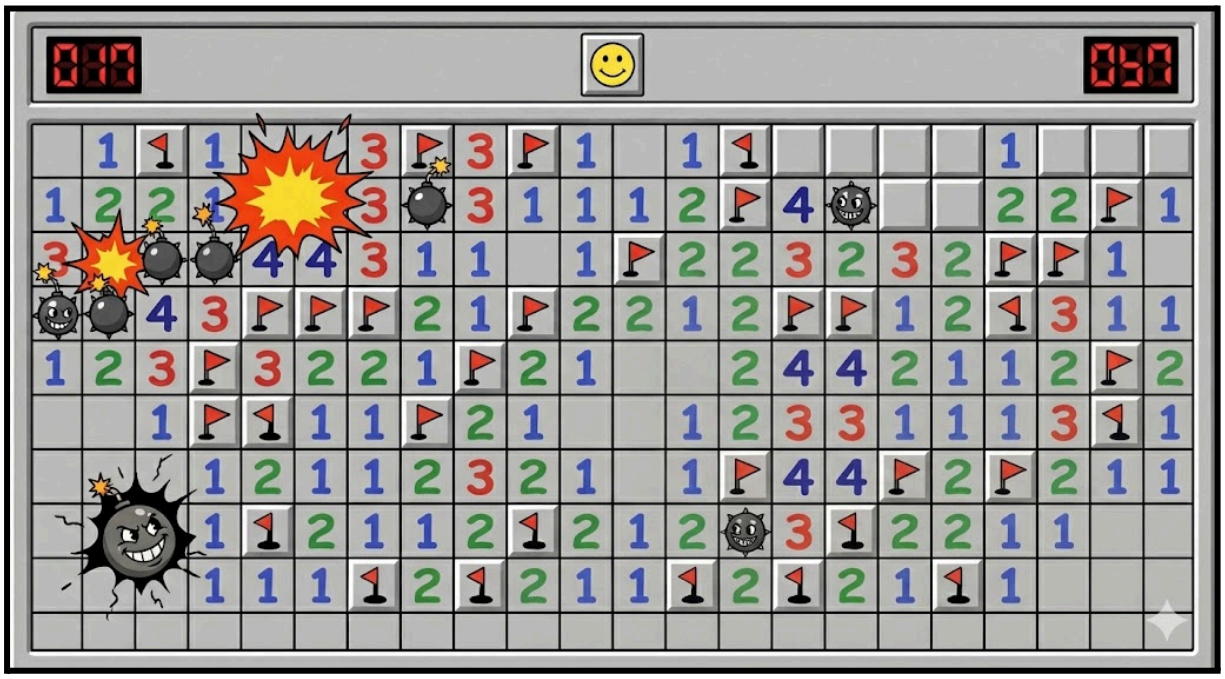
INFORME PRÀCTICA
TEST I QUALITAT DEL SOFTWARE

BUSCAMINES



Integrants:

David Morales López - NIU: 1703724

Ivan Guerra Alarcón - NIU: 1704491

ÍNDEX:

NOTA IMPORTANT

Hem de comentar un petit **error** que ens ha sorgit i ens hem adonat dos dies després de l'entrega del codi de la pràctica. Hem detectat que a la classe JocTest, destinada a fer el testing del controlador del nostre buscaminas, compta amb un petit error a 2 línies del codi, que estaven correctes en el seu moment, però una falta de comunicació entre el el meu company i jo va fer que un dels dos pugés una versió de JocTest que comptava amb un error en aquestas 2 lines (la resta del codi està tot bé i igual). L'error és el següent:

```
257
258             try (BufferedReader br = new BufferedReader(new FileReader("data/simulacio2.txt"))) {
259
```

La versió final que esta al gitHub del test de simulacioTest a la classe JocTest compta amb aquest error a 2 línies del codi, la 258 i la 255. **L'error es la ruta que utilitza el FileReader per llegir els fitxers de text de Simulacio1.txt i Simulacio2.txt, que no estan ben actualitzats a la versió final del gitHub.**

La forma correcta és la següent:

```
257
258         try (BufferedReader br = new BufferedReader(new FileReader("src/test/java/es/uab/tqs/buscamines/simulacio2.txt"))) {
259
```

Amb aquesta ruta (per la simulació 2, en el cas de l'altre simulació es cambiar el 2 per 1 al final)

Rutes correctes:

Simulació 1: **`src/test/java/es/uab/tqs/buscamines/simulacio1.txt`**
Simulación 2: **`src/test/java/es/uab/tqs/buscamines/simulacio2.txt`**

Context de la pràctica:

La nostra pràctica es basa en la creació del conegut joc o aplicació del Buscamines clàssic. Joc en què, per guanyar, s'han de destapar (emprant el clic esquerre) totes les caselles que no siguin mines. En cas de clicar una d'aquestes mines amb la intenció de destapar la casella on es troba, es perd i la partida s'acaba. Per evitar aquest cas, hi ha la capacitat de marcar les mines amb una bandera mitjançant el clic dret del ratolí.

Enfocant-nos en el nostre Buscamines en especial, compta amb un tauler de 13 × 13 caselles, és a dir, un total de 169 caselles, on es reparteixen 30 mines de manera aleatòria. Cal remarcar que l'algorisme de generació de les coordenades de les mines contempla el primer clic del jugador; és a dir, en el nostre Buscamines no es pot perdre directament en el primer clic, sinó que s'utilitza un mètode especial que permet generar les mines aleatòriament sense interferir amb la primera jugada de l'usuari. Això garanteix una experiència de joc molt més agradable i permet realitzar jugades més racionals en comptes de "clicar per clicar". Amb tot això aclarit, començarem amb l'explicació del testing realitzat a la pràctica, que no és poc!

Enllaç GitHub: https://github.com/Daviteee/BuscaMinas.git

Important!
El nostre projecte consta de 4 classes principals:

- **Model**:
  - **Casella:** Classe que representa la unitat bàsica del Buscamines, on poden aparèixer mines, números o banderes. Compta amb atributs com: teMina (boolean), teBandera (boolean), estaDestapat (boolean) i numMinesVoltant (int).
  - **Tauler**: Classe que representa el conjunt de caselles i que estableix la base per poder jugar una partida al Buscamines. Compta amb la mida del tauler definida (13) i amb atributs com: una matriu de caselles, el nombre màxim de mines al tauler i un atribut de tipus Random per poder generar les mines de manera aleatòria.
- **Controlador**:
  - **Joc**: Classe que, com el seu package indica, controla el flux de la partida del Buscamines, permetent realitzar els clics corresponents a cada casella i comunicant-se amb la vista i el model per garantir la integritat del joc. Compta amb un tauler de tipus Tauler, un atribut que indica si la partida està acabada, un atribut de la vista i el nombre de banderes restants per actualitzar el comptador que posteriorment es mostra a la vista del joc. Aquesta classe és un pilar en la comunicació del joc i compta amb mètodes com clicEsquerra i clicDret, que permeten realitzar les accions del joc i asseguren que la lògica del joc no es trenqui en cap moment.
- **Vista**:
  - **BuscaminesVista**: Finalment, trobem la vista de l'aplicació, encarregada de mostrar tots els esdeveniments que passen al joc a l'usuari. Per a la vista, hem utilitzat la interfície gràfica més senzilla que existeix per a Java (JFrame i JPanel). Compta amb una matriu de botons que escolten els clics del ratolí i utilitzen la informació enviada pel controlador (Joc) per mostrar els diferents

esdeveniments en temps d'execució. També inclou un comptador de banderes per assegurar una experiència de joc agradable i té una aparença user-friendly. Com indica l'enunciat, la vista no està testejada.

Imatges del projecte:


```java
package controlador;

import java.util.Random;

import model.Tauler;
import vista.BuscaminesVista;

public class Joc {

    private Tauler tauler;
    private boolean partidaAcabada;
    private BuscaminesVista vista;
    private int banderesRestants;

    public Joc(Tauler t) { //Constructor per paràmetres on li pasem el tauler del joc.
        //Precondició per comprovar que hi ha tauler.
        assert(t != null) : "Error el tauler és null";

        this.partidaAcabada = false;
        this.banderesRestants = 30;
        this.tauler = t;
        this.vista = null; //Inicialitzem la vista a null al constructor.

        //Postcondicions per comprovar que s'ha iniciat correctament el joc abans de generar la vista.
        assert(!this.partidaAcabada) : "Error la partida s'ha inicialitzat acabada(incorrecte)";
        assert(this.banderesRestants == 30) : "Error, el numero de banderes no s'ha inicialitzat correctament";
        assert(this.tauler == t) : "Error el tauler no s'ha iniciat correctament amb el tauler per paràmetre";
        assert(this.vista == null) : "Error la vista no s'ha iniciat a null correctament";
    }
```

CasellaTest:

Començarem explicant el testing realitzat a la classe més bàsica del projecte. Dins d'aquest testing trobem un total de quatre tests: PosarMinaTest, PosarAndTreureBanderaTest, DestaparCasellaTest i setNumMinesVoltantTest. També compta amb un setup per inicialitzar una casella que es testeja al llarg de tot el testing.

PosarMinaTest:

→ Aquest test es basa en la comprovació que l'atribut teMina es posa amb el valor que li indiquem. És un test simple que només comprova el correcte valor de l'atribut teMina de la classe Casella. S'utilitzen asserts per verificar els resultats esperats.

PosarAndTreureBanderaTest:

→ De la mateixa manera que el test anterior, aquest és un test simple que comprova que el valor assignat a l'atribut teBandera de la classe Casella és el correcte. S'utilitzen asserts per verificar els resultats esperats.

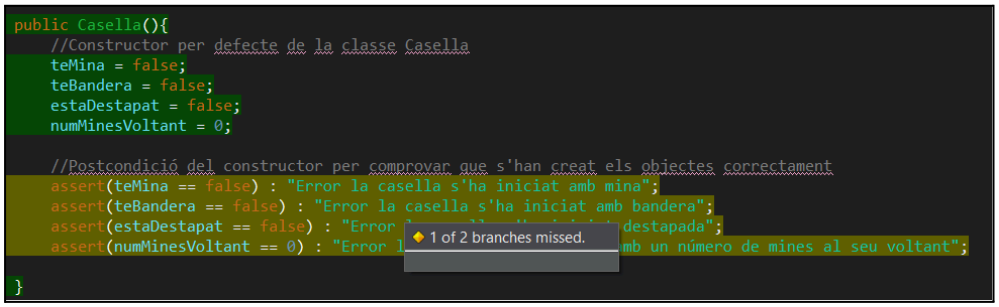
DestaparCasellaTest:

→ Un altre test simple però necessari comprova que l'atribut estaDestapat de la classe Casella té el valor assignat correctament. Aquests tests, tot i semblar insignificants a simple vista, asseguren que l'assignació de valor als atributs de la casella és correcta. S'utilitzen asserts per verificar els resultats esperats.

SetNumMinesVoltantTest:

→ Aquest test es diferencia dels altres de la classe Casella, ja que és el primer que utilitza particions equivalents. En el Buscamines, el nombre màxim de mines que pot tenir una casella al voltant és 8 i el mínim 0; per tant, ens trobem amb la primera partició equivalent del projecte. Comprovem els valors frontera (0 i 8), els valors límit (-1, 1, 7 i 9) i un valor del mig de la partició (4, per exemple). S'utilitzen asserts per verificar els resultats esperats.

La classe Casella no compta amb cap prova de caixa blanca, ja que és una classe massa senzilla per poder aplicar aquest tipus de testing.
El statement coverage de la classe Casella és del 72,2%; el percentatge que falta es deu als asserts que formen part de les postcondicions del codi desenvolupat, ja que és impossible provocar que falli un operador d'assignació.



El condition i decision coverage és del 100% ja que executem totes les possibles combinacions dels condicionals que hi ha al codi desenvolupat.

```
public void setNumMinesVoltant(int num) {
    //Setter de l'altribut numMinesVoltant
    //Precondició per veure que el número de mines a instaurar es correcte (entre 0 y 8)
    if(num < 0) {
        throw new IllegalArgumentException("Nombre de mines menor a 0!!!");
    }
    if(num > 8) {
        throw new IllegalArgumentException("Nombre de mines major a 8!!!");
    }

    this.numMinesVoltant = num;
    //Post condició per veure si el número de mines al voltant s'ha instaurat correctament
    assert(numMinesVoltant == num) : "Error el número de mines al voltant no s'ha instaurat correctament";
}
```

TaulerTest:

Seguim amb el testing del model, ara amb la classe Tauler. On hem fet un testing molt més dens comparat amb el de la classe Casella. En aquest testing hem emprat aspectes com Mock Objects, particions equivalents, path coverage. El testing de la classe Tauler es basa en comprovar els límits del tauler principal, on contemplarem una partició equivalent que serà essencial pel testing de Tauler, que són els índexs de les coordenades del tauler:
El tauler és una matriu de caselles 13x13, on els índexs van de 0 a 12 (des de la casella 0,0 fins a la casella 12,12). Per tant, el testing de particions equivalents tindrà com a valors frontera el 0 i el 12, com a valors límit el -1, 1, 11 i 13 i alguns valors de dins de la partició com per exemple el 6. **Cal remarcar que els mètodes mai reben més de dos paràmetres (les coordenades de la casella en qüestió), per tant el Pairwise testing serà totes les combinacions i per tant no podem fer com tal un exercici de Pairwise Testing en el nostre projecte, aquest és un aspecte que hem parlat molts cops amb el docent Xavier Otazú i ens ha aclarit que si no es pot fer no es pot fer i que ho comentarem a l'informe.**

Els tests que hem fet per la classe Tauler són:

- ConstructorInicialTest → Test dels constructors de Tauler.
- getCasellaTest → molt important ja que comprova límits i permet evitar testing repetitiu.
- generaMinesRandomTest → Comprova la correcta generació de les mines de manera aleatoria al Tauler.
- setNumMinesVoltantTest → Comprova la generació del nombre de mines al voltant d'una casella en concret amb diferents casos.
- destapaCasellaTest → Comprova el correcte funcionament del destapament de les caselles en tots el casos.
- totesDestapadesSenseMinesTest → Comprovar la condició de victoria.
- setMaxMinesTest → Comprova la inicialització del nombre de mines del tauler.
- destaparCasellesAmbMines → Comprova casos especial per destapar caselles amb mines.

També, el testing de tauler compta amb mètodes privats que ajuden a fer el codi de test més amè, per tractar el tema de la repetició de codi o els try catch excessius.

Aspectes importants de TaulerTest:

Mock Objects:

A la classe Tauler comptem amb un atribut de tipus Random (myRandom) que ens ajuda a generar de manera aleatòria les coordenades x i y de la casella on es generarà la mina corresponent. L'utilitzem al mètode generaMinesRandom d'aquesta manera:

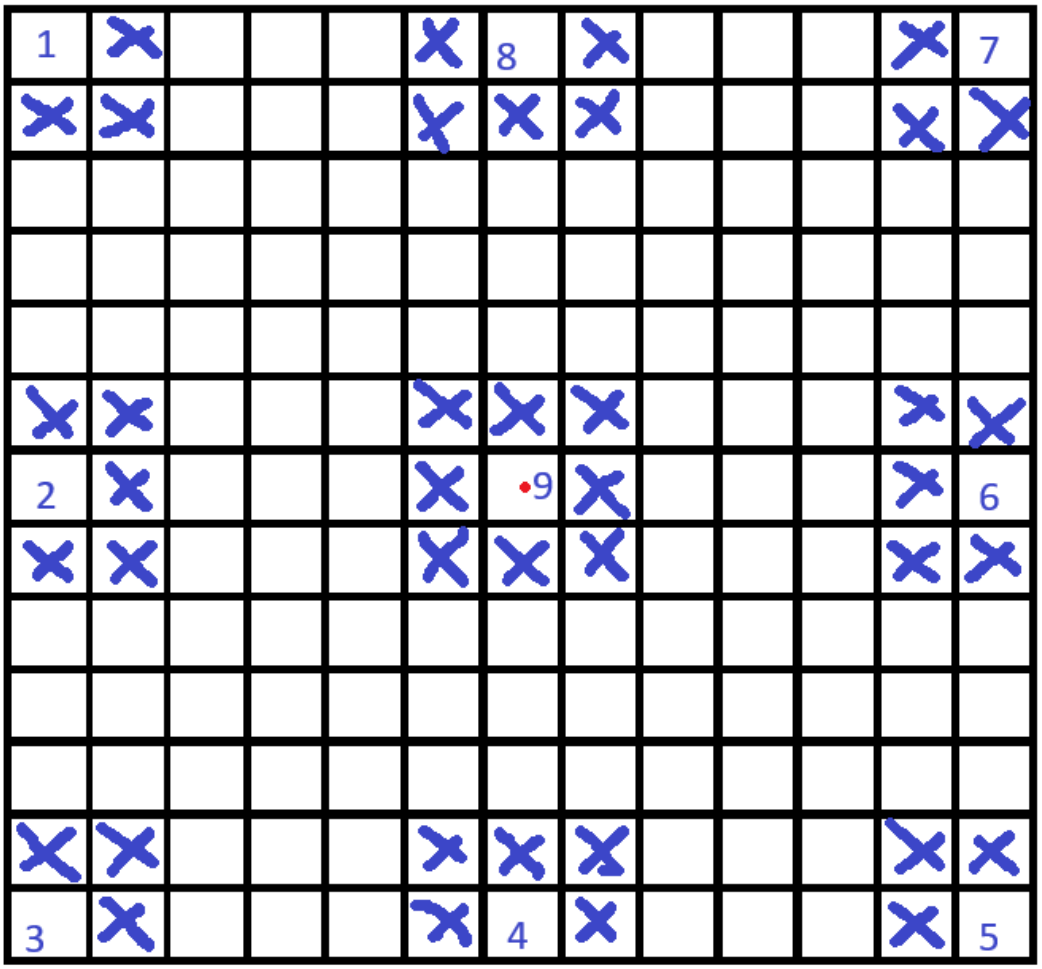
```
public void generaMinesRandom(int xPlayer, int yPlayer) {
    invariant();
    if(xPlayer > 12 || yPlayer > 12 || xPlayer < 0 || yPlayer < 0){
        //Precondició dels límits del tauler segons les coordenades del clic del jugador.
        throw new IllegalArgumentException("Les coordenades del clic de l'usuari no són correctes (fora dels limits
    }

    int nMines = 0; //Número de mines al teuler.

    while(nMines < nMaxMines) { //Iterem fins que hi hagi 30 mines al tauler ben col·locades.
        int xMina = myRandom.nextInt(13); //Generació de x aleatòriament del 0 al 12.
        int yMina = myRandom.nextInt(13); //Generació de y aleatòriament del 0 al 12.

        if( Math.abs(xMina - xPlayer) <= 1 && Math.abs(yMina - yPlayer) <= 1) {
            //La posició aleatòria generada de la mina està dins de les 8 del voltant on el jugador a fet el clic i
            // La norma del buscamines diu que quan clica no pot haver-hi una mina en les 8 caselles del voltant. P
            continue;
        }

        if(!tauler[xMina][yMina].isMina()) {
            tauler[xMina][yMina].setMina(); //Posem la casella generada correctament com una mina (mina = true) per
            nMines++; //Augmentem el nombre de mines ben col·locades al tauler.
        }

    }
    invariant();
}
```

Com hem vist a les classes de teoria, un objecte Random no es pot utilitzar per fer testing ja que no podem saber quin valor ens donarà en cada execució, per tant, hem implementat un Mock Object de Random anomenat MockRandom:

```
package model;

import java.util.Random;

public class MockRandom extends Random{
    int opcio1,opcio2,n_mina;
    boolean is_x;

    public MockRandom(int opcio1,int opcio2){
        this.opcio1 = opcio1; //Per saber quins valors de x o y retornar
        this.opcio2 = opcio2;
        this.n_mina  = 0;
        this.is_x = true;
    }

    public int nextInt(int x) {
        // Retornar els valors que nosaltres volguem entre (0 - x)
        int[][][][] valor = {
                //Casella 0-0
                {
                    {{0,1}}, //1mina
                    {{0,1},{1,0}}, //2mines
                    {{0,1},{1,0},{1,1}} //3mines
                },
                //Casella 6-0
                {
                    {{5,0}}, //1mina
                    {{5,0},{5,1}}, //2mines
                    {{5,0},{5,1},{6,1}},    //3mines
                    {{5,0},{5,1},{6,1},{7,0}},  //4mines
                    {{5,0},{5,1},{6,1},{7,0},{7,1}} //5mines
```

Aquest Mock consisteix en un array de 4 dimensions, on al punt més atòmic ens trobem amb un array amb 2 coordenades; aquests valors representen la posició d'una mina al Tauler. Per exemple, a la casella 0,0 podem posar una mina a la posició 0,1 (1 mina), o a la 0,1 i a la 1,0 (2 mines) i així fins arribar al màxim de mines que es poden posar en aquella casella.

Per explicar això millor, utilitzarem el esquema del tauler que ens va ajudar a implementar el Mock complet amb tots els casos:



En aquest esquema podem veure els 9 casos que implementa el MockRandom per posar les mines, els nombres representen el número de cas en què estem: Els casos 1, 3, 5 i 7 són casos en què el **nombre màxim de mines al voltant de la casella pot ser 3**, ja que són les cantonades del tauler i casos de particions equivalents (valors frontera) importants (la 0,0, la 0,12, la 12,0 i la 12,12). Després trobem els casos 2, 4, 6 i 8, que són les caselles del lateral del tauler que consten de combinacions de valors frontera amb valors de dins de la partició equivalent pel que fa a les coordenades (la 0,6, la 12,6, la 6,12 i la 6,0); en aquests casos, el **nombre màxim de mines que pot tenir la casella al voltant és 5**. Finalment, trobem el cas general o més habitual, que és el cas 9, on trobem **la casella que pot tenir com a màxim 8 mines al voltant**. Aquest cas és essencial ja que la majoria de les caselles del tauler estan dins d'aquest cas.

Per tant, utilitzem aquest Mock per posar les mines on nosaltres vulguem per comprovar i, a més, seguim el principi de les particions equivalents amb els diferents casos que tenim dins del Mock.

ConstructorInicialTest:

→ Aquest és un test simple que comprova la correcta generació del Tauler i dels seus atributs. Mitjançant dos bucles anidats comprovem que totes les caselles s'han inicialitzat amb els atributs de mina, bandera i destapat en false. Per últim, comprovem que no es pot declarar un Tauler sense un objecte random:

```
@Test
void ConstructorInicialTaulerTest() {
    //Comrpovem que el tauler inicial s'inicialitza correctament amb totes
    r1 = new Random();
    t1 = new Tauler(r1);
    for(int i=0; i < 13; i++) {
        for(int j=0; j < 13; j++){
            assertFalse(t1.isBandera(i,j));
            assertFalse(t1.isMina(i,j));
            assertFalse(t1.isDestapat(i,j));
        }
    }

    // Cas on inicialitzem el random del tauler en null (no s'ha de poder)
    try{
        t1 = new Tauler(null);
    }catch(AssertionError e) {
        System.err.println(e.getMessage());
    }
}
```

GetCasellaTest:

Aquest test del getter de la casella que correspon amb les coordenades passades per paràmetres és un dels tests que ens ha permès aprofitar i estalviar més temps, ja que molts mètodes utilitzen el mètode getCasella. **En aquest test comprovem que es compleixin els límits del tauler en qualsevol crida que es faci a aquest mètode**. Per comprovar això hem utilitzat 4 arrays: 2 amb les combinacions de les coordenades correctes dins de la partició (0 - 12) i 2 arrays més amb les combinacions dels valors incorrectes de la partició. Realitzant així una mena de "PairWise Testing" amb els valors de les particions (tot i que sabem que no es pot considerar pairwise, ja que només tenim dos paràmetres d'entrada, però, com hem comentat abans, en el nostre projecte no es pot fer cap cas de Pairwise).

```
@Test
void getCasellaTest() {
    // Test del GetCasella, mètode que apartir de dos parametres d'entrada (x i y) retorna la casella corresponent
    // a la coordenada indicada pels paràmetres d'entrada dins del tauler.
    // I comprova que el mètode getCasella fa un control correcte dels límits del tauler.
    // Utilitzant testing amb particions equivalents, no es pot utilitzar Pairwise Testing ja que com el mètode
    // reb només 2 parametres no es possible realitzar el pairwise ja que el resultat són totes les combinacions.
    r1 = new Random();
    t1 = new Tauler(r1);
    int []valors_x = new int [] {0,0,0,0,0,1,1,1,1,12,12,12,12,12,11,11,11,11,11,6,6,6,6,6};
    int []valors_y = new int [] {0,1,12,11,6,0,1,12,11,6,0,1,12,11,6,0,1,12,11,6,0,1,12,11};

    for(int i = 0; i < valors_x.length;i++)
            assertsGetCasella(valors_x[i],valors_y[i]);

    valors_x = new int [] {0,0,1,1,-1,-1,-1,-1,-1,-1,12,12,13,13,13,13,13,13,11,11,6,6,-1,5,13,7};
    valors_y = new int [] {-1,13,-1,13,0,1,-1,12,13,11,-1,13,0,1,12,13,11,6,-1,13,-1,13,5,-1,7,13};

    for(int i = 0; i < valors_x.length;i++)
        llançarExcepcionsCasella(valors_x[i],valors_y[i]);

}
```

Com es pot veure a la imatge d'adalt, utilitzem dos mètodes auxiliars de la classe TaulerTest que són **assertsGetCasella** i **llançarExcepcionsCasella**. Aquests mètodes privats auxiliars ens ajuden a no repetir tantes línies de codi i a gestionar millor els asserts. Són aquests:

```
//Realitzem una funció per llançar les excepcions del mètode getCasella de quan les coordenades no estan a dins del tau
private void llançarExcepcionsCasella(int x, int y){
    try{
        t1.getCasella(x, y);
        assertTrue(false);
    }catch(Exception e){
        System.err.println(e.getMessage());
    }
}
```

```
//Fem el mateix per els asserts de totes els mètodes anteriors
private void assertsGetCasella(int x, int y) {
    assertTrue(t1.getCasella(x, y) instanceof Casella); //Simplement ha de retornar una i
}
```

GeneraMinesRandomTest:

Aquest test comprova que la generació de les mines ha sigut correcta, és a dir, a les caselles del voltant a on ha clicat l'usuari no hi ha cap mina i el tauler té les 30 mines generades correctament. A més es realitza un loop testing simple ja que es comprova el correcte funcionament del bucle while de la funció generaMinesRandom del codi desenvolupat, comprovant els casos amb 0 iteracions(mines), iteracions mínimes, valors típics i valor màxim de mines. En aquest test també utilitzem les combinacions dels valors de la partició equivalents (0 - 12) correctes, és a dir valors frontera, valors límit (de dins de la partició) i valors de dins de la partició. I també les combinacions dels valors incorrectes com els valors límit de fora de la partició. Cridem al mètode generaMinesRandom amb totes les caselles de l'esquema del tauler indicat adalt (9 casos) i després fem ús de dos arrays per fer les combinacions de valors incorrectes:

```
@Test
void generaMinesRandomTest() {
    r1 = new Random();
    t1 = new Tauler(r1);
    t1.generaMinesRandom(0,0);
    assertFalse(t1.isMina(0,0));
    assertFalse(t1.isMina(1,0));
    assertFalse(t1.isMina(0,1));
    assertFalse(t1.isMina(1,1));

    te30Mines();

    t1 = new Tauler(r1);

    t1.generaMinesRandom(0,12);
    assertFalse(t1.isMina(0,12));
    assertFalse(t1.isMina(1,12));
    assertFalse(t1.isMina(1,11));
    assertFalse(t1.isMina(0,11));

    te30Mines();
```

```
t1 = new Tauler(r1);

t1.generaMinesRandom(12,0);
assertFalse(t1.isMina(12,0));
assertFalse(t1.isMina(11,0));
assertFalse(t1.isMina(12,1));
assertFalse(t1.isMina(11,1));

te30Mines();

t1 = new Tauler(r1);

t1.generaMinesRandom(12,12);
assertFalse(t1.isMina(12,12));
assertFalse(t1.isMina(11,11));
assertFalse(t1.isMina(12,11));
assertFalse(t1.isMina(11,12));

te30Mines();
```

```
t1 = new Tauler(r1);

t1.generaMinesRandom(0,6);
assertFalse(t1.isMina(0,6));
assertFalse(t1.isMina(0,5));
assertFalse(t1.isMina(0,7));
assertFalse(t1.isMina(1,5));
assertFalse(t1.isMina(1,6));
assertFalse(t1.isMina(1,7));

te30Mines();


t1 = new Tauler(r1);

t1.generaMinesRandom(6,0);
assertFalse(t1.isMina(6,0));
assertFalse(t1.isMina(5,0));
assertFalse(t1.isMina(7,0));
assertFalse(t1.isMina(5,1));
assertFalse(t1.isMina(6,1));
assertFalse(t1.isMina(7,1));

te30Mines();
```

```
t1 = new Tauler(r1);

t1.generaMinesRandom(6,6);
assertFalse(t1.isMina(6,6));
assertFalse(t1.isMina(5,5));
assertFalse(t1.isMina(5,6));
assertFalse(t1.isMina(5,7));
assertFalse(t1.isMina(6,5));
assertFalse(t1.isMina(6,7));
assertFalse(t1.isMina(7,5));
assertFalse(t1.isMina(7,6));
assertFalse(t1.isMina(7,7));

te30Mines();

//Casos de prova que llencen excepcions (fora dels limits del tauler):
int []valors_x = new int [] {0,0,1,1,-1,-1,-1,-1,-1,-1,12,12,13,13,13,13,13,13,11,11,6,6,-1,5,13,7};
int []valors_y = new int [] {-1,13,-1,13,0,1,-1,12,13,11,-1,13,0,1,12,13,11,6,-1,13,-1,13,5,-1,7,13};

for(int i = 0; i < valors_x.length;i++)
    llançarExcepcionsGeneraMines(valors_x[i],valors_y[i]);
```

Com es pot veure, fem ús de dos mètodes auxiliars per ajudar-nos a tractar les excepcions, els asserts i no repetir tantes línies de codi iguals (de la mateixa manera que en el test anterior). En aquest cas, trobem les funcions te30Mines, que comprova que després de cridar a generaMinesRandom el tauler té les 30 mines que li corresponen, i també el mètode llançarExcepcionsGeneraMines, que tracta les excepcions de les combinacions de valors incorrectes:

```
//Mètode privat per comprovar si al tauler hi han 10 mines.
private void te30Mines(){
    int n_mines = 0;
    for(int i=0;i<13;i++)
        for(int j=0;j<13;j++){
            if(t1.getCasella(i,j).isMina())
                n_mines ++ ;
            if(n_mines == 30) {
                assertTrue(true);
                break;
            }
        }

    if(n_mines < 30)
        assertTrue(false);
}
```

```
//Realitzem una funció per llançar les excepcions del mètode generaMinesRandom
private void llançarExcepcionsGeneraMines(int x, int y) {
    try{
        t1.generaMinesRandom(x, y);
        assertTrue(false);
    }catch(Exception e){
        System.err.println(e.getMessage());
    }
}
```

SetNumMinesVoltantTest:

Aquest test comprova el número de mines al voltant de cada casella (casos especials i regulars) sigui correctament generat. Les caselles que comprovem son les mateixes que les del esquema del tauler: (0,0), (6,0), (12,0), (12,6), (12,12), (6,12), (0,12), (0,6) i (6,6). Amb aquests casos tindrem comprovades les cantonades, els laterals i les caselles regulars del tauler amb tots els nombres d emines posibles. Recordem que el número de mines pot ser de 0 a 8, però que depenent de la casella pot ser 3 (cantonada) o 5 (lateral).
En aquest test realitzem les particions equivalents corresponents: Per les cantonades comprovem els valors frontera (0 i 3) i valors limit interiors (1 i 2). Per els laterals comprovem valors frontera (0 i 5), valors limit interiors (1 i 4) i un del mig (2 o 3) Per les caselles amb 8 caselles adjacents comprovem valors frontera (0 i 8), valors límits interiors (1 i 7) i un del mig (2, 3, 4, 5 o 6) No comprovem valors com el -1 perquè és impossible que el comptador de mines al voltant generi aquest tipus de valor.
Cal remarcar que en aquest test la crida al mètode generaMinesRandom no té cap rellevància en el resultat del test però és necessàri per poder fer els següents passos del test. Per que no hi hagi cap error hem de clicar a generarMinesRandom a una casella llunyana d'on posarem les mines perquè si no no funcionara correctament per com està fet el codi desenvolupat. Aquest test també realitza 2 loops testing anidats ja que cobreix explícitament els bucles for anidats del mètode setNumMinesVoltant i de la funció privada comptarNumMines del codi desenvolupat, comprovant cantonades, laterals i caselles centrals amb tots els valors límit.

Per aquest test hem utilitzat el Mock Object de Random (MockRandom) per posar les mines on nosaltres vulguem per poder fer el testing correctament:

```
// Casella 0,0 (cantonada adalt esquerra):
// Generem els següents casos de mines al voltant de la casella 0,0 a traves d'un bucle:
for(int i=-1;i<3;i++) {
    r1 = new MockRandom(0,i);
    t1 = new Tauler(r1);
    t1.setMaxMines(i+1);
    t1.generaMinesRandom(11,0);
    t1.setNumMinesVoltant();
    assertEquals(i+1, t1.getNumMinesVoltant(0,0));
}

//Casella 6,0
// Generem els següents casos de mines al voltant de la casella 6,0 a traves d'un bucle:
for(int i=-1;i<5;i++) {
    r1 = new MockRandom(1,i);
    t1 = new Tauler(r1);
    t1.setMaxMines(i+1);
    t1.generaMinesRandom(11,0);
    t1.setNumMinesVoltant();
    assertEquals(i+1, t1.getNumMinesVoltant(6,0));
}
```

Com podem observar hem emprat bucles for per accedir als casos de l'array de 4 dimensions del Mock de Random per evitar repetir massa codi.

DestapaCasellaTest:

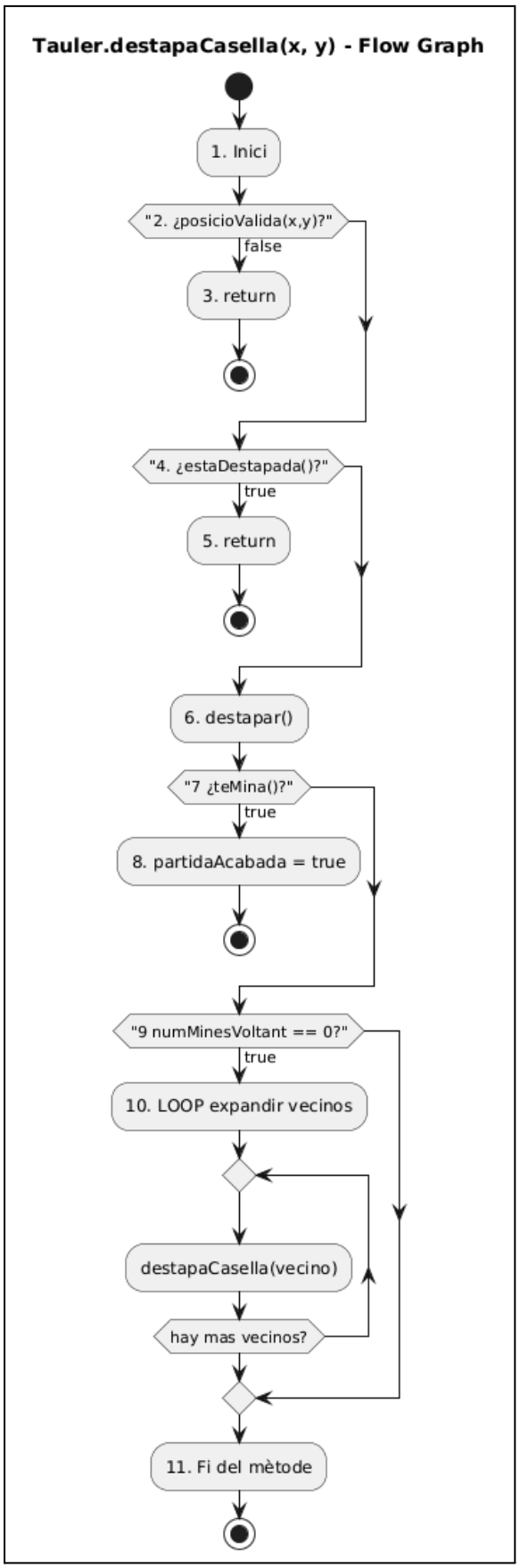
Aquest test comprova que una casella n'ha estat destapada correctament (quan tocava). Segons les circumstàncies, hi ha caselles que no s'han de destapar, que el seu destapament ha de provocar que es destapin altres caselles adjacents o que s'acabi la partida. Es realitza loop testing anidat, ja que es comprova el correcte funcionament del doble bucle for de destapaCasella. En aquest test també farem ús del Mock Object de Random per posar les mines a les posicions que nosaltres vulguem. També és un dels 2 mètodes que hem escollit per fer el path coverage, degut a la bona estructura de condicionals que té. Aquest és el codi de la funció desenvolupada:

```
public void destapaCasella(int x, int y) {
    Casella c = getCasella(x, y); // Comprova límits i invariant.

    // Si ja està destapada o té bandera, no fem res.
    if (c.isDestapat() || c.isBandera()) {
        return;
    }

    // Destapem la casella.
    c.destaparCasella();

    // Si és una mina, finalitzem (game over).
    if (c.isMina()) {
        return;
    }

    // Si no hi ha mines al voltant, destapem recursivament les caselles adjacents.
    if (c.getNumMinesVoltant() == 0) {
        for (int dx = -1; dx <= 1; dx++) {
            for (int dy = -1; dy <= 1; dy++) {
                if (dx == 0 && dy == 0)
                    continue;

                int nx = x + dx;
                int ny = y + dy;

                if (nx >= 0 && nx < MIDA && ny >= 0 && ny < MIDA) {
                    destapaCasella(nx, ny);
                }
            }
        }
    }
    invariant();
}
```

Com es pot apreciar, hem utilitzat la recursivitat per aconseguir la funcionalitat del destapament de les caselles adjacents a una casella amb cap mina al voltant o buida.

Després d'analitzar el codi desenvolupat detalladament hem trobat aquests paths dintre del mètode:
Path 1 — Coordenadas fora de rang → excepció
Path 2 — Casella ja destapada
Path 3 — Casella amb mina → es destapa y s'acaba
Path 4 — Casella sense mina i amb número > 0
Path 5 — Casella amb 0 mines → expansió recursiva

A continuació, mostrem el diagrama (flow graph) realitzat amb l'eina PlantUML:



A continuació mostrem el testing corresponent a cada path:

```java
// Path 1:
Tauler t = new Tauler(new MockRandom(1,0));
try {
    t.destapaCasella(-1, 5); // Intentem destapar una casella fora de rang.
}catch(Exception e) {
    System.err.println(e.getMessage());
}

// Path 2:
t = new Tauler(new MockRandom(1,0));
t.destapaCasella(3, 3);;  // destapem la casella 3,3
t.destapaCasella(3, 3);  // No hauria de fer res
assertTrue(t.isDestapat(3, 3));
```

```java
// Path 3:
t = new Tauler(new MockRandom(1, 0));
t.setMaxMines(1);
t.generaMinesRandom(0, 0);
t.destapaCasella(5, 0);
assertTrue(t.isDestapat(5, 0)); // Comprovem que esta destapada
assertTrue(t.isMina(5, 0));

// Path 4:
t = new Tauler(new MockRandom(1, 0)); // Posem una mina a la posició 5,0
t.setMaxMines(1);
t.generaMinesRandom(0, 0);
t.setNumMinesVoltant();
t.destapaCasella(5, 1); // Destapem un veí
assertTrue(t.isDestapat(5, 1));
assertEquals(1, t.getNumMinesVoltant(5, 1));

// Path 5:
t = new Tauler(new Random());
// Cap mina → tots haurien d'expandir-se
t.setNumMinesVoltant();
t.destapaCasella(6, 6);
assertTrue(t.isDestapat(6, 6));
assertTrue(t.isDestapat(5, 6));
assertTrue(t.isDestapat(7, 6));
assertTrue(t.isDestapat(6, 5));
assertTrue(t.isDestapat(6, 7));
```

Amb aquests casos de test complim el path coverage del mètode. També cal mencionar que el statement coverage, condition coverage i el decision coverage de la classe Tauler és del 100%. L'únic problema és el mateix de la classe Casella, amb alguns asserts de les postcondicions després de l'assignació de valors a setters o als constructors, línies que no són accessibles però que ens asseguren que el valor indicat es posa correctament a la variable.

TotesDestapadesSenseMinesTest:

Aquest test comprova que la condició de guanyar la partida s'activa correctament en diferents casos. Els casos comprovats en aquest test són: no hi ha cap mina al tauler i totes estan destapades (retorna true), cas on les mines estan totes tapades (retorna false), cas on una mina està destapada (retorna false), cas on les mines estan tapades i les altres destapades (retorna true), cas on totes les caselles són mines menys on cliquem (retorna true), cas on estan totes destapades menys una que no té mina (retorna false).
Es podria considerar que es realitza un loop testing simple a més d'anidat, ja que es comprova el correcte funcionament del doble bucle for, i si consideres el return false, es podria considerar un loop testing simple amb sortida anticipada.

```
@Test
public void totesDestapadesSenseMinesTest() {
    //Funció que comprova si ha guanyat la partida o no.
    // Cas on estan totes tapades (retorna false)
    r1 = new Random();
    t1 = new Tauler(r1);
    assertFalse(t1.totesDestapadesSenseMines()); // Inicialment cal que cap estigui destapada.

    // Cas on estan totes destapades i sense mines (returnoa true)
    t1.destapaCasella(0, 0); // Destapem una casella i com no te cap mina el tauler es destapen totes.
    assertTrue(t1.totesDestapadesSenseMines()); // Després de destaparles totes sense mines cal que ens retorni true.

    // Cas on hiha una mina destapada, game over (retorna false)
    r1 = new MockRandom(1, 0);
    t1 = new Tauler(r1);
    t1.setMaxMines(1);
    t1.generaMinesRandom(0, 0);
    t1.setNumMinesVoltant();
    t1.destapaCasella(5, 0);
    assertFalse(t1.totesDestapadesSenseMines());

    // Cas on les mines estan tapades i les altres destapades (retorna true)
    r1 = new MockRandom(1, 1);
    t1 = new Tauler(r1);
    t1.setMaxMines(2);
    t1.generaMinesRandom(0, 0);
    t1.setNumMinesVoltant();
    t1.destapaCasella(0, 0);
    assertTrue(t1.totesDestapadesSenseMines());
```

Com es pot observar utilitzem el MockRandom per inicialitzar les mines al teuler on ens va més bé per poder realitzar el testing adequat.

SetMaxMinesTest:

Aquest és un test simple que comprova que el nombre màxim de mines que guarda la classe Tauler s'ha inicialitzat correctament. Aquest test, encara que sembli simple, és molt important fer-ho ja que l'atribut de la classe Tauler nMaxMines ajusta el nombre d'iteracions que hauria de fer el mètode generaMinesRandom.

```
@Test
void setMaxMinesTest() {
    // Comprovar assert de nMaxMines >= 0
    Tauler tNegatiu = new Tauler(new Random());
    try {
        tNegatiu.setMaxMines(-1);
    }catch(AssertionError e) {
        System.err.println(e.getMessage());
    }
}
```

DestaparCasellesAmbMines:

Test simple que comprova que es destapen correctament les caselles que tinguin una mina i que estiguin tapades (si està ja destapada evita tornar-la a destapar si no salta excepció). Evidentment, també es realitza test loop dels bucles anidats. Aquest test ens permet comprovar el mètode que utilitzem per revelar totes les mines del tauler un cop ha perdut l'usuari. Utilitzem el MockRandom per posar les mines i després comprovem que, si cridem al mètode destapaCasellesAmbMines, funciona correctament:

```
@Test
void destaparCasellesAmbMines() {
    //Comprovem que es destapen las casellas que tinguin una mina correctament
    r1 = new MockRandom(2, 1); //Posem mines a les posicions 12,1 i 11,0
    t1 = new Tauler(r1);
    t1.setMaxMines(2);
    t1.generaMinesRandom(0, 0);
    t1.destaparCasellesAmbMines();
    //Comprovem que s'han destapat correctament les caselles que tenen mina.
    assertTrue(t1.isDestapat(12, 1));
    assertTrue(t1.isDestapat(11, 0));
    //Segona comprovacio si una mina ja esta destapada no dona error.
    r1 = new MockRandom(2, 1); //Posem mines a les posicions 12,1 i 11,0
    t1 = new Tauler(r1);
    t1.setMaxMines(2);
    t1.generaMinesRandom(0, 0);
    t1.destapaCasella(12, 1); //Destapem la casella amb mina
    t1.destaparCasellesAmbMines(); //Destapem totes les caselles amb mina i hauria de

}
```

JocTest:

Ara passem a l'explicació del testing realitzat al controlador del nostre projecte, la classe Joc. Aquest fitxer de test compta amb un total de 4 tests que comproven el correcte funcionament del controlador, on realitzem el segon cas de path coverage amb el mètode clicEsquerra, utilitzem 2 Mock Objects més (MockTauler i MockVista) i altres casos.

Cal remarcar que en el nostre projecte comptem amb un total de 3 Mock Objects, i no hem pogut implementar-ne cap altre, ja que amb els 3 que tenim són suficients per realitzar totes les comprovacions necessàries que es demanen en el projecte i, més concretament, en el joc del Buscamines. Aquest aspecte ja l'hem comentat múltiples vegades amb el nostre professor de pràctiques, en Xavier Otazu, i ens ha comentat que ho comentéssim a l'informe, ja que compartia el mateix punt de vista que nosaltres: que amb 3 Mocks era suficient per comprovar tot el necessari del nostre Buscamines.

Mock Objects:

MockBuscaminesVista:

Aquest és el segon Mock Object realitzat en el nostre projecte, és el més simple dels 3. La seva funció principal és substituir la vista, per evitar que s'obri a l'hora de realitzar els tests

que utilitzen la vista. Compta amb els mateixos mètodes que la vista, i, per exemple, el mètode més important de la vista real (actualitzar) està buit, ja que el Mock no ha de fer res.

```java
package vista;

import controlador.Joc;

public class MockBuscaminesVista extends BuscaminesVista{

    //Mock object per a quan realitzem els test de joc no ens mostri la vista, ja que no és necessària.
    public MockBuscaminesVista(Joc joc) {
        super(joc);
    }
    //Aquesta funció no fa res ja que no ha d'iniciar cap vista
    public void initVista() {

    }
    //Aquesta funció no fa res perquè només volem que no es mostri la vista quan realitzem els tests de joc
    public void actualitzar() {
    }
}
```

MockTauler:

Aquest és l'últim Mock implementat en el nostre projecte, la seva funció és substituir el Tauler. Aquest mock realitza un posicionament de 30 mines per poder realitzar una simulació dels clics del jugador, ja siguin clics esquerres o clics drets. D'aquesta manera, podem realitzar un testing adequat i "real" d'una partida completa del Buscamines, on testegem les diferents funcionalitats del joc/controlador. El mock substitueix la funció generaMinesRandom de la classe Tauler, per poder realitzar aquest posicionament de les mines que comentem. Encara que aquest Mock està al package de Model, s'utilitza en el testing del controlador per utilitzar-lo amb el Joc (que és la classe superior de Tauler).

```java
package model;

import java.util.Random;

public class MockTauler extends Tauler{

    // Mock object de la classe Tauler per facilitar els casos de prova que requereixen generar taulers específics.
    // Com que la generació de mines és aleatòria en la classe original, necessitem taulers amb mines
    // posicionades manualment per poder verificar el funcionament d'altres mètodes.

    public MockTauler(Random r) {
        super(r);
    }
```

```java
@Override
public void generaMinesRandom(int xPlayer, int yPlayer) {
    // Override de la funció generaMinesRandom perquè no sigui aleatòria.
    // Això ens permet controlar exactament on es col·loquen les mines i així realitzar proves deterministes.
    // Utilitzem el paràmetre xPlayer per seleccionar diferents configuracions predefinides de mines.
    // En aquest cas, només fem servir la configuració 1, que genera un tauler fix de 30 mines.

    switch(xPlayer) {
    case 1: // Posició del tauler 1: mines predefinides per un cas de prova on el jugador pot guanyar.
        tauler[0][0].setMina();
        tauler[1][1].setMina();
        tauler[1][3].setMina();
        tauler[1][4].setMina();
        tauler[1][5].setMina();
        tauler[1][9].setMina();
        tauler[1][10].setMina();
        tauler[1][11].setMina();
        tauler[1][12].setMina();
        tauler[2][10].setMina();
        tauler[3][1].setMina();
        tauler[3][4].setMina();
        tauler[4][11].setMina();
        tauler[5][2].setMina();
        tauler[5][6].setMina();
        tauler[5][9].setMina();
        tauler[6][1].setMina();
        tauler[6][2].setMina();
        tauler[7][4].setMina();
        tauler[7][11].setMina();
        tauler[8][7].setMina();
        tauler[10][0].setMina();
        tauler[10][2].setMina();
        tauler[10][9].setMina();
        tauler[11][0].setMina();
        tauler[12][0].setMina();
        tauler[12][1].setMina();
        tauler[12][5].setMina();
```

Ara si, comencem amb l'explicació del testing realitzat a JocTest:

TestConstructorPerParametresJoc:

Aquest test realitza comprovacions d'inicialització del joc. La classe Joc compta amb els atributs següents: un Tauler, partidaAcabada (boolean), la Vista del Buscamines i el nombre de banderes restants (int). En aquest test comprovem que no es pot inicialitzar un Joc sense Tauler i que no podem crear la vista del joc sense una vista definida. També utilitzem el Mock Object de la vista explicat anteriorment per evitar obrir la vista real quan executem els tests.

```java
@Test
public void TestConstructorPerParàmetresJoc() {
    Random r = new Random();
    Tauler t1 = new Tauler(r);

    // Constructor per paràmetres: comprovem que s'inicialitza correctament.
    // Això es comprova mitjançant les postcondicions del codi. Si no es llença cap error, és correcte.
    Joc joc = new Joc(t1);

    // Creem un mock de la vista per evitar obrir la vista real.
    BuscaminesVista vista  = new MockBuscaminesVista(joc);
    vista.initVista(); // El mock no fa res.

    // Assignem la vista al controlador. Es comprova mitjançant postcondicions.
    joc.crearVistaDelJoc(vista);

    // Comprovem les excepcions del constructor (tauler null) i del mètode crearVistaDelJoc (vista null),
    // controlades mitjançant precondicions al codi.
    try {
        joc = new Joc(null);
    } catch (AssertionError e) {
        System.err.println(e.getMessage());
    }

    try {
        joc.crearVistaDelJoc(null);
    } catch (AssertionError e) {
        System.err.println(e.getMessage());
    }
}
```

ClicDretTest:

Test que comprova el correcte funcionament del mètode del controlador Joc, clicDret. Aquest mètode té com a objectiu permetre posar una bandera al tauler de la partida. On comprovem els possibles casos: posar bandera en una casella tapada (correcte), treure una bandera si ja n'hi havia una i no permetre posar una bandera en una casella ja destapada anteriorment. En aquest test també utilitzem el Mock Object de la vista, per evitar obrir la vista real a l'hora d'executar els tests.

```java
@Test
public void clicDretTest() {

    // El clic dret permet posar o treure una bandera.
    // Casos possibles:
    // 1. Posar bandera en una casella tapada
    // 2. Treure bandera si ja n'hi havia
    // 3. No permetre bandera en una casella destapada

    // Cas 1: Posar bandera correctament
    Random r = new Random();
    Tauler t1 = new Tauler(r);
    Joc joc = new Joc(t1);

    BuscaminesVista vista  = new MockBuscaminesVista(joc);
    vista.initVista();
    joc.crearVistaDelJoc(vista);

    joc.clicDret(2, 2);
    assertEquals(29, joc.getBanderesRestants());
    assertTrue(joc.isBandera(2,2));

    // Cas 2: Treure bandera
    joc.clicDret(2, 2);
    assertEquals(30, joc.getBanderesRestants());
    assertFalse(joc.isBandera(2,2));
```

Fem comprovacions del nombre de banderes, si la bandera s'ha col·locat correctament i també si s'ha tret correctament.

```java
// Cas 3: No permetre bandera en una casella destapada
r = new Random();
t1 = new Tauler(r);
joc = new Joc(t1);

vista  = new MockBuscaminesVista(joc);
vista.initVista();
joc.crearVistaDelJoc(vista);

joc.clicEsquerra(2, 2); // Destapem
joc.clicDret(2, 2);

assertEquals(30, joc.getBanderesRestants());
assertFalse(joc.isBandera(2,2));
```

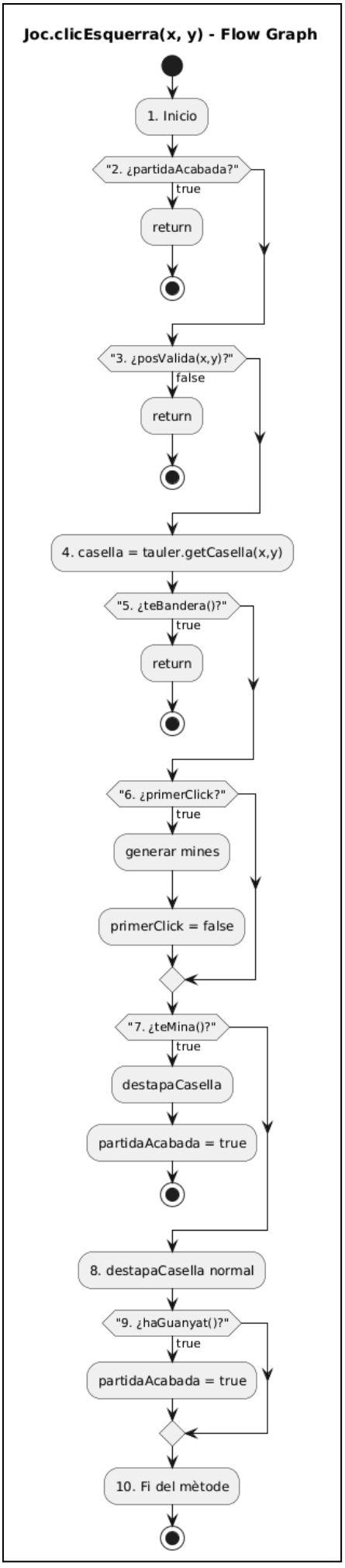
ClicEsquerraTest:

Test que comprova que el funcionament del mètode clicEsquerra de la classe Joc, que té com a objectiu destapar una casella si es pot, funciona correctament. En aquest test utilitzem el Mock de la vista, igual que en l'altre test. En aquest test també realitzem el path coverage, ja que el mètode clicEsquerra de la classe Joc compta amb una bona estructura de condicionals per realitzar el path coverage. Aquest és el mètode clicEsquerra:

```java
public void clicEsquerra(int x, int y) {

    // No podem interactuar si la partida ha acabat.
    if (this.partidaAcabada)
        return;

    // No podem destapar una casella amb bandera.
    if (this.tauler.isBandera(x, y))
        return;

    // Determinem si és el primer clic del jugador.
    boolean primerClic = true;

    for(int i = 0; i < 13; i++) {
        for(int j = 0; j < 13; j++) {
            if(this.tauler.isDestapat(i, j)) {
                primerClic = false;
                break;
            }
        }
    }

    // Si és el primer clic: generem mines garantint que la casella clicada no tingui mina.
    if (primerClic) {
        this.tauler.generaMinesRandom(x, y);
        this.tauler.setNumMinesVoltant();
    }

    // Destapem la casella indicada.
    this.tauler.destapaCasella(x, y);

    // Si hem clicat una mina → derrota.
    if (this.tauler.isMina(x, y)) {
        this.partidaAcabada = true;
        this.tauler.destaparCasellesAmbMines(); // Mostrem totes les mines.
    }
```

```java
        // Si totes les caselles sense mines han estat destapades → victòria.
        if (this.tauler.totesDestapadesSenseMines()) {
            this.partidaAcabada = true;
            this.tauler.destaparCasellesAmbMines(); // Mostrem també les mines.
        }
    }
```

Com es pot apreciar, hi ha un nombre considerable de condicionals per realitzar el path coverage. A continuació, mostrem el flow graph que hem utilitzat per realitzar el path coverage d'aquest mètode.


Joc.clicEsquerra(x, y) - Flow Graph
1. Inicio
"2. ¿partidaAcabada?"
true
return
"3. ¿posValida(x,y)?"
false
return
4. casella = tauler.getCasella(x,y)
"5. ¿teBandera()?"
true
return
"6. ¿primerClick?"
true
generar mines
primerClick = false
"7. ¿teMina()?"
true
destapaCasella
partidaAcabada = true
8. destapaCasella normal
"9. ¿haGuanyat()?"
true
partidaAcabada = true
10. Fi del mètode

Els camins únics que hem detectat són:

0. Casella invalida (cas comprovat en altres tests, no cal repetir codi) (1, 2, 3)
1. Partida ja acabada → no fa res (1, 2)
2. Casella amb bandera → no es pot destapar (1, 2, 4)
3. Primer clic → generar mines (1, 2, 4, 5, 6)
4. Casella ja destapada → no fa res (1, 2, 3)
5. Casella amb mina → perdre la partida (1, 2, 4, 5, 6, 7)
6. Últim clic per guanyar (test complet a simulacioPartida) (1, 2, 4, 5, 6, 7, 8, 9, 10)
   Aquest últim path es realitza en un test únic i especial per ell sol, ja que hem implementat una simulació completa d'una partida (data driven).

Testing realitzat:

```java
@Test
public void clicEsquerraTest() {

    // clicEsquerra destapa caselles.
    // Casos possibles:
    // 1. Partida ja acabada → no fa res
    // 2. Casella amb bandera → no es pot destapar
    // 3. Primer clic → generar mines
    // 4. Casella ja destapada → no fa res
    // 5. Casella amb mina → perdre la partida
    // 6. Últim clic per guanyar (test complet a simulacioPartida)

    // Path coverage: aquest mètode permet fer test per paths perquè té una estructura condicional clara.

    // Path 1 – partida acabada
    Random r = new Random();
    Tauler t1 = new Tauler(r);
    Joc joc = new Joc(t1);

    BuscaminesVista vista  = new MockBuscaminesVista(joc);
    vista.initVista();
    joc.crearVistaDelJoc(vista);

    joc.setPartidaAcabada();
    joc.clicEsquerra(0,0);
    assertTrue(joc.getPartidaAcabada());

    // Path 2 – casella amb bandera
    joc.clicDret(0, 0);
    joc.clicEsquerra(0, 0);
    assertFalse(joc.isDestapat(0, 0));
```

```java
// Path 3 – primer clic
r = new Random();
t1 = new Tauler(r);
joc = new Joc(t1);

vista  = new MockBuscaminesVista(joc);
vista.initVista();
joc.crearVistaDelJoc(vista);

joc.clicEsquerra(6,6);
assertTrue(joc.isDestapat(6, 6));
assertFalse(joc.getPartidaAcabada());

// Path 4 – casella ja destapada
joc.clicEsquerra(6, 6);
assertTrue(joc.isDestapat(6, 6));

// Path 5 – destapar una mina
r = new MockRandom(8, 7); // Mock que controla la mina
t1 = new Tauler(r);
t1.setMaxMines(1);

joc = new Joc(t1);
vista  = new MockBuscaminesVista(joc);
vista.initVista();
joc.crearVistaDelJoc(vista);

joc.clicEsquerra(0, 0);
joc.clicEsquerra(5, 5); // Casella amb mina

assertTrue(joc.isDestapat(12, 5));
assertTrue(joc.getPartidaAcabada());

// Path 6 – es comprova completament al test simulacioPartida()
```

Com es pot observar, també utilitzem el Mock Object de random per la generació de les mines al tauler.

SimulacioPartidaTest:

Test que realitza una simulació sencera d'una partida real del buscamines d'inici a fi. Realitzem una automatització (data driven) mitjançant dos fitxers txt on hem escrit les coordenades de les caselles que va clicant l'usuari i si és clic esquerra o clic dret. La simulació 1 representa una victòria (totes les caselles que no són mina estan destapades) i la simulació 2 representa una derrota (clica en una mina i perd la partida). Captura d'una de les dues simulacions per veure l'estructura i els casos de prova realitzats:

```
esq 1 2
esq 1 7
drt 1 5
drt 1 9
esq 0 5
esq 0 9
esq 10 12
drt 12 10
esq 8 2
drt 10 0
esq 10 1
drt 10 2
esq 10 3
esq 10 4
drt 8 7
```

```java
@Test
public void simulacioPartida() {

    // Test complet de la classe Joc.
    // Utilitza MockTauler per controlar la posició de les mines i MockBuscaminesVista
    // per evitar obrir la vista real.
    // Els fitxers simulacio1.txt i simulacio2.txt contenen una seqüència real de clics.
    // Cada línia indica "esq x y" o "drt x y".
    //
    // Al final:
    //   simulació 1 → ha de guanyar
    //   simulació 2 → ha de perdre
    //
    // Això comprova el Path 6 del mètode clicEsquerra (condició final de victòria o derrota).

    Random r = new Random();
    Tauler t1 = new MockTauler(r);
    t1.generaMinesRandom(10, 0);

    Joc joc = new Joc(t1);

    Vista vista  = new MockBuscaminesVista();
    vista.initVista();
    joc.crearVistaDelJoc(vista);
```

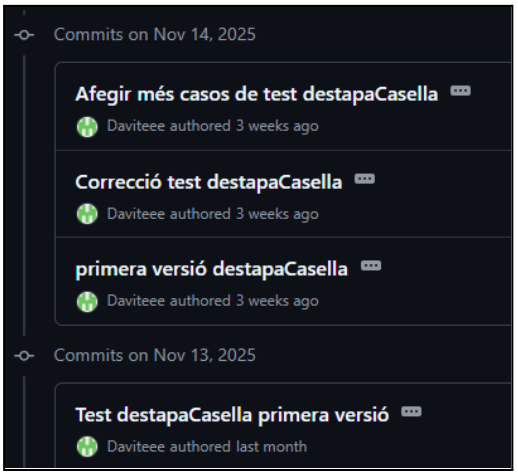
```java
// Simulació 1 → ha de guanyar
try (BufferedReader br = new BufferedReader(new FileReader("src/test/java/es/uab/tqs/buscamines/simulacio1.txt"))) {

    String linea;

    while ((linea = br.readLine()) != null) {
        String[] parts = linea.split(" ");

        String accion = parts[0];
        int x = Integer.parseInt(parts[1]);
        int y = Integer.parseInt(parts[2]);

        if (accion.equals("esq")) joc.clicEsquerra(x, y);
        else if (accion.equals("drt")) joc.clicDret(x, y);
    }

    assertTrue(t1.totesDestapadesSenseMines()); // Ha de guanyar

} catch (IOException e) {
    e.printStackTrace();
}
```

Llegim línia a línia el fitxer de text Simulacio1.txt per rebre les coordenades de cada casella i si és clic esquerre o dret. Finalment, després de realitzar totes les accions, comprovem si ha guanyat o ha perdut segons la simulació executada.
Amb aquest test comprovem el path número 6 del mètode clicEsquerra, que comprova que es pot guanyar la partida destapant l'última casella.

Criteris d'avaluació complerts i on trobar-los al codi:

Hem realitzat tot el testing i el codi desenvolupat necessari per aconseguir assolir tots els criteris d'avaluació que es demanen a la pràctica. Ara indicarem on es poden trobar cadascun d'ells:

1. **TDD:** El test driven development es pot veure al llarg de tota la pràctica; sempre que havíem d'implementar un mètode, es feia primer el test i després el codi desenvolupat, amb diverses versions a les dues bandes. Es pot comprovar als commits del repositori de gitHub:

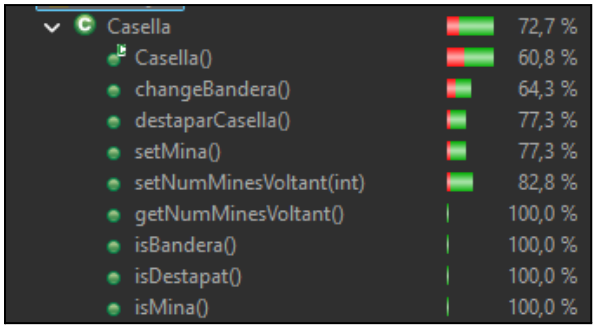


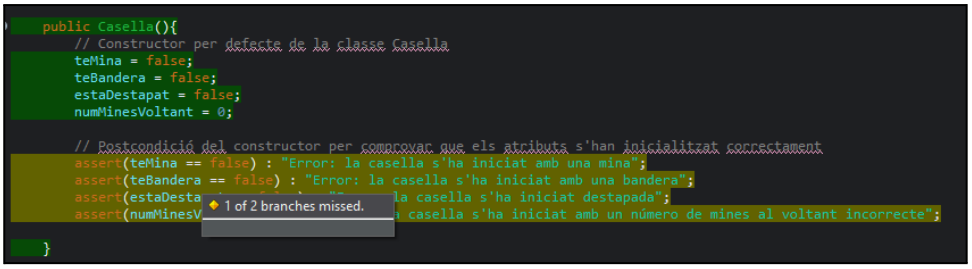
2. **Particions equivalents, valors límit i frontera i pairwise testing:** hem detectat, realitzat i testejat totes les particions equivalents possibles, les més destacables les de les combinacions de coordenades al testing de tauler amb els límits del tauler (particions equivalents: per les cantonades comprovem els valors frontera (0 i 3) i valors límit interiors (1 i 2). Pels laterals comprovem valors frontera (0 i 5), valors límit interiors (1 i 4) i un del mig (2 o 3). Per les caselles amb 8 caselles adjacents comprovem valors frontera (0 i 8), valors límit interiors (1 i 7) i un del mig (2, 3, 4, 5 o 6)) i casella amb el número de mines que pot tenir una casella al voltant (particions equivalents del domini: 0 - 8). Pel que fa al pairwise testing, ja hem comentat abans que no es pot realitzar, ja que cap mètode del buscamines rep més de 2 paràmetres, però, encara així, hem realitzat totes les combinacions possibles en tests com el de getCasellaTest. Es pot trobar el testing realitzat de particions equivalents i valors límit i frontera als test: setNumMinesVoltantTest (CasellaTest), getCasellaTest (TaulerTest), generaMinesRandomTest (TaulerTest) i setNumMinesVoltantTest (TaulerTest).

```
int []valors_x = new int [] {0,0,0,0,0,1,1,1,1,12,12,12,12,12,11,11,11,11,11,6,6,6,6,6};
int []valors_y = new int [] {0,1,12,11,6,0,1,12,11,6,0,1,12,11,6,0,1,12,11,6,0,1,12,11};

for(int i = 0; i < valors_x.length;i++)
        assertsGetCasella(valors_x[i],valors_y[i]);

valors_x = new int [] {0,0,1,1,-1,-1,-1,-1,-1,-1,12,12,13,13,13,13,13,13,11,11,6,6,-1,5,13,7};
valors_y = new int [] {-1,13,-1,13,0,1,-1,12,13,11,-1,13,0,1,12,13,11,6,-1,13,-1,13,5,-1,7,13};
```

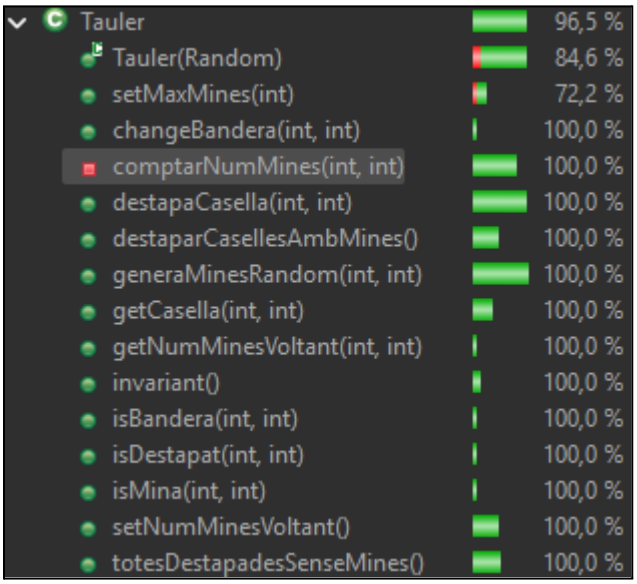
3. **Mock Objects:**Hem implementat un total de 3 mocks: el de Random, el de Tauler i el de la vista. Al PDF de requisits es comenta un total de 4, però, com hem comentat abans, el nostre professor de pràctiques, en Xavier Otazu, ens ha comentat que ho té en compte i que el buscamines es pot testejar sencer sense fer un quart mock. Els mocks implementats són MockRandom, MockTauler i MockVista.

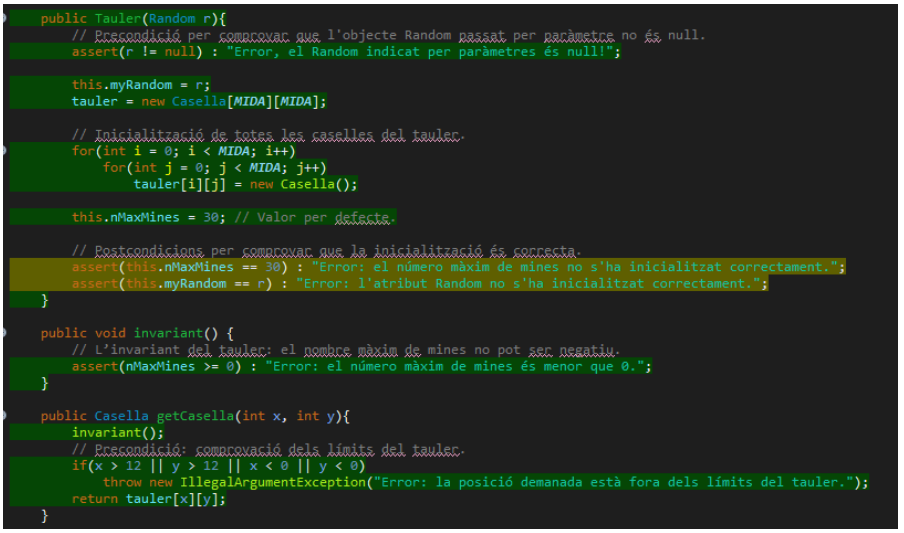
4. **StateMent coverage:**
   4.1 Casella: A la classe Casella hem aconseguit un statement coverage del 72,7%; el percentatge restant és degut a asserts inaccessibles, ja que són comprovacions de postcondicions en setters i en el constructor.



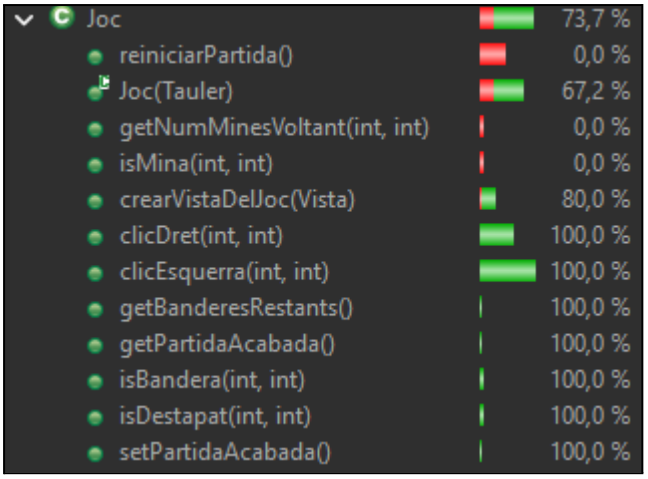


   4.2 Tauler: A la classe Tauler hem aconseguit un statement coverage del 96,5%, el percentatge restant es degut al mateix motiu que el de la classe Casella, els asserts inaccessibles dels setters i els constructors:





   4.3 Joc: Finalment, a la classe Joc hem aconseguit un 73,7% de statement coverage el percentatge restant resideix en el mateix motiu que les altres dos classes (asserts inaccessibles) i també en tres funcions que utilitza la vista i no testejem degut a que estan ja comprovades (2 getters (no fa falta comprovació) i reiniciar partida que mitjançant post condicions ja ho comprovem). Com son funcionalitats extres que només son necessàries per la vista, al test de joc no hem hagut de comprovar-les ja que no son ncessàries per veure el correcte funcionament del joc.

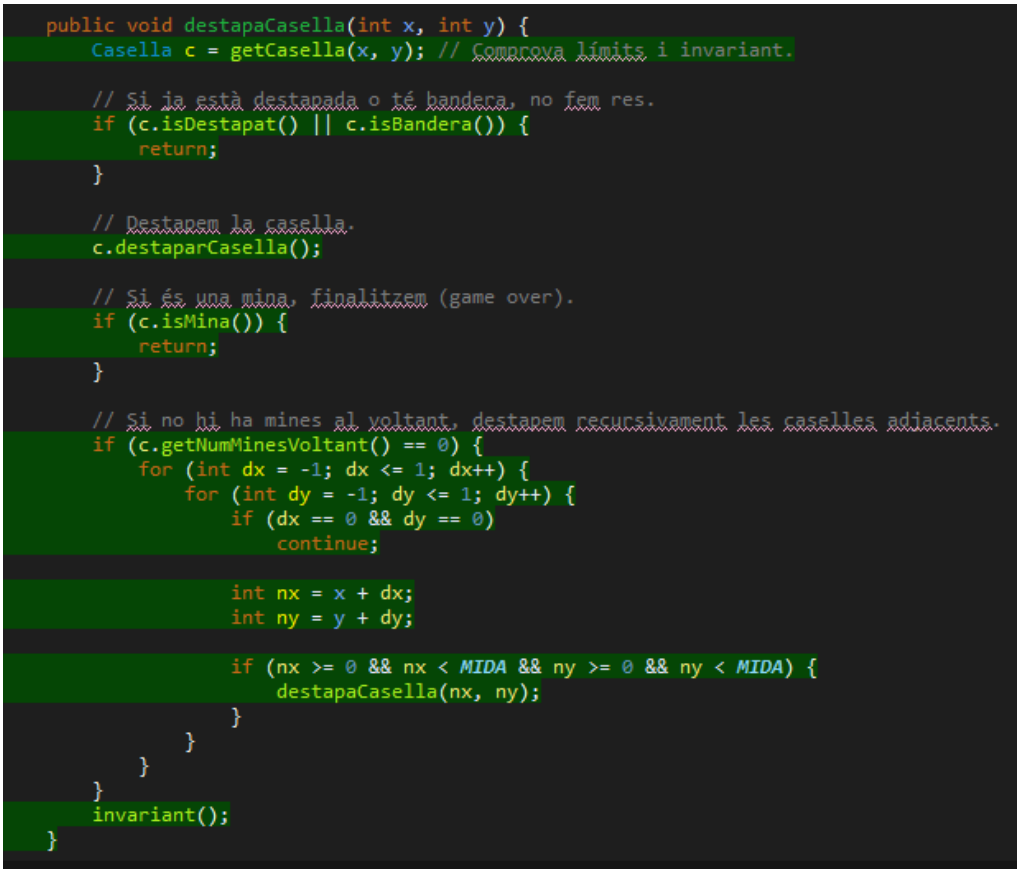



```
public Joc(Tauler t) {
    // Constructor per paràmetres on passem el tauler del joc.
    // Precondició per comprovar que el tauler no és null.
    assert(t != null) : "Error: el tauler és null";

    this.partidaAcabada = false;
    this.banderesRestants = 30;
    this.tauler = t;
    this.vista = null; // Inicialitzem la vista a null al constructor.

    // Postcondicions per comprovar que el joc s'ha inicialitzat correctament.
    assert(!this.partidaAcabada) : "Error: la partida s'ha inicialitzat com acabada (incorrecte)";
    assert(this.banderesRestants == 30) : "Error: el nombre de banderes no s'ha inicialitzat correctament";
    assert(this.tauler == t) : "Error: el tauler no s'ha inicialitzat correctament";
    assert(this.vista == null) : "Error: la vista no s'ha inicialitzat a null correctament";
}

public void crearVistaDelJoc(Vista vista) {
    // Funció que assigna la vista del joc al controlador. S'utilitza per poder instanciar un mock de vista.
    // Precondició per comprovar que la vista no és null.
    assert(vista != null): "Error: la vista és null";

    this.vista = vista;

    // Postcondició per comprovar que la vista s'ha instanciat correctament.
    assert(this.vista == vista): "Error: la vista no s'ha instanciat correctament";

    this.vista.actualitzar(); // Actualitza o immcimeix el tauler inicial (si és mock, no fa res).
}

public void reiniciarPartida() {
    // Funcionalitat extra per reiniciar el joc amb la mateixa vista.
    // Aquesta funció només la crida la vista, i actua com un "nou joc" sense necessitat de mocks,
    // ja que sempre es farà servir un tauler real i un Random real.
    // Les postcondicions comproven que la funció reinicia correctament l'estat del joc.

    Random r = new Random();
    this.tauler = new Tauler(r);
    this.partidaAcabada = false;
    this.banderesRestants = 30;

    assert(!this.partidaAcabada) : "Error: la partida s'ha reiniciat com acabada (incorrecte)";
    assert(this.banderesRestants == 30) : "Error: el nombre de banderes no s'ha reiniciat correctament";
}

public int getBanderesRestants() {
    return this.banderesRestants;
}

public boolean getPartidaAcabada() {
    return this.partidaAcabada;
}
```

5. **Decision i Condition coverage:** Els 3 mètodes que hem escollit per realitzar aquests dos tipus de coverage són: ClicEsquerra (Joc), DestapaCasella(Tauler) i ClicDret (Joc). A continuació les captures del codi amb l'eina coverage d'eclipse:

```java
public void clicEsquerra(int x, int y) {

    // No podem interactuar si la partida ha acabat.
    if (this.partidaAcabada)
        return;

    // No podem destapar una casella amb bandera.
    if (this.tauler.isBandera(x, y))
        return;

    // Determinem si és el primer clic del jugador.
    boolean primerClic = true;

    for(int i = 0; i < 13; i++) {
        for(int j = 0; j < 13; j++) {
            if(this.tauler.isDestapat(i, j)) {
                primerClic = false;
                break;
            }
        }
    }

    // Si és el primer clic: generem mines garantint que la casella clicada no tingui mina.
    if (primerClic) {
        this.tauler.generaMinesRandom(x, y);
        this.tauler.setNumMinesVoltant();
    }

    // Destapem la casella indicada.
    this.tauler.destapaCasella(x, y);

    // Si hem clicat una mina → derrota.
    if (this.tauler.isMina(x, y)) {
        this.partidaAcabada = true;
        this.tauler.destaparCasellesAmbMines(); // Mostrem totes les mines.
    }

    // Si totes les caselles sense mines han estat destapades → victòria.
    if (this.tauler.totesDestapadesSenseMines()) {
        this.partidaAcabada = true;
        this.tauler.destaparCasellesAmbMines(); // Mostrem també les mines.
    }
}
```

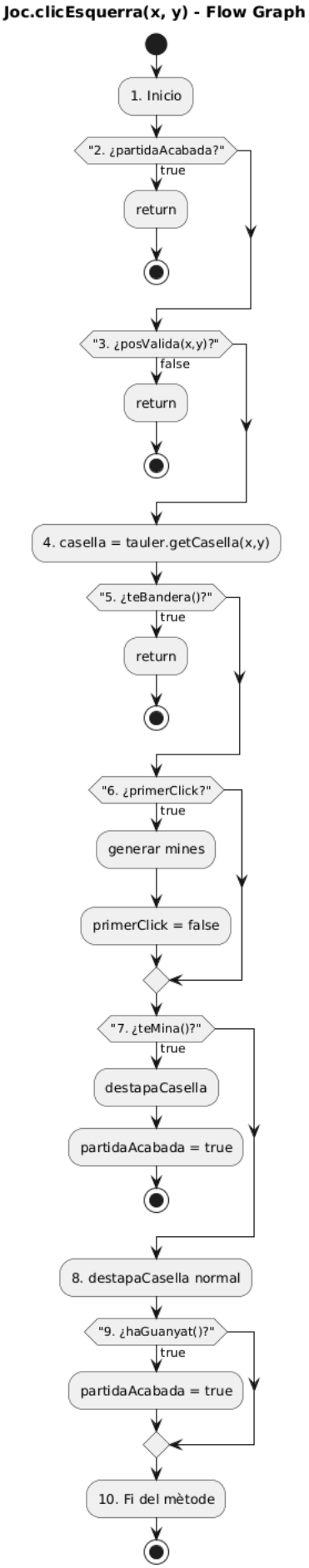
```java
public void clicDret(int x, int y) {

    // Si la partida ha acabat, no fem res.
    if (this.partidaAcabada)
        return;

    // No es poden posar banderes sobre caselles destapades.
    if (this.tauler.isDestapat(x, y))
        return;

    boolean teniaBandera = this.tauler.isBandera(x, y);

    // Si ja tenia bandera → la traiem i incrementem banderes restants.
    if (teniaBandera) {
        this.tauler.changeBandera(x, y);
        this.banderesRestants++;
    }
    else {
        // Si no queden banderes disponibles → no fem res.
        if (this.banderesRestants == 0) {
            return;
        }
        // Posem una bandera.
        this.tauler.changeBandera(x, y);
        this.banderesRestants--;
    }

}
```

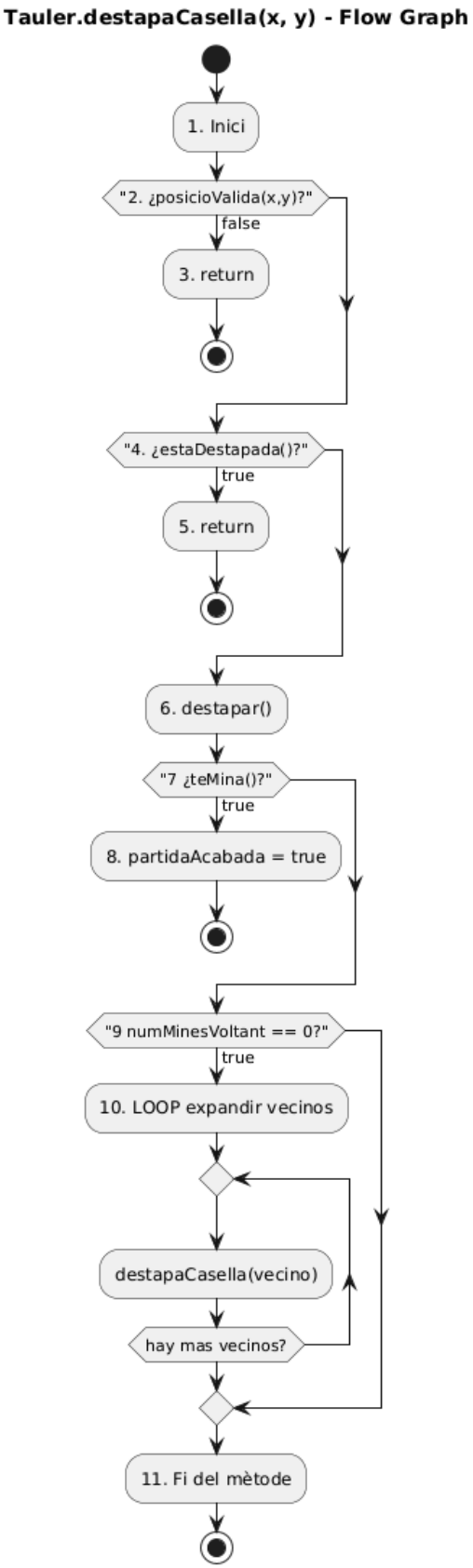


```java
public void destapaCasella(int x, int y) {
    Casella c = getCasella(x, y); // Comprova límits i invariant.

    // Si ja està destapada o té bandera, no fem res.
    if (c.isDestapat() || c.isBandera()) {
        return;
    }

    // Destapem la casella.
    c.destaparCasella();

    // Si és una mina, finalitzem (game over).
    if (c.isMina()) {
        return;
    }

    // Si no hi ha mines al voltant, destapem recursivament les caselles adjacents.
    if (c.getNumMinesVoltant() == 0) {
        for (int dx = -1; dx <= 1; dx++) {
            for (int dy = -1; dy <= 1; dy++) {
                if (dx == 0 && dy == 0)
                    continue;

                int nx = x + dx;
                int ny = y + dy;

                if (nx >= 0 && nx < MIDA && ny >= 0 && ny < MIDA) {
                    destapaCasella(nx, ny);
                }
            }
        }
    }
    invariant();
}
```

Com es pot apreciar, totes les possibles combinacions dels condicionals estan cobertes.

6. **Path Coverage**: Explicat anteriorment, captures dels diagrames:





7. **Loop Testing**: Explicat anteriorment. Els 2 loops simples són: El while de generaMinesRandom (Tauler) i el loop interior de totesDestapadesSenseMines ja que té un return false a dins. Y els 2 loops aniuats són: El doble for de destaparCasellesAmbMines (Tauler) i el doble for de clicEsquerra (Joc), es pot trobar el testing corresponent als tests dels mètodes mencionats anteriorment.

```
public void generaMinesRandom(int xPlayer, int yPlayer) {
    invariant();

    // Precondició: les coordenades del clic inicial han de ser dins dels límits.
    if(xPlayer > 12 || yPlayer > 12 || xPlayer < 0 || yPlayer < 0){
        throw new IllegalArgumentException("Les coordenades del clic de l'usuari són incorrectes (fora del tauler).");
    }

    int nMines = 0; // Nombre de mines col·locades fins al moment.

    // Itera fins que hi hagi nMaxMines mines col·locades correctament. nMaxMines serveix per quan només volem posar x mines en
    while(nMines < nMaxMines) {
        int xMina = myRandom.nextInt(13); // Generació de x aleatòria (0-12) o utilitzant el mockRandom que retorna una coordena
        int yMina = myRandom.nextInt(13); // Generació de y aleatòria (0-12) utilitzant el mockRandom que retorna una coordenada

        // No es poden col·locar mines en les 8 caselles adjacents al primer clic.
        if(Math.abs(xMina - xPlayer) <= 1 && Math.abs(yMina - yPlayer) <= 1) {
            continue;
        }

        // Si la casella no té una mina, la col·loquem.
        if(!tauler[xMina][yMina].isMina()) {
            tauler[xMina][yMina].setMina();
            nMines++;
        }
    }
    invariant();
}
```

```
public boolean totesDestapadesSenseMines() {
    invariant();
    // Comprova si totes les caselles sense mina e
    for (int i = 0; i < MIDA; i++) {
        for (int j = 0; j < MIDA; j++) {
            Casella c = getCasella(i, j);
            if (!c.isMina() && !c.isDestapat()) {
                invariant();
                return false;
            }
        }
    }
    invariant();
    return true;
}
```

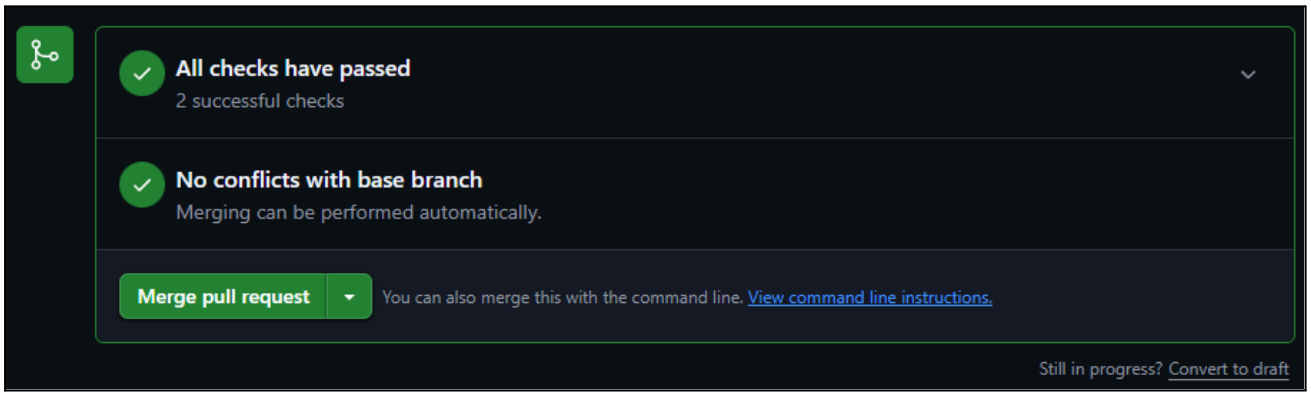
```
public void destaparCasellesAmbMines() {
    invariant();
    // Destapa totes les caselles que són mines, e
    for (int i = 0; i < MIDA; i++) {
        for (int j = 0; j < MIDA; j++) {
            Casella c = getCasella(i, j);
            if (c.isMina() && !c.isDestapat()) {
                c.destaparCasella();
            }
        }
    }
    invariant();
}
```

```
public void clicEsquerra(int x, int y) {

    // No podem interactuar si la partida ha acabat.
    if (this.partidaAcabada)
        return;

    // No podem destapar una casella amb bandera.
    if (this.tauler.isBandera(x, y))
        return;

    // Determinem si és el primer clic del jugador.
    boolean primerClic = true;

    for(int i = 0; i < 13; i++) {
        for(int j = 0; j < 13; j++) {
            if(this.tauler.isDestapat(i, j)) {
                primerClic = false;
                break;
            }
        }
    }

    // Si és el primer clic: generem mines garantint que la casella clicada no tingui mina.
    if (primerClic) {
        this.tauler.generaMinesRandom(x, y);
        this.tauler.setNumMinesVoltant();
    }

    // Destapem la casella indicada.
    this.tauler.destapaCasella(x, y);

    // Si hem clicat una mina → derrota.
    if (this.tauler.isMina(x, y)) {
        this.partidaAcabada = true;
        this.tauler.destaparCasellesAmbMines(); // Mostrem totes les mines.
    }

    // Si totes les caselles sense mines han estat destapades → victòria.
    if (this.tauler.totesDestapadesSenseMines()) {
        this.partidaAcabada = true;
        this.tauler.destaparCasellesAmbMines(); // Mostrem també les mines.
    }
}
```

8. **Codi amb comentaris**: Tot el codi tant desenvolupat com de test compta amb comentaris explicatius per entendre el codi perfectament.

```
class TaulerTest {

    Tauler t1;
    Random r1;

    // Comentari d'aclaració: Cal tenir en compte que totes les funcions que tinguin límits del tauler, es a dir els getters de la classe Tauler com isMina,isBandera...
    // TOTS utilitzen el mètode getCasella per fer la comprovació dels límits Tauler, per això mateix no fa falta realitzar test sobre ells ja que el mètode que està provat
    // i testejat és getCasella i com està testejat ja sabem que comprova correctament els límits del tauler. Per tant com tots els mètodes anteriorment mencionats ho utlitzen al codi desenvolupat
    // per comprovar els límits, sabem que aquests límits es comproven correctament. L'únic mètode que no tuilitza getCasella al codi desenvolupat és generaMinesRandom per aquest mateix
    // motiu en aquest mètode si es realitza la comprovació dels límits del tauler.
```

9. **CD/CI**: Hem realitzat el CD i CI correctament, al github hem afegit els fitxers necessaris que se'ns demanava per comprovar els test i checklist al fer un merge a la branca main:



Main implementat: Aquest és el Main que hem fet per jugar al buscamines, amb la vista, tauler i Random reals:

```java
package es.uab.tqs.buscamines;
import java.util.Random;

import es.uab.tqs.buscamines.controlador.Joc;
import es.uab.tqs.buscamines.model.Tauler;
import es.uab.tqs.buscamines.vista.BuscaminesVista;
import es.uab.tqs.buscamines.vista.Vista;

public class Main {
    public static void main(String[] args) {

        //Classe Main per realitzar el joc real ja una vegada tot esta testejat correctament.
        Random r = new Random();
        Tauler t = new Tauler(r);
        // Crear joc amb vista incorporada
        Joc joc = new Joc(t);
        Vista vista  = new BuscaminesVista(joc); //Creem la vista real
        vista.initVista(); //La vista és genera.
        joc.crearVistaDelJoc(vista);
    }
}
```