Roger Rey Mesa 1492221

Àlex Salvador Abad 149155

TDD Project: BattleShip

(En el nostre repositori de gitHub el projecte té com a nom Tetris, al principi vam començar amb el tetris pero degut a la complexitat i la falta de temps vam decidir canviar de joc i ja no varem poder canviar el nom del repositori).

Class Boat

1.Funcionalitat: Sobrecàrrega del constructor de la classe Boat, inicialitzem en funció de la mida del vaixell que passem per paràmetre, tots els atributs que tenim a la classe Boat com poden ser; el número de vides del vaixell que serà el mateix que la longitud, la longitud, arrays de coordenadas 'X' i 'Y' i en funció de la longitud se li assigna un nom (mida 5 "Portaavio", mida 4 "Creuer", mida 3 "Destructor" i mida 2 "Fragata").

Localització: Boat.java, class Boat i Boat(int longitud).

Test: BoaTest.java, class BoaTest i void testBoat().

En aquest test realizem proves de caixa negra i apliquem la tècnica de partició equivalent. El que fem per realitzar les proves es primer de tot, crear amb el constructor el màxim d'objectes de tipus Boat que permet una partida, és a dir 4, i els hi assignem les diferents longituds que tindran els 4 vaixells (5, 4, 3 i 2).

Després per comprovar que els resultats són els esperats, cridem el mètode *assertEquals()* i comprovem cridant la funció de la classe Boat *getNom()* que el nom assignat al nostre objecte Boat coincideix amb el resultat esperat, per exemple en el cas del vaixell de mida 5 el seu nom ha de ser "Portaavió", això ho comprovem per cada vaixell amb el seu nom corresponent.

Per acabar, comprovem que la mida assignada al objecte sigui la mateixa que la esperada per tant, cridem la funció *assertEquals()* i comprovem que el valor que ens retorna la funció *getLongitud()* és el mateix que el valor esperat, això ho comprovem per a cada vaixell.

2.Funcionalitat: Actualiza la variable num_vides (número de vides del vaixell) restant-li 1. Està pensada per quan disparan a un vaixell i l'han tocat, la seva vida es reduirà.

Localització: Boat.java, class Boat i public void Disparat().

Test: BoaTest.java, class BoaTest i public void DisparatTest().

En aquest test es realitzen proves de caixa negra i apliquem la tècnica de partició equivalent. Primer de tot creem un objecte de tipus Boat i li assignem una dimensió, aquesta dimensió sera el número de vides que tindrà. Per comprovar si el número de vides es veu reduït, cridem el mètode *disparat()* per al objecte que tenim creat i després comprovem amb un *assertEquals(b.getVides(),4)* que el número de vides inicial (5) ha estat reduït en un valor, això ho comprovem per als valors frontera(5 i 2), els valors límit interiors(4 i 3) i un valor límit

exterior (-1), en aquest últim passa el valor frontera pero no es cap problema ja que només ens interessa que arribi a 0 o ser més petit.

3.Funcionalitat: Afegim als arrays de coordenades ‘x’ i ‘y’ a la posició *aux* (variable de tipus enter que tenim per indicar la posició del array) els valors enters passats per paràmetre. Després incrementen un valor la variable *aux* que tenim declarada com a atribut privat a la classe, no la incialitzem a 0 en el mètode ja que d’aquesta manera aconseguim la mateixa funció que faria la funció *push_back()* en el cas d’una *List* i afegim els valors a la última posició dels arrays.

Localització: Boat.java, class Boat i public void afegeixCoordenada(int x, int y).

Test: BoaTest.java, class BoaTest i public void DisparatTest().

En aquest test realizem proves de caixa negra i apliquem la tècnica de partició equivalent. Primer de tot creem un objecte de tipus Boat amb una mida de les disponibles (en aquest cas la mida és 2), després cridem al mètode *afegeixCoordenada()* i li passem uns valors enters com a paràmetres, per exemple (0,0), comprovem comparant el valor que ens retorna la funció *getCoordenadaX()* i *getCoordenadaY( )* amb els valors afegits prèviament (0,0), també comparem si la variable *aux* s’ha incrementat després d’afegir les coordenades. Tornem a comprovar el mateix però amb un altre parella de valors, en aquest cas s'ha afegit a la següent posició del array tal i com volíem.

Class Taulell

1.Funcionalitat: Constructor de la classe Taulell per paràmetres, concretament dos nombres enters que representen el nombre de caselles que volem en horitzontal i vertical al nostre taulell de joc.

Inicialitzem els atributs de la classe midax i miday amb els valors que entren com a paràmetres, construim una matriu d'enters amb les dimensions "mida_x x mida_y", creem una llista de vaixells que ens servirà futurament per controlar quan moren i quan no els vaixells del nostre taulell i creem un primer string_taulell (cadena de caràcters que representen l'estat actual del taulell) sense els vaixells col·locats, que ens servirà per mostrar-lo a l'usuari i demanar que col·loqui els vaixells.

Localització: Taulell.java, class Taulell i Taulell(int mida_x, int mida_y).

Test: TaulellTest.java, class TaulellTest i void testTaulell().

Per a testejar el constructor hem emprat la tècnica caixa negra, ja que simplement comprovem mitjançant assertEquals i getters que les variables que s'han d'inicialitzar a la classe Taulell quan en creem una instància a partir d'aquest constructor s'han creat correctament. Un petit exemple d'això seria la següent sentencia de codi:

assertEquals(t.getMidaX(),midax);

2.Funcionalitat: El mètode construeixTaulell(int[][] matriu) és fonamental per al funcionament del programa, ja que funciona com a "template" per imprimir el taulell a l'usuari. Bàsicament el que fa el mètode és recòrrer dos *loops* que generen l'string final afegint el contingut de la matriu de control del taulell traduit als caràcters que enten l'usuari.

Localització: Taulell.java, class Taulell i void construeixTaulell(int[][] matriu)

Test: TaulellTest.java, class TaulellTest i testconstrueixTaulell().

En aquest cas hem desenvolupat un test de caixa negra on generem manualment el taulell desitjat després d'instanciar un taulell de 10x10, i mitjançant un assertEquals hem comprovat que efectivament el taulell generat satisfà els requisits.

assertEquals(taulell.get_string_taulell(), taulell_copia);

3. Funcionalitat: El mètode hihaVaixell(int x, int y) retornà un booleà que indica si a les coordenades indicades del taulell des d'on es crida la funció hi ha o no vaixell. Per fer-ho,

s'accedeix a la matriu del taulell i es comprova si el valor a la fila x, columna y correspon a la constant global *VAIXELL*, en cas afirmatiu retorna true i en cas contrari retorna false.

Localització: Taulell.java, class Taulell i boolean hihaVaixell(intx, int y).

Test: TaulellTest.java, class TaulellTest i TesthihaVaixell().

En aquest test realitzem proves de caixa negra. Per fer-ho, col·loquem un vaixell amb el mètode "colocaVaixell" en una posició vàlida (ja que aquí no estem testejant que els límits es compleixin i tal mètode ja està comprovat).

El vaixell es col·loca a les coordenades (0,0) i (1,0), i en primer lloc comprovem els seus valors frontera, que al ser un vaixell d'únicament dos posicions fa que el comprovem sencer. Per fer-ho, comprovem que els valors 00 i 10 de la matriu hagin passat a tenir el valor *VAIXELL* i per tant hihaVaixell ens retorni true (mitjançant assertTrue).. Posteriorment, provem punts que siguin límits exteriors, com 11 o 01, i per comprovar que no continguin el vaixell erròniament i per tant amb assertFalse, veiem que hihaVaixell funciona. Finalment provem algun altre valor per veure que no detecti vaixells que no hi són i el mètode retorni false.

4.Funcionalitat: La funció traductor(int valor), que retorna un caràcter, ens serveix per traduir les posicions de la nostra matriu de control a caràcters que l'usuari entengui per tal de col·locar aquests darrers al taulell que dibuixem.

Per fer-ho simplement implementem un switch-case que en funció del valor introduït a la funció retorni el caràcter que volem utilitzar pel taulell final.

Localització: Taulell.java, class Taulell i char traductor(int valor).

Test: TaulellTest.java, class TaulellTest i TestTraductor().

En aquest test realitzem proves de caixa negra.

En primer lloc creem un taulell, per tal d'utilitzar la funció i comprovar-ne el seu funcionament. Realitzem a continuació mitjançant assertEquals, comprovacions amb tots els valors possibles que ha de saber traduir la funció, equiparant-los als valors esperats per veure així que rebem en tots els casos una traducció correcta del valor.

5.Funcionalitat: El funció colocaVaixell(int x, int y,int mida, boolean horitzontal), és l'encarregat de posicionar els vaixells que l'usuari triï en funció de les coordenades, el tamany i la direcció, dins la matriu de control de taulell, que finalment modificarà també el taulell que es printa a l'usuari (mitjançant el mètode *construeixTaulell*, del punt 2). Per fer-ho, inicialment

comprova que les coordenades introduïdes són vàlides fent que si aquestes són superiors o inferiors als marges superior i/o exteriors del taulell, respectivament, la funció retornarà el valor false (no s'ha pogut col·locar correctament).

Seguidament, comprova amb la funció *hihaVaixell* (punt 3), si ja hi ha un vaixell existent a les coordenades introduïdes, de ser així retorna false, i en cas contrari continua l'execució de la funció.

Ara ja sabem que la posició inicial del vaixell és correcte, però cal comprovar, tant pel cas horitzontal, com pel vertical, si al col·locar el vaixell aquest "trepitjarà" algun altre ja col·locat prèviament. Per fer-ho, es comprovaran amb la funció *hihaVaixell* les si les coordenades on teoricament establirem el vaixell ja estan ocupades o no, i en cas de ser vàlides procedirem a col·locar-lo, ja que totes les comprovacions ens han indicat que el vaixell està correctament situat.

A continuació es crea un objecte de tipus vaixell (Boat), i es fa un *loop* que posteriorment comentarem al test, que afegeix les coordenades que ocupa el vaixell mitjançant el mètode *afegeixCoordenada (*punt 3 de la classe Boat) i modifica la matriu a les coordenades que ocupa el vaixell posant-hi el valor VAIXELL.

Finalment actualitza la llista de vaixells afegint-hi el vaixell introduït, augmenta el nº de vaixells i retorna true ja que en aquest cas s'ha pogut col·locar satisfactoriament.

Localització: Taulell.java, class Taulell i boolean colocaVaixell(int x, int y, int mida, boolean horitzontal)

Test: TaulellTest.java, class TaulellTest i testColocaVaixell().

Per a testejar aquesta funció hem utilitzat les següents tècniques:

- Caixa negra i particions equivalents: Comprovarem amb *assertTrue i assertFalse* si hem pogut col·locar vaixells a les següents particions:
    - Col·locar vaixell correctament (no toca amb cap d'altre i està dins dels límits del taulell):
        - *assertTrue(taulell.colocaVaixell(1, 2, 2, false));*
    - Col·locar vaixell incorrectament fora dels límits del taulell:
        - *assertFalse(taulell.colocaVaixell(11, 4, 4, false)); //coordenada x invàlida*
        - *assertFalse(taulell.colocaVaixell(2, 10, 5, true)); //coordenada y invàlida*
        - *assertFalse(taulell.colocaVaixell(8, 4, 4, false)); //el vaixell surt del taulell*
        - *assertFalse(taulell.colocaVaixell(11, -1, 4, false)); //x i y invàlides*
    - Col·locar vaixell sobre vaixell existent:
        - *assertFalse(taulell.colocaVaixell(1, 2, 2, false))*;
    - Col·locar vaixell que toqui amb un d'existent:
        - *assertFalse(taulell.colocaVaixell(1, 1, 4, true));*

- Proves caixa blanca
    - Loop-testing simple: Hem realitzat aquesta prova sobre el següent bucle:

```
for(i = 0; i<mida;i++)
{
        this.matriuTaulell[x+i][y]= VAIXELL;
        barco.afegeixCoordenada(x+i, y);

}
```

Nota: aquest bucle és pels vaixells verticals, pero procedimentem del mateix mode amb els horitzontals

Per a aquesta prova cal comprovar que el *loop* s'executa correctament pels valors: valor_inicial, valor_inicial+1, valor_interior, valor_final-1, valor_final.

En el nostre cas, hem comprovat que aquests valors s'executen satisfactoriament mirant que les coordenades corresponents al vaixell que testejem continguin el valor VAIXELL a les posicions de la matriu que utilitza el *loop,* com ara bé:

```
//Comprovació Portaviones dins taulell
taulell.colocaVaixell(6, 1, 5, true);
assertEquals(taulell.getValor(6, 1), 1);
assertEquals(taulell.getValor(6, 2), 1);
assertEquals(taulell.getValor(6, 3), 1);
assertEquals(taulell.getValor(6, 4), 1);
assertEquals(taulell.getValor(6, 5), 1);
```

En aquest cas els valors del loop-testing són: valor_inicial=0, valor_inicial+1=1, valor_final=5, valor_final-1=4, valor interior=3

- Condition i decision coverage: Per a realitzar aquesta prova cal comprovar que totes les condicions dels condicionals prenen els diferents valors que fan complir el condition coverage i a més a més prenen totes les decisions(salts) possibles per realitzar el decision coverage.

  A continuació mostrem els screenshots conforme totes les condicions han sigut comprovades i les comentem respecte el test.

```
91      public boolean colocaVaixell(int x, int y, int mida, boolean horitzontal)
92      {
93          boolean colocat=false;
94          boolean vaixell = false;
95          if(x<0 || x>9)
96          {
97              return colocat;
98          }
99          if(y<0 || y>9)
100         {
101             return colocat;
102         }
103
104
105
106         if(hihaVaixell(x,y))
107         {
108
109             //System.out.println( "Posició ocupada per un vaixell");
110
111             return colocat;
112
113         }
```

En aquest fragment de codi inicialment fem condition coverage amb els dos primers condicionals *if*, fent que les coordenades x i y valguin menys de 0 en algun assert i més de 9 en algun altre:

assertFalse(taulell.colocaVaixell(-1, 3, 5, true));
assertFalse(taulell.colocaVaixell(11, -1, 4, false));

A continuació, farem el condition coverage fent col·locant un vaixell correcte i un sobre un vaixell (crida a *hihaVaixell*).

assertTrue(taulell.colocaVaixell(6, 1, 5, true));
assertFalse(taulell.colocaVaixell(6, 1, 5, true));

Finalment per complir decision coverage en aquests casos només cal complir condition coverage.

```
114         else
115         {
116
117             //Posició passada per paràmetre és vàlida, ara comprovar la mida i el sentit
118
119                     if(!horitzontal)
120                     {
121                         if(x+mida-1 <= 9)
122                         {
123                             int i = 0;
124                             while(!vaixell && i<mida) {
125                                 if (!hihaVaixell(x+i,y)) {
126                                     i++;
127                                 }
128                                 else {
129                                     vaixell = true;
130                                 }
131                             }
132                             if(!vaixell) {
133
134                                 Boat barco=new Boat(mida);
135
136                                 //loop testing simple-----------------------------------
137
138                                 for(i = 0; i<mida;i++)
139                                 {
140
141                                     this.matriuTaulell[x+i][y]= VAIXELL;
142                                     barco.afegeixCoordenada(x+i, y);
143
144                                 }
```

En aquest fragment de codi, primerament complim condition i decision fent que un vaixell creat sigui horitzontal i un sigui vertical, com podem veure en exemples anteriors. Seguidament comprovem amb un segon *if* si el vaixell sortiria dels marges al crear-lo (x+mida-1 <=9), que provem en ambdòs casos amb un vaixell que càpiga al taulell i un que no.

assertTrue(taulell.colocaVaixell(1, 2, 2, false));
assertFalse(taulell.colocaVaixell(8, 4, 4, false));

Seguidament comprovem si al crear el nou vaixell tocarà amb algun vaixell existent, recorrent un bucle while que fa un *and* entre els booleans *no hi ha vaixell (!vaixell)* i *"i<mida"*, que sempre farà *decision i condition coverage* perquè funciona de la següent manera:
Les dues condicions s'inicialitzen a *true* i *false* respectivament, i el canvi en una sola d'elles fa que surti del bucle i es compleixi així el decision coverage. Podem afirmar que sempre complirà la condició de sortir i que a més a més podrà sortir de les dues maneres perquè el *while* dins té dues condicions, una de les quals activa una condició de sortida, i una altra n'activa l'altra, i com el node de dins fa condition coverage, podem afirmar que al el while també el complirà, concretament fent que en un assert el vaixell no toqui cap altre (primera condició dins el *while,* i fent que en un altre *assert*, si que toqui un vaixell existent.
i per fer condition cal provar ambòs casos.

assertTrue(taulell.colocaVaixell(1, 2, 2, false));
assertFalse(taulell.colocaVaixell(4, 2, 4, false)); //toca amb un altre vaixell

A continuació comprovem si hem tocat amb un vaixell o no per veure si podem col·locar-lo (es comprova amb els asserts anteriors), i finalment tenim un for que recorre la mida del vaixell per

col·locar-lo a la matriu i guardar-ne les coordenades, i que fa condition i decision per si sol, ja que sempre entra al node *for* i sempre surt perquè és un recorregut.

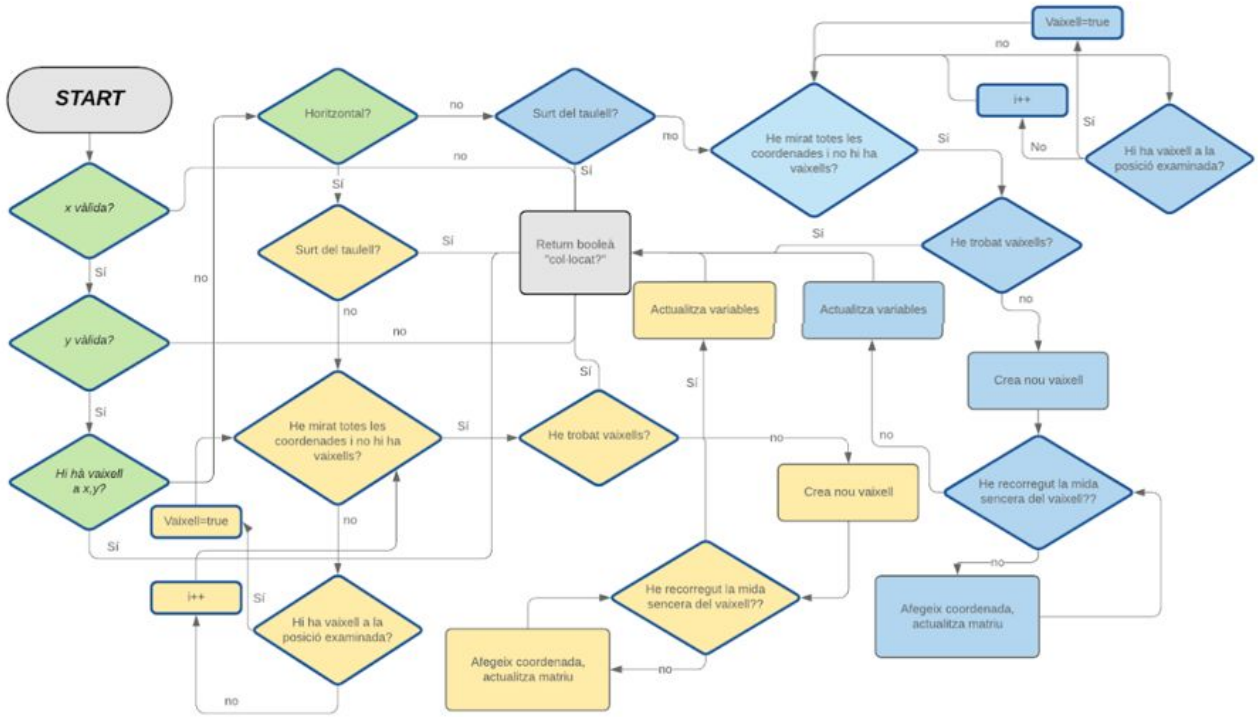
```
146                                 //loop testing simple--------------------
147
148                                 this.llistaVaixells[num]=barco;
149                                 num++;
150                                 colocat=true;
151                                 return colocat;
152                             }
153
154                         }
155
156
157                     }
158                     else //posició horitzontal
159                     {
160                         if(y+mida-1 <= 9)
161                         {
162                             int i = 0;
163                             while(!vaixell && i<mida) {
164                                 if (!hihaVaixell(x,y+i)) {
165                                     i++;
166                                 }
167                                 else {
168                                     vaixell = true;
169                                 }
170                             }
171                             if(!vaixell) {
172
173                                 Boat barco=new Boat(mida);
174                                 for(i = 0; i<mida;i++)
175                                 {
176
177                                     this.matriuTaulell[x][y+i]= VAIXELL;
```

```
171                             if(!vaixell) {
172
173                                 Boat barco=new Boat(mida);
174                                 for(i = 0; i<mida;i++)
175                                 {
176
177                                     this.matriuTaulell[x][y+i]= VAIXELL;
178                                     barco.afegeixCoordenada(x, y+i);
179                                 }
180                                 this.llistaVaixells[num]=barco;
181                                 num++;
182                                 colocat=true;
183                                 return colocat;
184                             }
185                     }
186
187             }
188
189         }
190
191         return colocat;
192
193     }
```

Després del *loop (línea 138)*, s'actualitzen certes variables i es retorna el valor de col·locat, finalitza la funció per aquest *path.*

A continuació trobem l'*else* del node *if-else* que regulava si la posició era horitzontal o vertical, i es fa el mateix que amb horitzontal pero recorrent la matriu de control horitzontalment, per tant no cal comentar altre cop node per node.

- Path coverage:



Aquest és el diagrama de flux que indica com es distribueix el flux d'execució del mètode en funció de les condicions que es triïn. Com podem observar, hi ha diferents colors, que hem colorejat per explicar que el flux groc i el blau que surten del verd, fan el mateix exactament canviant una petita part del càlcul, pero en el codi es distribueixen com el flux que veiem.

Com hem vist abans, fem statement coverage i per fer-ho, hem hagut de recórrer tots els camins possibles del nostre diagrama.

Finalment cal indicar que la complexitat és de 15, què és el nº de nodes de condició +1.

6.Funcionalitat: El mètode dispara és un dels mètodes més importants per a la funcionalitat del joc, ja que s'encarrega de simular l'atac del jugador a la IA a través de dues coordenades que són passades com a paràmetre del mètode i el taulell on s'ataca que també és un paràmetre del mètode. Per fer-ho, inicialment comprovem que el jugador ha inserit coordenades correctes, ja que de no ser així ho considerarem com a moviment erroni i es perdrà el torn d'atac. Seguidament, comprovarem si la posició de la matriu on vol atacar el

jugador conté aigua, un vaixell tocat prèviament o un enfonsat a partir de comparar el valor de la seva matriu amb les constants corresponents, ja que en el cas de disparar a algun d'aquests tres elements, no hi haurà modificacions. Posteriorment, mitjançant la mateixa comprovació amb la matriu del taulell enemic (gràcies al getter getMatriu()), veurem si la posició conté el valor INEXPLORAT o si conté VAIXELL, ja que operarem diferent en aquests dos casos, els quals descriure a continuació com a "cas a" i "cas b", respectivament.

Cas a) Com el valor de la matriu atacada és INEXPLORAT, això vol dir que al inicialitzar-se no se li ha col·locat un vaixell, i per tant el que farem en aquest cas és modificar el valor INEXPLORAT pel valor AIGUA, mitjançant el mètode modificaMatriu().

Cas b) Si hem arribat fins aquí, només cap la possibilitat de que hi hagi un vaixell, per tant, el primer que farem és comprovar si el vaixell té 1 o més vides (caselles no disparades), mitjançant un getter (getNumVides()). Si té 1 sola vida, recorrerem totes les posicions del vaixell i canviarem el seu valor a la matriu per ENFONSAT, ja que aquesta darrera vida serà la que eliminarem. Si el vaixell té més d'una vida, només cal canviar el valor de la matriu a la posició on disparem per TOCAT.

Finalment disminuim en 1 el nº de vides del vaixell amb la crida al mètode disparat() i retornem true si ha tocat o enfonsat vaixell i false si no ho ha aconseguit.

Localització: Taulell.java, class Taulell i bool dispara(Taulell t, int x, int y).

Test: TaulellTest.java, class TaulellTest i TestDispara().

En aquest test hem emprat proves de caixa negra.

En primer lloc creem dos taulells, el "disparador" i el "víctima". Després col·loquem un seguit de vaixells correctament al taulell víctima i per disparar-lo comprovem els següents casos:

-Disparem a un vaixell no disparat prèviament.

-Disparem en una posició per 2n cop.

-Disparem a un vaixell TOCAT.

-Disparem a una posició invàlida.

-Disparem a l'aigua.

7.Funcionalitat: En aquest mètode es modifica la matriu de integers que tenim definida com a atribut privat en funció de les posicions 'x' i 'y' que passem per paràmetre i el valor, és a dir, en

la posició que passem per paràmetre cambien el valor que tingui per el valor que li passem per paràmetre. Aquest són els valors que pot tindre la matriu:

public static final int INEXPLORAT = 0;

public static final int VAIXELL = 1;

public static final int AIGUA = 2;

public static final int TOCAT = 3;

public static final int ENFONSAT = 4;

Localització: Taulell.java, class Taulell i public void modificaMatriu(int valor, int posX, int posY).

Test: TaulellTest.java, class TaulellTest i public void testModificaMatriu().

En aquest test realitzem proves de caixa negra i apliquem la tècnica de partició equivalent. Primer de tot creem un objecte de tipus taulell amb una mida de 10x10. Tot seguit cridem al mètode *modificaMatriu()* i li passem per paramètre en quina posició del taulell volem modificar el valor i el valor pel qual el canviarem, per exemple:

taulell.modificaMatriu(1, 0, 0);

taulell.modificaMatriu(2, 5, 5);

taulell.modificaMatriu(3, 9, 9);

Després, el que fem és comprovar cridant al mètode assertEquals() que el valor en les posicions indicades anteriorment ha estat modificat pel valor indicat:

assertEquals(taulell.getValor(0, 0),1);

assertEquals(taulell.getValor(9, 9),3);

assertEquals(taulell.getValor(5, 5),2);

Una vegada comprovat que els valors són els indicats, tornem a fer el mateix però amb tots els altres valors i altres posicions.

8.Funcionalitat: En aquesta funció es recorre en dos *loop simples* aniuats el llistat d'objectes que tenim de tipus *Boat*. Cridarem a les funcions *getCoordenadaX()* i *getCoordenadaY()* per a que ens retorni els objecte de tipus *Boat* que hi hagi en el llistat de *Boat* i una de les seves posicions coincideixi amb els valors passats per paràmetre ‘x’ i ‘y’.

Localització: Taulell.java, class Taulell i public Boat getBoat(int x,int y).

Test: TaulellTest.java, class TaulellTest i public Boat getBoat(int x,int y).

En aquest test realitzem proves de caixa negra, apliquem la tècnica de partició equivalent i comprovem els valors frontera i límit. Primer de tot, creem un objecte de tipus taulell 10x10 i col·loquem un vaixell en el taulell. Després, comprovem amb un *assertEquals()* que la primera posició del llistat de vaixells, és a dir, el primer vaillex del llistat, sigui el mateix que quan cridem a la funció *getBoat()* en la posició que hem col·locat el vaixell, per exemple:

```
t.colocaVaixell(1, 2, 5, false);

//frontera

assertEquals(t.getLlistaVaixells()[0], t.getBoat(1, 2) );

assertEquals(t.getLlistaVaixells()[0], t.getBoat(5, 2) );
```

Una vegada ja tenim comprovat els valors frontera del vaixell en el taulell, comprovem que els valors límits interiors també formen part del mateix vaixell que abans.

```
//limits interiors

assertEquals(t.getLlistaVaixells()[0], t.getBoat(2, 2) );

assertEquals(t.getLlistaVaixells()[0], t.getBoat(4, 2) );
```

Per acabar comprovem els valors interiors i valors límits exteriors, pels límits exteriors no han de pertànyer al vaixell. Després tornarem a col·locar un vaixell en la taulell que s'afegeix al llistat de vaixells i tornarem a fer les mateixe comprovacions.

```
//limits exteriors

assertEquals(null, t.getBoat(4, 6) );

assertEquals(null, t.getBoat(4, 9) );
```

9.Funcionalitat: Aquesta funció permet traduir la matriu enemiga per no veure'n els vaixells. Es crea una matriu de integers auxiliar amb les mateixes mides que l'original i li passem per paràmetre a la funció la matriu original. Tenim dos *loops simples* aniuats que recorren tota la matriu original i si en alguna posició es troba que el valors és *VAIXELL*, a la matriu auxiliar en la mateixa posició que l'original es canvia el valor i passa a ser *INEXPLORAT,* d'aquesta manera

aconseguim que quan es jugui la partida no es puguin veure on ha col·locat els vaixells en el taulell laIA.

Localització: Taulell.java, class Taulell i public int[][] traductorIA(int[][] matriu).

Test: TaulellTest.java, class TaulellTest i public void testTraductorIA(). En aquest test realitzem proves de caixa negra, apliquem la tècnica de partició equivalent i comprovem els valors frontera i límit. Primer de tot creem un objecte de tipus Taulell amb una mida de 10x10 i cridem al mètode *colocaVaixells()* per col·locar els vaixells al taulell. Seguidament, creem una matriu auxiliar amb la mateixa mida i valors que l'original, i cridem a la funció *traductorIA()*:

```
int[][] matriuaux= t.traductorIA(t.getMatriu());
```

Ara comprovem que els vaixell col·locats anteriorment no estiguin en aquesta matriu, per això comprovem els valors frontera i límits a les posicions on hem col·locat els vaixells i ara el seu valor haurà de ser INEXPLORAT:

```
//valors frontera

    assertEquals(matriuaux[1][2],INEXPLORAT);

    assertEquals(matriuaux[1][3],INEXPLORAT);

    assertEquals(matriuaux[2][5],INEXPLORAT);

    assertEquals(matriuaux[2][7],INEXPLORAT);

//valors limit

    assertEquals(matriuaux[2][6],INEXPLORAT);

    assertEquals(matriuaux[1][1],INEXPLORAT);

    assertEquals(matriuaux[1][4],INEXPLORAT);

    assertEquals(matriuaux[2][8],INEXPLORAT);
```

10.Funcionalitat: En aquesta funció li passem per paràmetre l'objecte taulell al que es vol disparar, el que fa es generar dos numeros enters aleatoris entre 0 i 9, i aquests números els passem com a paràmetre a la crida de la funció *dispara()* juntament amb el taulell a disparar i retornem la funció que ens retorna *true* si ha pogut disparar o *false* si no ha sigut possible.

Per generar els números aleatoris, tenim implementada una classe que es troba al arxiu *NumAleatori.java*, la classe es diu class *NumAleatori* i el únic mètode que té és *public int*

generaNumeroAleatori(int minimo,int maximo). Aquesta funció no la testejem ja que per testejar la funció *IAdispara()* necessitarem fer un mock object de la classe *NumAleatori.*

Localització: Taulell.java, class i public boolean IAdispara(Taulell t).

Test: En aquest test realitzem proves de caixa negra, apliquem la tècnica de mockups i partició equivalent.

Primer de tot abans de testejar la classe hem de crear un mockObject de classe NumAleatori per a poder controlar quins valors volem que ens generi aleatòriament i per a comprovar si s'ha desenvolupat correctament el mètode.

Per crear el mockObject hem creat una classe que es troba a MockNumAleatori.java, la classe es diu MockNumAleatori i hereda de la classe NumAleatori, tenim diferents mètodes pero el més important per aquest test és el *public int generaNumeroAleatori(int minimo,int maximo)* que ens retorna una posició d'un array de enters que hem definit nosaltres sabent quins són els valors exactes. Per a cridar-lo en el test hem d'haver creat prèviament un objecte de tipus NumAleatori a la classe Taulell i implementar un setter d'aquest objecte.

Implementació del mockObject en el test:

```
MockNumAleatori mock = new MockNumAleatori();

Taulell taulellIA = new Taulell(10,10);

taulellIA.setRandom(mock);

taulellIA.creaTaulellIA();

Taulell taulellVictima= new Taulell(10,10);

//coloquem vaixells correctament

taulellVictima.colocaVaixell(0, 0, 2, true);

taulellVictima.colocaVaixell(2, 0, 3, true);

taulellVictima.colocaVaixell(7, 4, 4, true);

taulellVictima.colocaVaixell(3, 1, 5, true);
```

Una vegada ja tenim el mockObject creat i sabem a quines posicions dispara, el que fem és col·locar vaixells en el taulell (col·locarem uns quants a les posicions que sabrem on dispararem per comprovar en el cas de que dispari, si ha tocat el vaixell o no). Comprovarem

amb *assertTrue()* i *assertFalse()* si s'ha pogut disparar al taulell que passem per parametre quan cridem la funció *IAdispara()*, aquests són uns exemples.

assertTrue(taulellIA.IAdispara(taulellVictima)); //hi ha vaixell disparo 00

assertFalse(taulellIA.IAdispara(taulellVictima)); //no hi ha vaixell disparo 1,2

11.Funcionalitat: Aquest mètode, coloca vaixells en posicions aleatòries del taulell de la IA. Tenim una estructura repetitiva de tipus while(), com a condició tenim que una variable enter *n* no pugui ser igual 1 i la tenim fora del loop inicialitzada a 5, d'aquesta manera aconseguim que per cada iteració la mida sigui igual a aquesta variable, per tant la mida de cada vaixell mai serà la mateixa i sempre serà entre 5 i 2.

El que fem apart de assignar la mida, és generar dos números aleatoris entre 0 i 9 per a la posició tal com feiem en el mètode *IAdispara()* i també generem un número aleatori entre 0 i 1 per indicar si col·loquem el vaixell en posició vertical o horizontal.

Per últim, cridem al mètode *colocaVaixell(x, y, mida, horitzontal)* passant per paràmetre les variables anteriors i aquest mètode s'encarregara de la resta, si s'ha pogut col·locar ens retorna true i llavors es decrementa en un valor la variable *n*.

Localització: Taulell.java, class Taulell i public void creaTaulellIA().

Test: TaulellTest.java, classTaulellTest i public void testcreaTaulellIA ().

En aquest test realitzem proves de caixa negra, apliquem la tècnica de mockups, partició equivalent i comprovem els valors límits i frontera.

Per controlar els números aleatoris hem de crear el mateix mockObject que utilitzem en el test per la funció IAdispara(). En el array de valors que hi ha en la classe MockNumAleatori, ja tenim controlat que cada dues posicions la tercera sigui d'un valor entre 0 i 1 per controlar el tema de la posició horitzontal i vertical.

```
MockNumAleatori mock = new MockNumAleatori();

Taulell taulellIA = new Taulell(10,10);

taulellIA.setRandom(mock);

taulellIA.creaTaulellIA();
```

Una vegada tenim creat el *taulellIA*, ara passem a comprovar per a cada vaixell si s'han col·locat els vaixells en les posicions indicades en el MockObject, comprovem els valors

frontera,límit interior i exterior i valors interiors per a cada vaixell, posem com exemple les proves de caixa negra del vaixell de mida 4:

```
//Vaixell mida 4

    assertEquals(taulellIA.getValor(2, 0), 1);//Frontera

    assertEquals(taulellIA.getValor(2, 1), 1);//Limit interior

    assertEquals(taulellIA.getValor(2, 2), 1);//Limit interior

    assertEquals(taulellIA.getValor(2, 3), 1);//Frontera

    assertEquals(taulellIA.getValor(2, 4), 0);//Limit exterior
```

Class Joc

1.Funcionalitat: Constructor per paràmetres de la classe *Joc* on li passem per paràmetre dos objectes de tipus *Taulell*. Inicialitzem els atributs privats de la classe (dos objectes de tipus *Taulell*) amb els que passem per paràmetre i tenim un objecte de la classe *LlegeixTeclat* que inicialitzem cridant al seu constructor. (La classe *LlegeixTeclat* no està testejada ja que testejarem el mètode que utilitza aquesta classe amb un MockObject, que explicarem en el seu test corresponent.)

Localització: Joc.java, class Joc i Joc(Taulell t1, Taulell t2).

Test: TestJoc.java, class TestJoc i public void testJoc().

En aquest test realitzem proves de caixa negra. Per comprovar el mètode, inicialitzem dos objectes de tipus *Taulell* amb mida 10x10 i els passem per paràmetre al constructor per paràmetres de tipus Joc. Una vegada tenim creat un objecte Joc, testejem si s'ha creat de forma correcte:

assertEquals(t1,joc.getTaulellJugador());

assertEquals(t2,joc.getTaulellIA());

2.Funcionalitat: La funció acabaJoc() retorna un booleà que indica si el joc ha acabat o no.

Per fer-ho, en primer lloc agafa les matrius de control dels 2 taulells que conté una instància de Joc. Posteriorment creen dos booleans inicialment falsos, que seran qui ens avisaran si el joc ha acabat.

Després es recorren les dues matrius buscant posició per posició si queda algun vaixell viu, ja que en el cas de trobar-ne'n algún, canviaran el booleà que els pertoca a true. Posteriorment veiem si alguna de les matrius ha quedat sense vaixells i de ser així retornarem *true*, acaba joc. En cas contrari retornarem *false.*

Localització: Joc.java, class Joc i boolean acabaJoc().

Test: TestJoc.java, class TestJoc i boolean acabaJoc().

En aquest test realitzem proves de caixa negra i caixa blanca.

- Caixa negra:
    - En primer lloc, creem dos taulells per instànciar un objecte tipus Joc, després els emplenem amb vaixells i cridem:
        - *assertFalse(joc.acabaJoc()); //*Joc no acabat.

- Després disparem fins tombar tots els vaixells d'un taulell i cridem:
  - *assertTrue(joc.acabaJoc());*//Joc acabat.
- Revivim el taulell mort i matem el taulell de la IA.
  - *assertTrue(joc.acabaJoc());*

- Caixa blanca:

```
160    public boolean acabaJoc()
161    {
162        int[][] mIA=this.taulellIA.getMatriu();
163        int [][] mJugador =this.taulellJugador.getMatriu();
164
165        boolean ia=true;
166        boolean jugador=true;
167
168        for(int i=0;i<taulellIA.getMidaY();i++)
169        {
170            for (int j=0;j<taulellIA.getMidaX();j++)
171            {
172                if(mIA[i][j]==VAIXELL) {
173                    ia=false;
174                }
175            }
176        }
177
178
179        for(int i=0;i<taulellJugador.getMidaY();i++)
180        {
181            for (int j=0;j<taulellJugador.getMidaX();j++)
182            {
183                if(mJugador[i][j]==VAIXELL) {
184                    jugador=false;
185                }
186            }
187        }
188
189        if (jugador || ia)
190            return true;
191        else
192            return false;
193
194    }
```

- Decision coverage:
  - Com les úniques dos decisions que pot prendre la funció són acabar el joc o no acabar-lo en funció de les variables booleanes *ia* i *jugador*, es realitza decision coverage al generar almenys un *assert* on hagi acabat el joc i un altre on no hagi acabat.
- Condition coverage:
  - Com les condicions de recorrer el for es donaran sempre inequívocament, només cal que en dos *asserts* o més,els booleans *ia* i *jugador* hagin variat de valor.

3.Funcionalitat: La funció *tradueix(char c)* és l'encarregada de transformar les lletres que ens passa l'usuari per seleccionar una columna als nombres que entén la nostra matriu de control. Per fer-ho tan sols cal un switch-case que retorna valors de 0 a 9 per les lletres de l'A a la J.

Localització: Joc.java, class Joc i int tradueix(char c).

Test: TestJoc.java, class TestJoc i tradueixTest().

En aquest test utilitzem caixa negra per comprovar que tots els valors es tradueixen adequadament, com ara bé:

assertEquals(joc.tradueix('j'),9);

4.Funcionalitat: Aquest mètode ens permet introduir per pantalla, un número enter (Coordenada X), un caràcter tipus char (Coordenada Y) i un altre número enter (Horitzontal o Vertical). Per a poder-ho introduir per pantalla, cridem a dues funcions de la classe *LlegeixTeclat (LlegeixInt() i LlegeixChar()).*

La classe *LlegeixTeclat* utilitza un objecte de la classe *Scanner* per a poder introduir *inputs*. Aquí tenim la declaració del mètode int *LlegeixTeclat()*:

```
public int LlegeixInt() {

    Scanner sc = new Scanner(System.in);

    int x = (int) sc.next().charAt(0) - 48;

    return x; }
```

Fem el mateix pel cas del mètode *LlegeixChar().*

Localització: Joc.java, class Joc i public void LlegeixCoordenades().

Test: TestJoc.java, class TestJoc i public void llegeixCoordenadesTest().

En aquest test realitzem proves de caixa negra, apliquem la tècnica de mockups, partició equivalent i comprovem els valors límits i frontera.

Per controlar quins números s'introdueixen per teclat, hem declarat una classe *MockLlegeixTeclat* que herada de *LlegeixTeclat* i hem implementat les dues funcions *LlegeixInt()* que retorna 0 i *LlegeixChar()* que retorna 'a'.

En el mètode *llegeixCoordenadesTest()*, inicialitzem un objecte de tipus Joc i li passem per paràmetre dos objectes de tipus Taulell que hem inicialitzat amb mida 10x10 cadascun. Tot seguit, creem el MockObject de *LlegeixTeclat* i cridem la funció *LlegeixCoordenades()* que ens inicalitzara les variables amb els valors que hem declarat a la classe *MockLlegeixTeclat*, per

comprovar si s'han assignat correctament els valors cridem els seus getters corresponents i comparem amb els valors esperats.

```
MockLlegueixTeclat mock = new MockLlegueixTeclat();

        joc.setTeclat(mock);

        joc.LlegeixCoordenades();

        assertEquals(joc.getX(), 0);

        assertEquals(joc.getY(), 0);

        assertEquals(joc.getHoritzontal(), 0);
```

En el cas de la coordenada 'Y' ens retorna 0 perquè prèviament hem cridat al mètode *tradueix(char a)* de la classe *Joc*.

5.Funcionalitat: El mètode preparaTaulells(), simplement és una unió del col·locaVaixells que es repeteix fins que l'usuari ha emplenat el seu taulell, i el mètode creaTaulellIA, que crea el taulell enemic. El fet d'unir-los és més que res per unificar codi i que la creació de la partida no sigui un gran fragment de codi, sinó que un únic mètode tingui la funció de fer-ho.

Localització: Joc.java, class Joc i void preparaTaulells().

Test: Per aquesta funció no hem acabat de desenvolupar test, ja que el fet d'utilitzar scanner i ia altre cop, inevitablement ens feia tornar a utilitzar mock objects, i vam optar per unir-los i comprovar jugant que el procediment era correcte(exploratory testing).

Class MasterTest

Funcionalitat: Aquesta classe només implementa la funció executaTest(), que és simplement una crida a tots els tests de les altres classes.

Localització: MasterTest.java, class MasterTest, void executaTest()

Test: Statement Coverage

Per demostrar que hem fet statement coverage (excepte en les classes que substituïm amb mockObjects i ja venen predefinides, i la funció on hem fet Exploratory Testing, executem el procediment executaTest(), i fem un *coverage run*, podem veure detalladament per quines línees de codi hem passat i per quines no. A continuació mostrem un screenshot:

MasterTest (21 nov. 2019 19:32:21)

Element	Covera...	Covered Ins...	Missed Instr...	Total Instruc...
Tetris_ProjectTQS	88,9 %	2.608	327	2.935
src	88,9 %	2.608	327	2.935
(default package)	88,9 %	2.608	327	2.935
main.java	0,0 %	0	182	182
Joc.java	60,8 %	160	103	263
LlegueixTeclat.java	10,3 %	3	26	29
NumAleatori.java	15,8 %	3	16	19
Boat.java	100,0 %	85	0	85
BoaTest.java	100,0 %	156	0	156
MasterTest.java	100,0 %	54	0	54
MockLlegueixTeclat.jav	100,0 %	7	0	7
MockNumAleatori.java	100,0 %	58	0	58
Taulell.java	100,0 %	613	0	613
TaulellTest.java	100,0 %	973	0	973
TestJoc.java	100,0 %	496	0	496

Com podem observar el tant per cent de statement coverage es veu reduït degut a la funció del arxiu Joc.java, en canvi totes els altres arxius tenen un 100% de statement coverage.