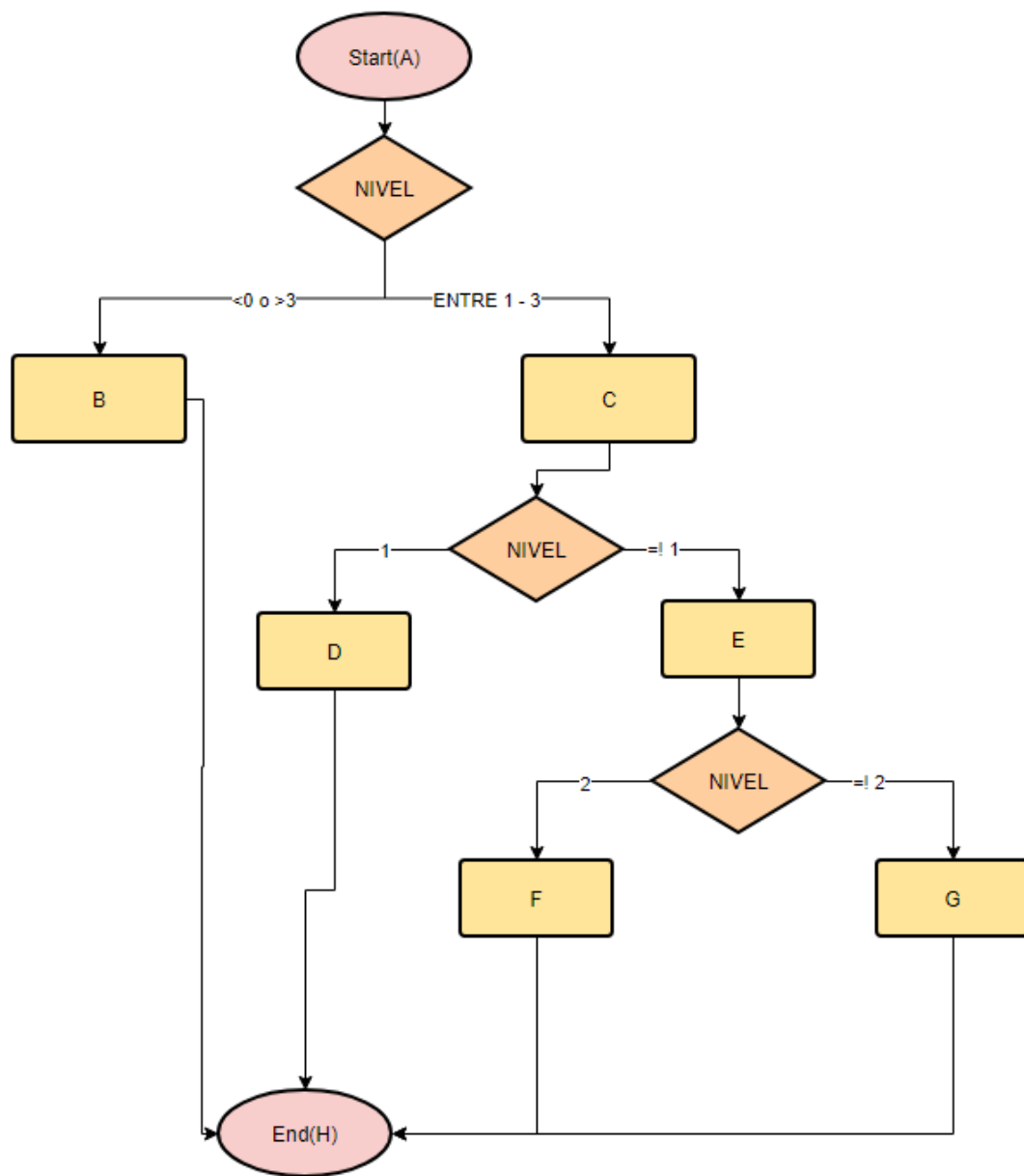


Foto 11 Diagrama flujo calculaNumBombas()

```

public void testPathCoveragecalculaNumBombas() {
    VistaVentanaAuxMock resultado=new VistaVentanaAuxMock();
    Tablero t5=new Tablero();
    t5.setVentana(resultado);
    System.out.println("INCIO TEST Path COVERAGE CALCULA NUM BOMBAS");
    int[] dat=resultado.pasarDatos();
    int expected1=8;
    dat[0]=8;
    dat[1]=8;
    dat[4]=1;
    t5.modTablero(dat);
    System.out.println("-----");
    System.out.println("Path NIVEL no se cumple");
    t5.setNivel(4);
    t5.calculaNumBombas();
    System.out.println("-----");
    System.out.println("Path NIVEL 1");
    t5.setNivel(1);
    t5.calculaNumBombas();
    System.out.println("-----");
    System.out.println("Path NIVEL 2");
    t5.setNivel(2);
    t5.calculaNumBombas();
    System.out.println("-----");
    System.out.println("Path NIVEL 3");
    t5.setNivel(3);
    t5.calculaNumBombas();
    System.out.println("-----");
    System.out.println("FIN TEST Path COVERAGE CALCULA NUM BOMBAS");
}

```

Foto 12 Path coverage calculaNumBombas()

```

INCIO TEST Path COVERAGE CALCULA NUM BOMBAS
-----
Path NIVEL no se cumple
A
B
H
-----
Path NIVEL 1
A
C
D
H
-----
Path NIVEL 2
A
C
E
F
H
-----
Path NIVEL 3
A
C
E
G
H
-----
FIN TEST Path COVERAGE CALCULA NUM BOMBAS

```

Foto 13 Paths resultados calculaNumBombas()

```

public void testLoopRepartirBombas() throws IOException {
    Tablero t = new Tablero(20,20,300,300,1);
    int btotales=t.calculaNumBombas();
    //System.out.println("INCIO TEST LOOP COVERAGE REPARTIR BOMBAS");
    //System.out.println("-----");
    //System.out.println("EVITAR LOOP");
    t.setBombasAColocar(0);
    t.repartirBombas();
    //System.out.println("-----");
    //System.out.println("1 PASADA");
    t.setBombasAColocar(1);
    t.repartirBombas();
    //System.out.println("-----");
    //System.out.println("2 PASADAS");
    t.setBombasAColocar(2);
    t.repartirBombas();
    //System.out.println("-----");
    //System.out.println("M PASADAS (5)");
    t.setBombasAColocar(5);
    t.repartirBombas();
    //System.out.println("-----");
    //System.out.println("N-1 PASADAS");
    t.setBombasAColocar(btotales-1);
    t.repartirBombas();
    //System.out.println("FIN TEST LOOP COVERAGE REPARTIR BOMBAS");
}

```

Foto 14 Loop testRepartirBombas()

```

public void testLoopRepartirBombasManual() throws IOException {
    Tablero t = new Tablero(6,6,300,300,2);
    int[][] cas_sel= {{0,0},{0,1},{0,2},{1,0},{1,1},{1,2}};

    int btotales=cas_sel.length;
    //System.out.println("INCIO TEST LOOP COVERAGE REPARTIR BOMBAS MANUAL");
    //System.out.println("-----");
    //System.out.println("EVITAR LOOP");
    t.repartirBombasManual(cas_sel,0);
    //System.out.println("-----");
    //System.out.println("1 PASADA");
    t.repartirBombasManual(cas_sel,1);
    //System.out.println("-----");
    //System.out.println("2 PASADAS");
    t.repartirBombasManual(cas_sel,2);
    //System.out.println("-----");
    //System.out.println("M PASADAS (3)");
    t.repartirBombasManual(cas_sel,3);
    //System.out.println("-----");
    //System.out.println("N-1 PASADAS");
    t.repartirBombasManual(cas_sel,btotales-1);
    //System.out.println("-----");
    //System.out.println("FIN TEST LOOP COVERAGE REPARTIR BOMBAS MANUAL");
}

```

Foto 15 Loop testrepartirBombasManual()

```
public void testLoopdescubrirTablero() throws I
    Tablero t = new Tablero(20,20,300,300,1);
    int fil,col;

    fil=20;
    col=0;
    t.descubrirTablero(fil, col);

    col=1;
    t.descubrirTablero(fil, col);

    col=2;
    t.descubrirTablero(fil, col);

    col=5;
    t.descubrirTablero(fil, col);

    col=19;
    t.descubrirTablero(fil, col);

    fil=0;
    col=20;
    t.descubrirTablero(fil, col);

    fil=1;
    t.descubrirTablero(fil, col);

    fil=2;
    t.descubrirTablero(fil, col);

    fil=5;
    t.descubrirTablero(fil, col);

    fil=19;
    t.descubrirTablero(fil, col);

}
```

Foto 16 Loop test descubrirTablero()

```

public void testLoopcheckWin() throws IOException {
    Tablero t = new Tablero(6,6,300,300,1);
    int fil,col;
    t.setNBombas(36);
    t.repartirBombas();
    boolean x=false;

    fil=6;
    col=0;
    x=t.checkWin(fil, col);

    col=1;
    x=t.checkWin(fil, col);

    col=2;
    x=t.checkWin(fil, col);

    col=3;
    x=t.checkWin(fil, col);

    col=5;
    x=t.checkWin(fil, col);

    fil=0;
    col=6;
    x=t.checkWin(fil, col);

    fil=1;
    x=t.checkWin(fil, col);

    fil=2;
    x=t.checkWin(fil, col);

    fil=3;
    x=t.checkWin(fil, col);

    fil=5;
    x=t.checkWin(fil, col);
}

```

Foto 17 Loop test checkWin()

PRAC.TQS (15 nov. 2020 1:19:21)				
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
PRAC TQS	90,3 %	9.381	1.004	10.385
src/main/java	67,9 %	2.119	1.004	3.123
src/test/java	100,0 %	7.262	0	7.262
PRAC.TQS	100,0 %	7.262	0	7.262
CasillaAuxMock.java	100,0 %	20	0	20
CasillaTest.java	100,0 %	385	0	385
TableroAuxMock.java	100,0 %	36	0	36
TableroTest.java	100,0 %	6.776	0	6.776
VistaVentanaAuxMock.java	100,0 %	45	0	45

Foto 18 Statement coverage de test al 100%












Juego (2) (15 nov. 2020 1:35:14)				
Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
▼ PRAC TQS	 22,6 %	2.343	8.024	10.367
> src/test/java	 0,0 %	0	7.262	7.262
▼ src/main/java	 75,5 %	2.343	762	3.105
▼ PRAC.TQS.Modelo	 69,1 %	1.515	678	2.193
▼ Tablero.java	 76,7 %	1.279	389	1.668
> Tablero	 76,7 %	1.279	389	1.668
> Casilla.java	 45,0 %	236	289	525
▼ PRAC.TQS.Vista	 88,2 %	547	73	620
> VistaVentana.java	 88,2 %	547	73	620
▼ PRAC.TQS.Controlador	 96,2 %	281	11	292
> Juego.java	 96,2 %	281	11	292

Foto 19 Statement coverage Juego