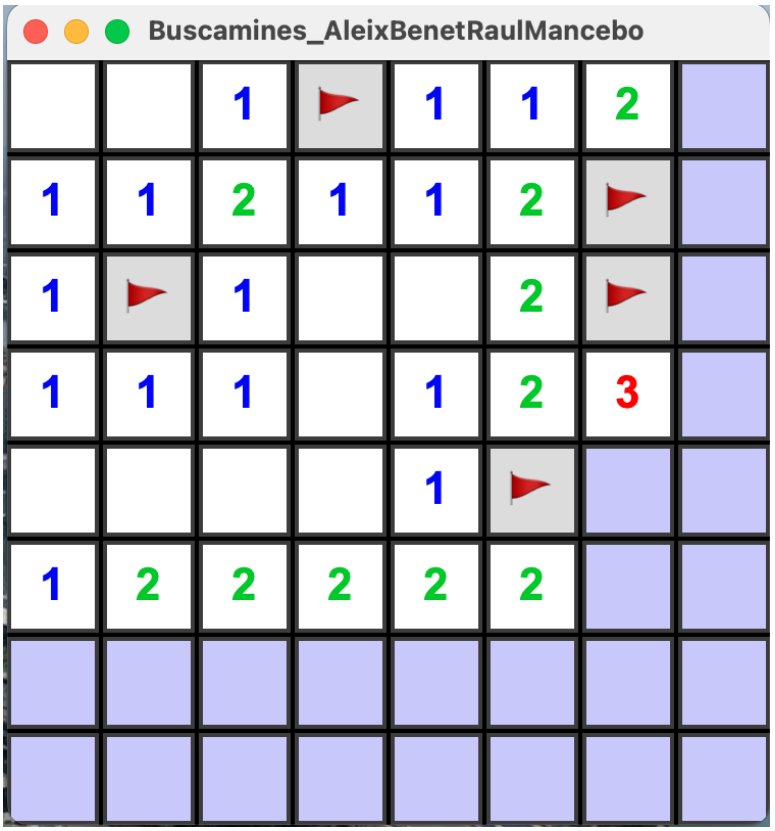
PRÀCTICA 1: APLICACIÓ DES DE ZERO

BUSCAMINES



Autors	Aleix Benet Bach - 1702609
	Raúl Mancebo González - 1706826
Assignatura	Test i Qualitat del Software
Professor	Cristóbal Pio
Data	09/12/2025

ÍNDEX

1. INTRODUCCIÓ

En aquest informe es els motius dels diferents testos realitzats en el marc de la pràctica de Test i Qualitat del Software i es mostra el cobriment de tals mètodes així com altres funcionalitats i metodologies aplicades durant el treball. El codi que resulta amb el joc de Buscamines té 3 grans blocs que estructuren el projcte en el paradigma model-vista-controlador. En el nostre programa tenim un controlador (classe Game) que és l'encarregat de gestionar la lògica i funcionalitats del joc mitjançant el principal model (classe Board) que utilitza una simple classe Cell i que, per una funció fa servir un generador Random (GenRandom). D'aquesta forma aconseguim el comportament del joc. Finalment, el controlador mostra a través de la vista de forma estilística les funcionalitats i el joc en sí. Aquesta vista està principalment composta d'una interfície (IView) i dues classe (BoardView i View) que la implementen.

Destacar que, tal com l'enunciat ens deia, els testos només s'han realitzat dels models i controlador. En el cas del model, només hem realitzar testeig de la classe Board ja que, per una banda, la Cell només està formada per *setters* i *getters* cosa que limita el sentit de testejar-la. I, per altra banda, el genRandom utilitza una llibreria (java.util.Random) que és de garanties i la funció no aporta cap més utilitat més que la de generar números randoms dins un determinat rang.

En total han sortit 149 testos entre caixa blanca (decision, condition coverage i loop testing), caixa negra (testos senzills inicials i particions equivalents + un pairwise) i una sèrie de testos definits a partir d'arxius de text CSV.

Aquesta ha estat la distribució dels testos:

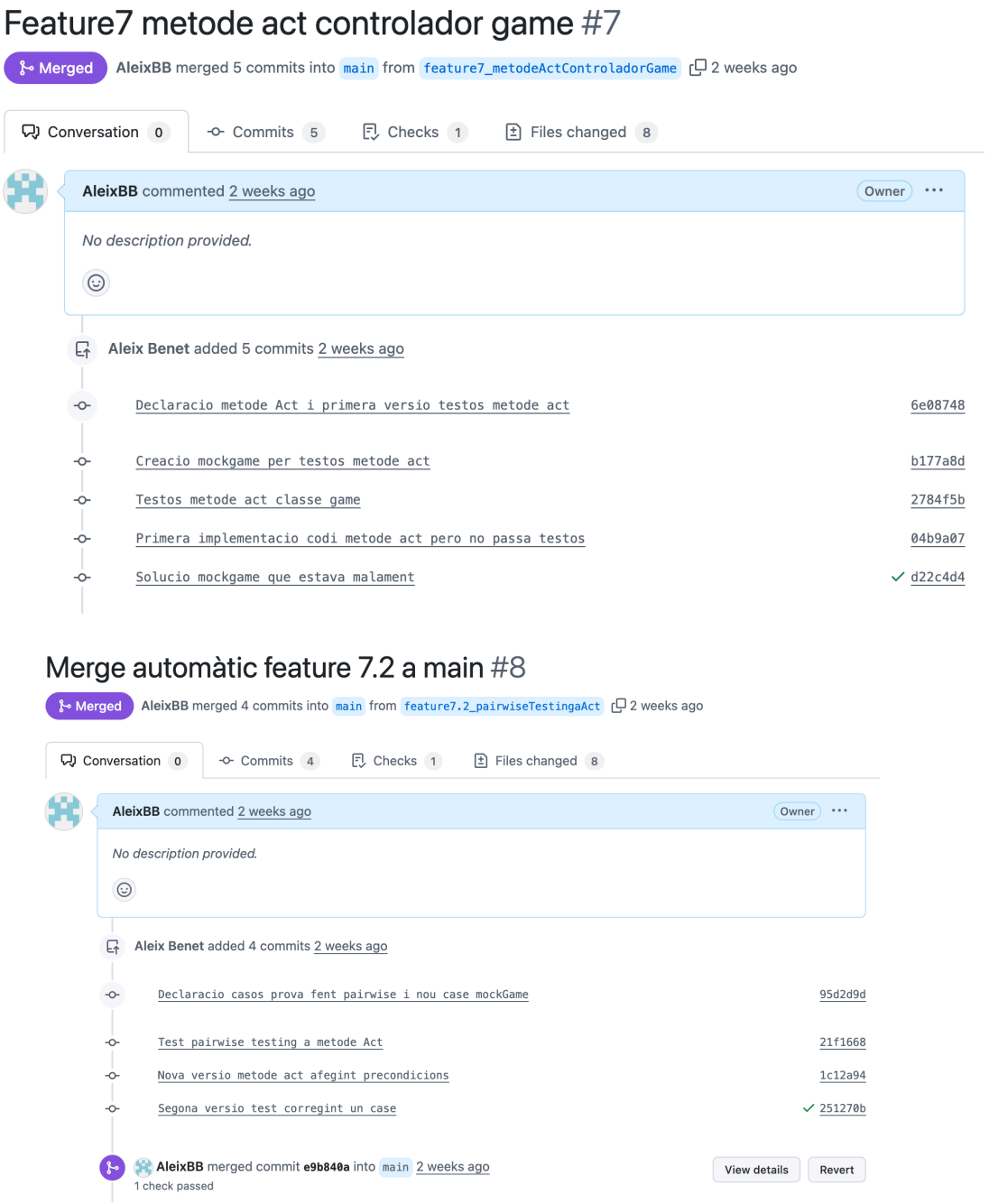
CAIXA NEGRA		CAIXA BLANCA			Data Driven	TOTAL
Senzills	Particions equivalents + Valors Frontera i Límit + Pairwise	Condition/Decision Coverage	Path Coverage	Loop Testing		
21	40	20	27	20	21	149

2. TDD

Al llarg del desenvolupament del projecte, hem seguit la metodologia TDD (Test Driven Development) és a dir, abans de fer cap implementació de funcions de codi, realitzar el test pertinent. Per mostrar aquest aspecte hem anat fent diferents *commits* a la plataforma *github* on es segueix els següents passos:

1. Declaració mètode (a codi implementat) i implementació test senzill
2. Implementació codi del mètode
3. Afegir testos més complexos
4. Si cal, corregir bugs detectats

La següent captura mostra una evidència i l'exemple d'aquest procediment en el mètode *act* de la classe *game* (controller/Game.java):



3. CAIXA NEGRA

Pel que respecta als testos de caixa negra, destacar les particions equivalents i els corresponents valors frontera i límit que en tots els mètodes s'apliquen per tal de comprobar entre les diferents jugades, el correcte funcionament dels mètodes. També hem aconseguit practicar la tècnica *pairwise testing* per disminuir els casos totals que haguéssim hagut de fer. L'hem aplicat al mètode *act* de la classe *Game*.

Classe Board

Mètode	**Test amb particions equivalents i valors límits frontera**	**Particions Equivalents**	
		Variable 1	Variable 2
Board ()	testInitNegatius()	**nMines** nMines < 0 → NO VÀLID nMines => 0 → VÀLID	**size** size <= 0 → NO VÀLID size > 0 → VÀLID
putMinesIntoBoard()	testPutMinesParticionsEquivalentsCentre() testPutMinesParticionsEquivalentsCantonada() testPutMinesParticionsEquivalentsExtremSupInfLat()	**fila, columna** 0<= fila, columna < size → VÀLID	**nMines** Si fila, columna és cantonada: nMines < size*size - 3 → VÀLID Si fila, columna és centre: nMines < size*size -8 → VÀLID Si fila, columna és extrem superior, inferior o lateral: nMines < size*size -5 → VÀLID
insertValueIntoCells()	testInsertValuesParticionsequivalent_capMines() testInsertValuesParticionsequivalent_unaMines() testInsertValuesParticionsequivalent_duesMines() testInsertValuesParticionsequivalent_tresMines()	**fila, columna** 0<=fila, columna < size → VÀLID	**matrix[fila][columna]** Si matrix[fila][columna].value() es 0 matrix[fila][columna].value() es -1 matrix[fila][columna].value() és > 0
firstClick()	testFirstClick_particionsEquivalent_clickAMina() testFirstClick_particioCero() testFirstClick_particioValorUn() testFirstClick_particioValorDos() testFirstClick_particioFilaiColumnesInvalides()	**fila, columna** 0<=fila, columna < size → VÀLID	**matrix[fila][columna]** Si matrix[fila][columna].value() es 0 matrix[fila][columna].value() es -1 matrix[fila][columna].value() és > 0
expandZeros()	testExpandZeros()	**fila, columna**	**matrix[fila][columna]**

	testExpandZeros_ParticionsEquivalentsFilaiColumnaInvalides() testExpandZeros_ParticionsEquivalentsCellaAmbZero() testExpandZeros_CellaAmbNum() testExpandZeros_CellaAmbMina() testExpandZeros_CellaYaRevelada()	0<=fila, columna < size →return	Si matrix[fila][columna].value() es 0 matrix[fila][columna].value() es -1 matrix[fila][columna].value() és > 0 matrix[fila][columna] està revelada matrix[fila][columna] no està revelada
clickAMina()	testParticionsEquivalents_ClickAMinaTaulerSenseMines() testParticionsEquivalents_ClickAMinaTaulerUnaMina() testParticionsEquivalents_ClickAMinaTaulerDosMines()	**size** 0 < size	**matrix[fila][columna]** Si matrix[fila][columna] és mina Si matrix[fila][columna] no és mina
isWin()	testIsWin_ParticionsEquivalents_CapReveal() testIsWin_ParticionsEquivalents_UnaReveal() testIsWin_ParticionsEquivalents_TotesRevealMenysUna() testIsWin_ParticionsEquivalents_TotesReveal()		**matrix[fila][columna]** Si alguna matrix[fila][columna] no reveal Si totes les matrix[fila][columna] estan reveal

En la següent captura es pot observar les 3 particions analitzades en el putMinesIntoBoard i que condicionen les altres funcions. Depenent d'on es fa el primer click, el nombre de mines possibles també varia. De forma que, si es fa a un extrem, hi haurà les tres cel·les adjaçents que hauran de tenir un 0 de forma forçosa: (la mina és on hi ha -1)

0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
-1	0	0	-1	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	0
0	0	0	0	0	0	-1

	CANTONADA	3
	CENTRE	8
	EXTREM LAT. O SUP.	5

Classe Game

Mètode	**Test amb particions equivalents i valors límits frontera**	**Particions Equivalents**		
		Variable 1	Variable 2	Particions
act()	arxiu GameTest.java	**action** “FLAG” o “REVEAL”	**x,y** 0<= x,y < size	REVEAL a cell {REVELADA, FLAGUEJADA} AMB {GAME OVER, WIN, ESTAT JOC} FLAG a cell {REVELADA, FLAGUEJADA} AMB {GAME OVER, WIN, ESTAT JOC}
startedGame()	arxius GameTest.java	**finish win** win == true win == false	**finish game over** game over == true game over == false	**firstClick** firstClick == true firstClick == false

4. CAIXA BLANCA

4.1 STATEMENT COVERAGE

Per veure el *coverage* del nostre codi el que hem fet ha estat utilitzar l'extensió JaCoCo que, juntament amb *maven*, ens ha permès extreure varis informes del cobriment de cada mètode i classe amb el percentatge i les línies cobertes amb els testos.
En general veiem com les dues parts que ens importen que són la classe Board (de tqs.prac.model) i la classe Game (de tqs.prac.controller) tenen un percentatge de cobriment molt pròxim al 100 tant en el cas d'instruccions executades com els casos executats.
Alguna condició mínima només s'avalua parcialment i és el que impedeix arribar al 100% en el cas de Game. En el cas de Board sí que totes les sentències s'han testejat. Per obtenir aquests informes hem realitzat aquestes comandes:

```
mvn test -Dcheckstyle.skip=true
```

```
mvn test jacoco:report -Dcheckstyle.skip=true
```

i seguidament obrir l'arxiu *html* generat.

Projecte en general

Buscaminas MVC

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
tqs.prac.view		0 %		0 %	34	34	105	105	15	15	4	4
tqs.prac		0 %		0 %	5	5	23	23	2	2	1	1
tqs.prac.model		97 %		100 %	1	65	5	108	1	19	0	3
tqs.prac.controller		100 %		94 %	2	29	0	55	0	11	0	1
Total	536 of 1.257	57 %	37 of 163	77 %	42	133	133	291	18	47	5	9

models

tqs.prac.model

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Cell		76 %		100 %	1	9	5	19	1	8	0	1
Board		100 %		100 %	0	54	0	85	0	9	0	1
GenRandom		100 %		n/a	0	2	0	4	0	2	0	1
Total	12 of 510	97 %	0 of 92	100 %	1	65	5	108	1	19	0	3

Board

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
putMinesintoBoard(int, int)		100 %		100 %	0	6	0	17	0	1
insertValueintoCells()		100 %		100 %	0	13	0	16	0	1
expandZeros(int, int)		100 %		100 %	0	12	0	14	0	1
clickACell(int, int)		100 %		100 %	0	7	0	12	0	1
Board(int, int, GenRandom)		100 %		100 %	0	5	0	13	0	1
isWin()		100 %		100 %	0	5	0	6	0	1
clickAMina()		100 %		100 %	0	4	0	5	0	1
getCell(int, int)		100 %		n/a	0	1	0	1	0	1
getSize()		100 %		n/a	0	1	0	1	0	1
Total	0 of 447	100 %	0 of 90	100 %	0	54	0	85	0	9

controlador

tqs.prac.controller

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Game		100 %		94 %	2	29	0	55	0	11	0	1
Total	0 of 223	100 %	2 of 36	94 %	2	29	0	55	0	11	0	1

Game

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
act(String, int, int)		100 %		96 %	1	16	0	24	0	1
startedGame()		100 %		83 %	1	4	0	12	0	1
Game()		100 %		n/a	0	1	0	7	0	1
gameOver()		100 %		n/a	0	1	0	2	0	1
win()		100 %		n/a	0	1	0	2	0	1
setFirstClick(Boolean)		100 %		n/a	0	1	0	2	0	1
setBoard(Board)		100 %		n/a	0	1	0	2	0	1
setView(IView)		100 %		n/a	0	1	0	2	0	1
getGameOver()		100 %		n/a	0	1	0	1	0	1
getWin()		100 %		n/a	0	1	0	1	0	1
getBoard()		100 %		n/a	0	1	0	1	0	1
Total	0 of 221	100 %	2 of 35	94 %	2	29	0	56	0	11

4.2

DECISION/CONDITION COVERAGE

4.2.1 Mètode Game/StartedGame

Per analitzar el comportament de la funció StartedGame on tots els condicionals prenien els valors True i False hem definit els següents casos de test:

Número de cas	firstClick	win	gameOver
1	True	False	False
2	True	False	True
3	True	True	False
4	False	False	False
5	False	False	True
6	False	True	False

El cas 2 sembla que no té sentit però el que hem fet ha sigut mitjançant el mock object de la vista definir dos accions, la primera que revela i òbviament no pot fer perdre al jugador, i la segona que també revela i sí que acaba fent perdre a l'usuari.
Els altres casos juguen amb diferents taulers senzills i comprovem que tots els condicionals prenen true i false.
Això ho podem observar amb la següent captura corresponent a haver executat la sentència *run with coverage* al test (les línies verdes signifiquen que es testeja i les vermelles amb línies discontínues que no):

```
mvn test -Dtest=GameDecisionCoverageTest -Dcheckstyle.skip=true
```

```
mvn test jacoco:report -Dtest=GameDecisionCoverageTest -Dcheckstyle.skip=true
```

4.2.2 Mètode Board/firstClick

```
94.     public Boolean firstClick(int fila, int columna){
95.         if (fila >= this.size || fila < 0)
96.         {
97.             throw new IllegalArgumentException("Fila fora rang");
98.         }
99.         if (columna >= this.size || columna < 0)
100.        {
101.            throw new IllegalArgumentException("Columna fora rang");
102.        }
103.
104.        Cell clicked = matrix[fila][columna];
105.        // Clic a una mina
106.        if (matrix[fila][columna].getValue() == -1)  {
107.            clickAMina();
108.            return false;
109.        } else {
110.            if(clicked.getValue() == 0) { // Clic a un zero --> Expansió
111.                expandZeros(fila, columna);
112.            } else  { // Clic a numero --> Revelació
113.                clicked.reveal();
114.            }
115.        }
116.        return true;
117.    }
```

Número de cas	cell
1	Mina
2	Valor 0
3	Valor > 0

4.2.3 Mètode Board/putMinesIntoBoard

```
mvn test -Dtest=BoardDecisionCoveragePutMinesTest -Dcheckstyle.skip=true
```

```
mvn test jacoco:report -Dtest=BoardDecisionCoveragePutMinesTest -Dcheckstyle.skip=true
```

```java
public void putMinesintoBoard(int fila, int columna)  { // Colocar les mines de manera aleatoria
    // Calculem la zona protegida 3x3
    int zonaProtegidaFiles = (Math.min(fila + 1, size-1) - Math.max(fila - 1, 0)) + 1;
    int zonaProtegidaCols = Math.min(columna + 1, size - 1) - Math.max(columna - 1, 0) + 1;
    int tamanyZonaProtegida = zonaProtegidaFiles * zonaProtegidaCols;
    int celdasDisponibles = size * size - tamanyZonaProtegida;

    // Precondició de Caixa Negra
    if (nMines > celdasDisponibles) { // Més mines que espais
        throw new IllegalArgumentException("El nombre de mines excedeix la mida del tauler");
    }

    int count = 0;
    while (count < nMines) {
        int r = random.nextInt(size);
        int c = random.nextInt(size);
        //Evitar la zona protegida
        if (Math.abs(r - fila) <= 1 && Math.abs(c - columna) <= 1) {
            continue;
        }
        // Col·loca la mina si no té encara una
        if (matrix[r][c].getValue() == 0) {
            matrix[r][c].setValue(-1);
            count++;
        }
    }
}
```

Board

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
insertValueintoCells()		0 %		0 %	13	13	16	16	1	1
expandZeros(int, int)		0 %		0 %	12	12	14	14	1	1
firstClick(int, int)		0 %		0 %	7	7	12	12	1	1
isWin()		0 %		0 %	5	5	6	6	1	1
clickAMina()		0 %		0 %	4	4	5	5	1	1
Board(int, int, GenRandom)		81 %		75 %	2	5	2	13	0	1
getSize()		0 %		n/a	1	1	1	1	1	1
putMinesintoBoard(int, int)		100 %		90 %	1	6	0	17	0	1
getCell(int, int)		100 %		n/a	0	1	0	1	0	1
Total	290 of 447	35 %	75 of 90	16 %	45	54	56	85	6	9

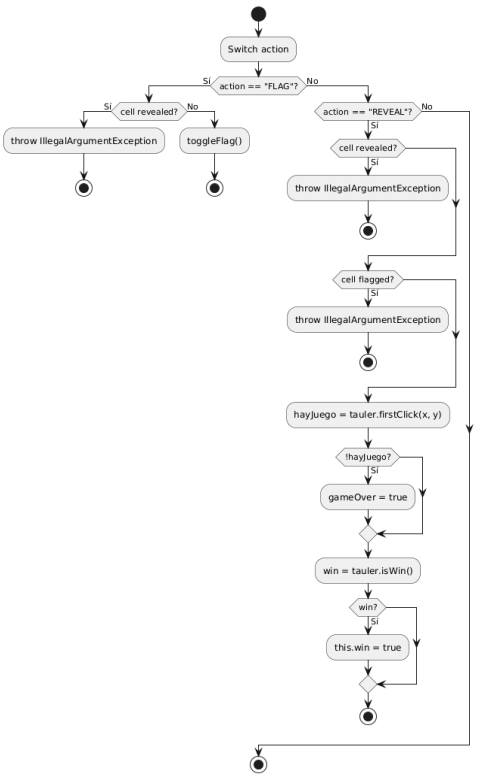
Número de cas	nMines > Cel.lesDisponibles	mines Zona Protegida	cell amb mina
1	True	False	False
2	False	False	False
3	False	True	False
4	False	False	False
5	False	False	True
6	False	False	False

4.4 PATH COVERAGE

4.4.1 Mètode Game/Act

Hem aprofitat que el mètode *act* de la classe Game tenia forçes condicionals anidats per analitzar el *path coverage*, és a dir, passar per totes les sentències que componen aquest mètode. Els diferents *test cases* són els diferents camins:

Número de cas	Path
1	FLAG → CELL NOT REVEALED
2	FLAG → CELL REVEALED
3	REVEAL → CELL REVEALED
4	REVEAL → CELL NOT REVEALED → CELL FLAGGED
5	REVEAL → CELL NOT REVEALED → CELL NOT FLAGGED
6	REVEAL → CELL NOT REVEALED → CELL NOT FLAGGED → NOT GAME OVER → WIN
7	REVEAL → CELL NOT REVEALED → CELL NOT FLAGGED → GAME OVER



```
mvn test -Dtest=GamePathCoverageActTest -Dcheckstyle.skip=true
```

```
mvn test jacoco:report -Dtest=GamePathCoverageActTest -Dcheckstyle.skip=true
```

Game

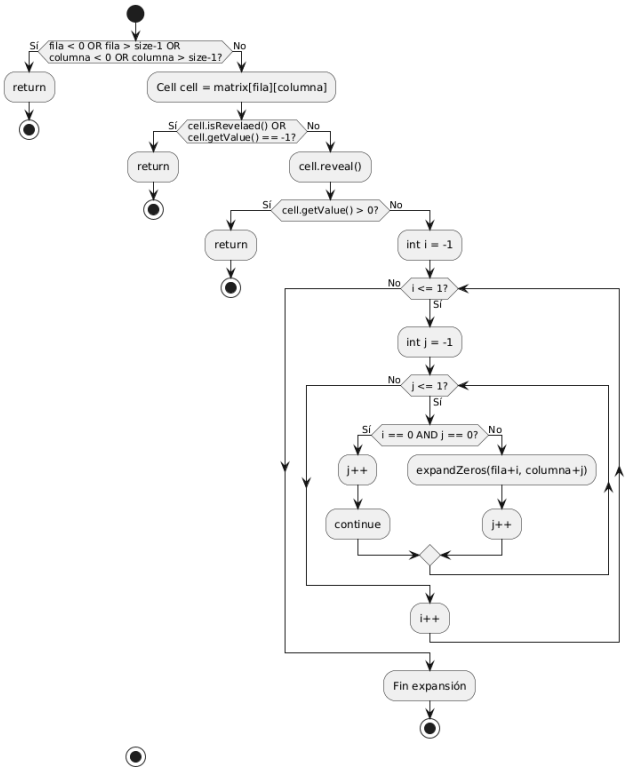
Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
startedGame()		0 %		0 %	4	4	12	12	1	1
setFirstClick(Boolean)		0 %		n/a	1	1	2	2	1	1
setView(IView)		0 %		n/a	1	1	2	2	1	1
act(String, int, int)		100 %		100 %	0	15	0	23	0	1
Game()		100 %		n/a	0	1	0	7	0	1
gameOver()		100 %		n/a	0	1	0	2	0	1
win()		100 %		n/a	0	1	0	2	0	1
setBoard(Board)		100 %		n/a	0	1	0	2	0	1
getGameOver()		100 %		n/a	0	1	0	1	0	1
getWin()		100 %		n/a	0	1	0	1	0	1
getBoard()		100 %		n/a	0	1	0	1	0	1
Total	54 of 216	75 %	6 of 34	82 %	6	28	16	55	3	11

```
54.     }
55.
56.     public void act(String action, int x, int y)  { // Rep acció del jugador (FLAG O REVEAL) i les coordenades
57.         //precondicions
58.         if (this.getGameOver() == true ||  this.getWin() == true)
59.         {
60.             throw new IllegalArgumentException("La partida ha finalitzat");
61.         }
62.
63.         if (x < 0 || x >= this.getBoard().getSize())
64.         {
65.             throw new IllegalArgumentException("Fila fora de rang");
66.         }
67.         if (y < 0 || y >= this.getBoard().getSize())
68.         {
69.             throw new IllegalArgumentException("Columna fora de rang");
70.         }
71.         if (action != "FLAG" && action != "REVEAL")
72.         {
73.             throw new IllegalArgumentException("Accio Invalida");
74.         }
75.         if (action == "FLAG")
76.         {
77.             if (tauler.getCell(x, y).isRevelaed()){
78.                     throw new IllegalArgumentException();
79.                 }
80.                 tauler.getCell(x, y).toggleFlag();
81.         }
82.         else
83.         {
84.             if ((tauler.getCell(x, y).isRevelaed())) {
85.                     throw new IllegalArgumentException();
86.                 }
87.                 if ((tauler.getCell(x, y).isFlagged())) {
88.                     throw new IllegalArgumentException();
89.                 }
90.                 Boolean hayJuego = tauler.firstClick(x, y);
91.                 if (!hayJuego)  { //game over o win???
92.                     this.gameOver = true;
93.                 }
94.                 Boolean win = tauler.isWin(); // Comprobar si hem guanyat o no
95.                 if (win) {
96.                     this.win = true;
97.                 }
98.         }
99.     }
100.
```

Com podem observar a la captura anterior, a la zona del número de línia si surt verd és que es testeja Es pot veure que s'analitzen tots els camins

4.4.2 Mètode Board/ExpandZeros

Número de cas	Path
1	COORDENADES FORA LÍMITS
2	COORDENADES DINS LIMITS → CELL REVEALED
3	COORDENADES DINS LÍMITS → CELL ES MINA (= -1)
4	COORDENADES DINS LÍMITS → CELL NO REVELADA I NO MINA → CELL VALOR MÉS GRAN QUE 0
5	COORDENADES DINS LÍMITS → CELL NO REVELADA I NO MINA → CELL VALOR ÉS 0




```
mvn test -Dtest=BoardPathCoverageExpandZerosTest -Dcheckstyle.skip=true
```

```
mvn test jacoco:report -Dtest=BoardPathCoverageExpandZerosTest -Dcheckstyle.skip=true
```

Board

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
firstClick(int, int)		0 %		0 %	7	7	12	12	1	1
isWin()		0 %		0 %	5	5	6	6	1	1
clickAMina()		0 %		0 %	4	4	5	5	1	1
Board(int, int, GenRandom)		81 %		75 %	2	5	2	13	0	1
putMinesintoBoard(int, int)		94 %		70 %	3	6	2	17	0	1
getSize()		0 %		n/a	1	1	1	1	1	1
insertValueintoCells()		100 %		100 %	0	13	0	16	0	1
expandZeros(int, int)		100 %		100 %	0	12	0	14	0	1
getCell(int, int)		100 %		n/a	0	1	0	1	0	1
Total	144 of 447	67 %	31 of 90	65 %	22	54	28	85	4	9

```
    public void expandZeros(int fila, int columna) {
        // Fora del tauler
        if (fila < 0||fila > (size-1) || columna < 0 ||columna > (size-1))  {
            return;
        }

        Cell cell = matrix[fila][columna];

        // Ja revelada o és una mina
        if (cell.isRevelaed() || cell.getValue() == -1) {
            return;
        }

        cell.reveal();

        if (cell.getValue() > 0) { // La cel·la es un número
            return;
        }

        for (int i = -1; i<= 1; i++) { // Recusiu Cell==0
            for (int j=-1; j<=1; j++) {
                if (i==0 && j==0) {
                    continue;
                }
                expandZeros(fila+i, columna+j);
            }
        }
    }
```

4.5 LOOP TESTING

Tipus	Funció testejada	Casos
Bucle simple 1	putMinesIntoBoard	No executar el bucle (nMines = 0) Una iteració (nMines = 1) Dos iteracions (nMines = 2) M iteracions (M < nMines) n-1 iteracions (nMines - 1) n iteraciones (nMines) n Iteracions + 1

```
    public void putMinesintoBoard(int fila, int columna)  {
        // Calculem la zona protegida (depen de on tinguem la mina)
        int zonaProtegidaFiles = (Math.min(fila + 1, size-1) - Math.max(fila - 1, 0)) + 1;
        int zonaProtegidaCols = Math.min(columna + 1, size - 1) - Math.max(columna - 1, 0) + 1;
        int tamanyZonaProtegida = zonaProtegidaFiles * zonaProtegidaCols;
        int celdasDisponibles = size * size - tamanyZonaProtegida;

        // Precondició
        if (nMines > celdasDisponibles) { // Més mines que espais disponibles
            throw new IllegalArgumentException("El nombre de mines excedeix la mida del tauler");
        }

        int count = 0;
        while (count < nMines) {
            int r = random.nextInt(size);
            int c = random.nextInt(size);
            //Evitar la zona protegida
            if (Math.abs(r - fila) <= 1 && Math.abs(c - columna) <= 1) {
                continue;
            }
            // Col·loca la mina si no té encara una
            if (matrix[r][c].getValue() == 0) {
                matrix[r][c].setValue(-1);
                count++;
            }
        }
    }
    /*
    Assigna a cada cel·la sense mina el valor que indica el nombre de mines
    adjacents a la seva àrea veïna (8 direccions).
    */
```

Bucles simple 2 (avaluem el bucle extern)	insertValuesIntoCells	No executar el bucle (nMines = 0) Una iteració (nMines = 1) Dos iteracions (nMines = 2) M iteracions (M < nMines) n-1 iteracions (nMines - 1) n iteracions (nMines) n Iteracions + 1

```
    public void insertValueintoCells() {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (matrix[i][j].getValue() == -1) { // si es mina
                    continue;
                }
                int count = 0;

                for (int di = -1; di <= 1; di++) { // Comptar veïns
                    for (int dj = -1; dj <= 1; dj++) {
                        if (di == 0 && dj == 0) {
                            continue;
                        }
                        int ni = i + di;
                        int nj = j + dj;

                        // Comprovar límits del tauler
                        if (ni >= 0 && ni < size && nj >= 0 && nj < size) {
                            if (matrix[ni][nj].getValue() == -1)
                                count++;
                        }
                    }
                }
                matrix[i][j].setValue(count);
            }
        }
    }
```

Bucles anidats 1	clickAMina	No executar el bucle (nMines = 0) Una iteració (nMines = 1) M iteracions (M < nMines) n-1 iteracions (nMines - 1) n iteracions (nMines) n Iteracions + 1

```
    public void clickAMina() {
        for (int i = 0; i < size; i++) {
            for (int j = 0; j < size; j++) {
                if (matrix[i][j].getValue() == -1) {
                    matrix[i][j].reveal(); // Revelem la mina
                }
            }
        }
    }
    /*
```

5. MOCK OBJECTS

Per realitzar els diferents tests hem hagut de simular vàries classes i funcionalitats com ara la interfície d'usuari, el generador de mines random i el tauler.

5.1 MockGenRandom

Aquest Mock Object ens ha permès obtenir un resultat determinista quan cridàvem a la funció models/Board.java/putMinesIntoBoard que realment posàva nMines a coordenades Random del tauler per aconseguir que el tauler del joc fos cada partida diferent. Per testejar els mètodes de la classe Board ens interessava en alguns testos tenir prefixada la posició de les mines i d'aquesta forma també analitzar el comportament en casos extrems.

5.2 MockBoard

Per poder testejar el controlador Game ens hem vist obligats a crear un fals tauler determinista i que ens ha permès tenir un control a l'hora de realitzar testos dels mètodes de Game.

5.3 MockView

Per poder realitzar alguns testos hem simulat la comunicació amb el jugador utilitzant aquest MockView. Senzillament el refresh() no fa res i l'únic que retorna és una llista de les accions que realitzaria un usuari.

6. CD/CI

Ho hem implementat a l'arxiu *maven.yml* a la carpeta *workflows* de *github*. Només ens permet fer un merge a main de les *features* si absolutament tots els testos del programa passaven correctament i no hi havia cap problema de tipus sintàctic, compilació, entre altres.. Ens ha permès tenir un control en tot moment i que no se'ns desviés algun test per alguna modificació d'un altra.
Per altra banda també hem afegit a aquest mateix fitxer una comprovació del *checkstyle* limitant-nos a que:

- la mida màxima d'una línia de codi és 100 caràcters.
- la mida màxima d'un mètode és de 50 línies.
- la complexitat ciclomàtica màxima per mètode és de 15.
- no permetre cap import que no s'utilitzi.

Llavors des de l'apartat *actions* de *github* només podem realitzar el merge a main si es passen tots els testos i es compleix aquests aspectes respecte la qualitat del codi.