Pablo Gil Gómez-1669275
Albert Vacas Martínez-1665473
Horari de pràctiques: Dilluns 10:30-12:30
Nom Projecte: Buscamines-tqs

Començar una partida:
Funcionalitat: Comença la partida amb un missatge, després crida al metode configurarPartida(), col·loca les bombes que hagi volgut el jugador i va fent crides per anar jugant fins que el jugador guanyi, (hagi descobert totes les caselles sense bombes) o perdi (ha trobat una bomba).

Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.controlador
- Classe: Partida
- Mètode: configurarPartida()

Test:

```java
class PartidaTest {

    @Mock
    private entryData mockInput;

    @Mock
    private VistaTauler mockVista;

    @Mock
    private MessageTxt mockMessage;

    @Mock
    private Tauler mockTauler;

    @InjectMocks
    private Partida partida; // Usa los mocks anteriores para inicializar Partida


    @BeforeEach
    public void setUp(){
        // Inicializar los mocks
        MockitoAnnotations.openMocks(this);

        // Crear la instancia de Partida usando los mocks
        partida = new Partida(mockTauler, mockMessage, mockInput, mockVista);
```

Hem realitzat els tests de la classe Partida amb Mockito ja que aquesta classe té dependencies amb les classes de la Vista i el Controlador. Hem fet mocks de les classes VistaTauler, MessageTxt, entryData i Tauler. Per poder fer testing de model,vista i controlador.

Per el mètode startPartida(), hem simulat els dos casos on el jugador guanya i el cas en que el jugador perd. Assegurant així statement coverage.

```java
@Test
void testStartPartidaDefeat() {

    // Configurar los valores simulados para input.readInput()
    when(mockInput.readInput()).thenReturn(new int[]{0, 0});

    // Configurar mockVista.validarPosicio() para siempre devolver true
    when(mockVista.validarPosicio(any(Tauler.class), anyInt(), anyInt())).thenReturn(true);

    // Configurar el comportamiento de tauler
    when(mockTauler.isWon()).thenReturn(false);
    when(mockTauler.isBomb(anyInt(), anyInt())).thenReturn(true); //Pisa una bomba en la pri

    // Ejecutar el método a probar
    partida.startPartida();

    // Verificar las interacciones
    verify(mockMessage, times(1)).presentacio(); // Verifica que se muestre el mensaje de pr
    verify(mockInput, atLeastOnce()).readInput(); // Verifica que se lea la entrada
    verify(mockTauler, times(1)).setRandomBombs(); // Verifica que se generen bombas
    verify(mockVista, atLeastOnce()).printTauler(mockTauler); // Verifica que se imprima el
    verify(mockVista, atLeastOnce()).printStatus(mockTauler);
    verify(mockTauler, atLeastOnce()).mostrarCasella(anyInt(), anyInt());
    verify(mockTauler, times(1)).mostrarTaulell();
    verify(mockMessage, times(1)).bomb(); // Verifica que se muestre el mensaje de victoria
}
```

```java
public void startPartida() {
    assert mensajes != null : "mensajes no puede ser null";
    assert entrada != null : "entrada no puede ser null";
    assert vistaTablero != null : "vistaTablero no puede ser null";
    mensajes.presentacio();
    configurarJuego();
    assert tablero != null : "tablero debe estar configurado después de configurarJuego";
    tablero.colocarBombasAleatorias();
```

També apliquem disseny per contracte.

```java
public void startPartida() {

    message.presentacio();

    this.configurarPartida();

    tauler.setRandomBombs();


    do {

        vista.printTauler(tauler);

        vista.printStatus(tauler);

        validarInput();

        tauler.mostrarCasella(moviment[0], moviment[1]);

    }while(!tauler.isWon() && (!tauler.isBomb(moviment[0], moviment[1])));

    tauler.mostrarTaulell();

    vista.printTauler(tauler);

    if(tauler.isWon()) {
        message.victory();
    }else {
        message.bomb();
    }
}
```

```java
    public void configurarPartida() {

        if(tauler == null) {
            int[] parametres;
            parametres = input.configurarPartida(maxFiles, minFiles, maxCol, minCol, maxMines, mi
            tauler = new Tauler(parametres[0], parametres[1], parametres[2]);
        }


    }

    public void validarInput() {

        do{
            moviment = input.readInput();

        }while(!vista.validarPosicio(tauler, moviment[0], moviment[1]));

    }

}
```

Configurar una partida:

Funcionalitat: Permet configurar una partida, col·locant las files, columnes i bombes que vulgui el jugador per personalitzar al màxim la partida:

Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.controlador
- Classe: Partida
- Mètode: configurarPartida()

Test:

```
@Test
void testConfigurarPartidaNull() {
    when(mockInput.configurarPartida(anyInt(), anyInt(), anyInt(), anyInt(), anyInt(),
            anyInt())).thenReturn(new int[] {0,0,0});
    partida.configurarPartida();

    verify(mockInput, times(0)).configurarPartida(anyInt(), anyInt(), anyInt(),
            anyInt(), anyInt(), anyInt());
}

@Test
void testConfigurarPartidaNotNull() {

    when(mockInput.configurarPartida(anyInt(), anyInt(), anyInt(), anyInt(),
            anyInt(), anyInt())).thenReturn(new int[]{0, 0, 0});

    partida = new Partida(mockMessage, mockInput, mockVista);
    partida.configurarPartida();

    verify(mockInput, atLeastOnce()).configurarPartida(anyInt(), anyInt(), anyInt(),
            anyInt(), anyInt(), anyInt());
}
```

Realitzem els tests pels dos casos possibles d'aquesta funció ja que conte un if()

- Quan el tauler es null, per tant es crida a la funció configuraTauler de la classe entryData dins del if.
- Quan el tauler ja està configurat per tant no entra dins l'estructura condicional del if.

També assegurem Statement coverage.

Validar un Input:

Comprova que la coordenada que es posa és correcte i entra dintre dels paràmetres permesos del joc. Per exemple, no permet números negatius.

Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.controlador
- Classe: Partida
- Mètode: validarInput()

Test:

```java
    @Test
    void testValidarInputInvalid() {
        when(mockInput.readInput()).thenReturn(new int[] {0,0});
        when(mockVista.validarPosicio(any(Tauler.class), anyInt(), anyInt()))
            .thenReturn(false).thenReturn(true);//introduce un valor valido a la segunda

        partida.validarInput();

        verify(mockInput, times(2)).readInput();
        verify(mockVista, times(2)).validarPosicio(any(Tauler.class), anyInt(), anyInt());
    }

    @Test
    void testValidarInputValid() {
        when(mockInput.readInput()).thenReturn(new int[] {0,0});
        when(mockVista.validarPosicio(any(Tauler.class), anyInt(), anyInt())).thenReturn(true);/

        partida.validarInput();

        verify(mockInput, times(1)).readInput();
        verify(mockVista, times(1)).validarPosicio(any(Tauler.class), anyInt(), anyInt());
    }

}
```

Assegurem amb aquest també statement coverage.

En aquest cas, realitzem dos test per els dos casos en el que l'usuari introdueix un valor:

Quan és invalid i per tant el bucle do-while es torna a executar fins que s'introdueix un valor correcte i en el cas en que l'usuari introdueix un valor vàlid i no es realitza cap iteració en el bucle.

Comentaris addicionals de la classe partida:

Fem un 100% de coverage de la classe partida i fem ús de diferents mocks creats per mockito per assegurar la funcionalitat de la classe.

Inicialitzar una casella:

Descripció: S'ha implementat la funcionalitat de reiniciar una casella utilitzant el mètode reset(). Aquest mètode assegura que els atributs de la casella es reestabliran el seu estat inicial: esVisible es marca com a false, el tipus de casella es defineix com a Normal, i els veïns es posen a 0.

Localització: es troba a la Classe Casella en el mètode reset().

Test: S'ha realitzat el test a la classe CasellaTest on s'ha fet statement coverage i proves de caixa negra amb els valors límit i frontera, en el mètode testReset().

Statement coverage:

```java
public void reset() {
    esVisible = false;
    tipus = new Normal();
    veins = 0;
    //assert !esVisible : "Postcondición fallida: esVisible debe ser false después de reset().";
    //assert tipus instanceof Normal : "Postcondición fallida: tipus debe ser Normal después de reset().";
    //assert veins == 0 : "Postcondición fallida: veins debe ser 0 después de reset().";
}
```

També s'ha aplicat Disseny per contracte.

```java
assert !esVisible : "Postcondición fallida: esVisible debe ser false después de reset().";
assert tipus instanceof Normal : "Postcondición fallida: tipus debe ser Normal después de reset().";
assert veins == 0 : "Postcondición fallida: veins debe ser 0 después de reset().";
```

Incrementar veïns:
Descripció: S'ha afegit el mètode addVei() que permet incrementar el nombre de veïns d'una casella en una unitat.
Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.model
- Classe: Casella
- Mètode: addVei()

Test: s'ha realitzat el statement coverage i proves de caixa negra pertinents amb els valors límit i frontera en el mètode testAddVei() de la classe CasellaTest.

```java
@Test
void testAddVei() {
    Casella c1 = new Casella();
    c1.addVei();
    assertEquals(c1.getVeins(), 1);


    Casella c2 = new Casella();
    c2.addVei();
    c2.addVei();
    assertEquals(c2.getVeins(), 2);


    Casella c3 = new Casella();
    for(int i = 0; i < 7; i++) {
        c3.addVei();
    }
    assertEquals(c3.getVeins(), 7);


    Casella c4 = new Casella();
    for(int i = 0; i < 8; i++) {
        c4.addVei();
    }
    assertEquals(c4.getVeins(), 8);


    Casella c5 = new Casella();
    for(int i = 0; i < 9; i++) {
        c5.addVei();
    }
    assertEquals(c5.getVeins(), 9);


    Casella c6 = new Casella();
    for(int i = 0; i < 50; i++) {
        c6.addVei();
    }
    assertEquals(c6.getVeins(), 50);
}
```

```java
public void addVei() {
    veins++;
}
```

Assignar una bomba a una casella i comprovar que es bomba

Descripció: El mètode setBomb() permet assignar una bomba a la casella canviant el seu tipus a Mina i amb isBomb(), mirem si es una bomba o no.

Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.model
- Classe: Casella
- Mètode: setBomb()

Test: Hem realtzat els tests en testSetBombs i testSetBomb amb statement coverage i proves de caixa negra pertinents en CasellaTest. També hem aplicat disseny per contracte per assegurar que la instancia es de tipus mina.

```java
public void setBomb() {
    tipus = new Mina();
    //assert tipus instanceof Mina;
}

public boolean isBomb() {
    return tipus.isBomb();
}
```

```java
public void setBomb() {
    tipus = new Mina();
    assert tipus instanceof Mina;
}
```

Mostrar una casella

Descripció: El mètode mostrarCasella() actualitza l'atribut esVisible de la casella a true.
Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.model
- Classe: Casella
- Mètode: mostrarCasella()

Test: Els tests es troben a CasellaTest a testMostrarCasella(), on s'ha fet statement coverage i proves de caixa negra. A més hem col·locat disseny per contracte a mostrarCasella(), per assegurar que es visible.

```java
public boolean getEsVisible() {
    return esVisible;
}

public void mostrarCasella() {
    esVisible = true;
    //assert esVisible;
}
```

```java
public void mostrarCasella() {
    esVisible = true;
    assert esVisible;
}
```

Representació textual d'una casella

Descripció: El mètode toString() genera una representació textual de la casella en funció del seu estat i atributs: una "B" si conté una bomba visible, un número si és visible i indica el nombre de veïns, o un "*" si és oculta.

Localització:

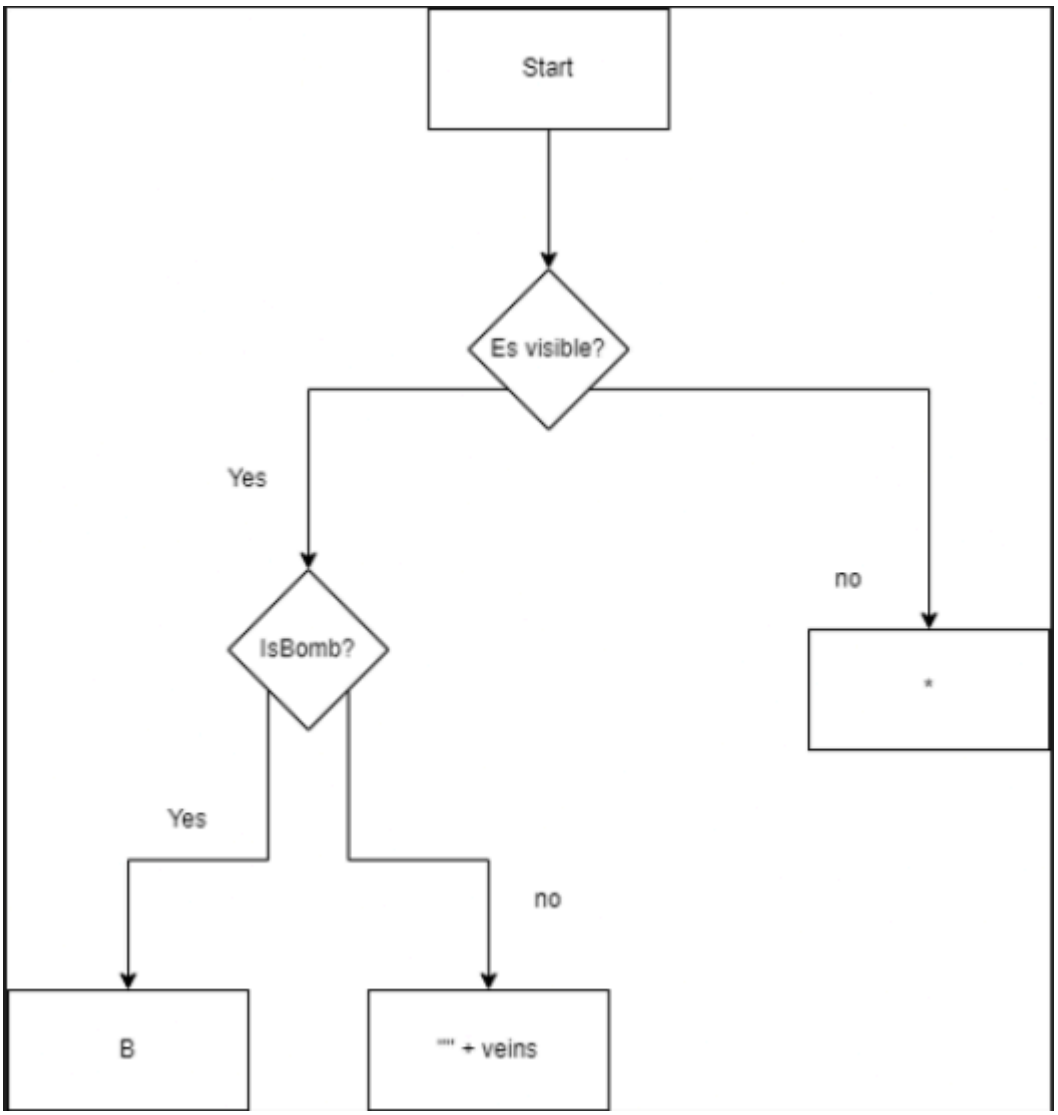
- Arxiu: main.java.cat.uab.tq.buscamines.model
- Classe: Casella

- Mètode: toString()

Test: Hem realitzat els tests a CasellaTest en el mètode testToString on s'ha fet statement coverage , les proves de caixa negra pertinents com valors límits i frontera. Amb decision coverage. També realitzem un Path coverage a la funcio toString().

```
public String toString() {
    if(esVisible) {
        if(isBomb()) {
            return "B";
        }
        else {
            return ""+veins;
        }
    }
    else
        return "*";
}
```

```
@Test
void testToString() {
    Casella c1 = new Casella();
    c1.setBomb();
    c1.mostrarCasella();
    assertEquals("B", c1.toString());

    Casella c2 = new Casella();
    assertEquals("*", c2.toString());

    Casella c3 = new Casella();
    c3.mostrarCasella();
    assertEquals("0", c3.toString());

    Casella c4 = new Casella();
    c4.addVei();
    c4.mostrarCasella();
    assertEquals("1", c4.toString());

    Casella c5 = new Casella();
    for(int i = 0; i < 8;i++) {
        c5.addVei();
    }
    c5.mostrarCasella();
    assertEquals("8", c5.toString());

}
```



Path Coverage.

```java
@Test
void testToString_esVisibleFalse() {
    Casella casella = new Casella();
    casella.reset();
    assertEquals("*", casella.toString(), "Cuando esVisible es false, debe devolver '*'.");
}

@Test
void testToString_esVisibleTrue_isBombTrue() {
    Casella casella = new Casella();
    casella.reset();
    casella.setBomb();
    casella.mostrarCasella();
    assertEquals("B", casella.toString(), "Cuando esVisible es true y es una bomba, debe devolver 'B'.");
}

@Test
void testToString_esVisibleTrue_isBombFalse() {
    Casella casella = new Casella();
    casella.reset();
    casella.addVei(); // Simula que tiene 1 vecino.
    casella.mostrarCasella();
    assertEquals("1", casella.toString(), "Cuando esVisible es true y no es una bomba, debe devolver el número de vecinos.");
}
```

Test del path coverage.

Comentaris addicionals de la classe Casella:

1. Tots els mètodes implementats inclouen assertions per assegurar que es compleixen les postcondicions descrites.
2. S'ha garantit una cobertura del 100% de sentències en les proves associades a cada funcionalitat.
3. Hem fet un 100% de coverage en aquesta classe.

Assignar tipus de casella Mina

Descripció: S'ha implementat la funcionalitat per identificar una casella del tipus Mina. Aquesta classe hereta de tipusCasella i redefineix el mètode isBomb() per retornar true quan es consulta si aquesta casella conté una bomba.
Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.model
- Classe: Mina
- Mètode: isBomb()

Test: Hem realitzat tests en la classe MinaTest on comprovem que es bomba en el mètode testEsBomba. Hem fet Statement Coverage.

```java
public class MinaTest {
    @Test
    public void testEsBomba() {
        Mina mina = new Mina();
        assertTrue("Mina debe identificarse como bomba", mina.isBomb());
    }
}
```

```java
public class Mina extends tipusCasella {
    @Override
    public boolean isBomb() { return true; }
}
```

Comentaris addicionals de la classe mina:

És filla de tipusCasella i té el 100% de coverage

Assignar tipus de casella Normal

Descripció: S'ha implementat la classe Normal, que representa una casella sense bomba. Aquesta classe hereta de tipusCasella i redefineix el mètode isBomb() per retornar false.
Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.model
- Classe: Normal
- Mètode: isBomb()

Test: S'ha fet els tests a la classe NormalTest, amb statement coverage.

```java
public class NormalTest {
    @Test
    public void testNoEsBomba() {
        Normal normal = new Normal();
        assertFalse("Normal no debe identificarse como bomba", normal.isBomb());
    }
}
```

```java
public class Normal extends tipusCasella {
    @Override
    public boolean isBomb() { return false; }
}
```

Comentaris addicionals de la classe Normal:

s'ha fet un coverage del 100%:

Definir tipus abstracte de casella

Descripció: S'ha creat la classe abstracta tipusCasella com a base per als diferents tipus de caselles (Mina i Normal). Aquesta classe defineix el mètode abstracte isBomb(), que cada subclasse implementa segons el seu comportament específic.
Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.model
- Classe: tipusCasella
- Mètode: isBomb() (abstracte)

Test: s'ha fet el test a TipusCasellaTest on s'ha fet statement coverage i proves de caixa negra dintre de la poca possiblitat que deixa aquesta classe perque la seva implementació és molt bàsica com la de les seves filles. Els tests estan en el mètode testHerencia() i testPolimorfismo().

```java
public class tipusCasellaTest {
    @Test
    public void testHerencia() {
        tipusCasella mina = new Mina();
        tipusCasella normal = new Normal();

        assertTrue("Mina debe ser instancia de tipusCasella", mina instanceof tipusCasella);
        assertTrue("Normal debe ser instancia de tipusCasella", normal instanceof tipusCasella);
    }

    @Test
    public void testPolimorfismo() {
        tipusCasella casella = new Mina();
        assertTrue("El método esBomba debe devolver true para Mina", casella.isBomb());


    }
}
```

```java
public abstract class tipusCasella {
```

Comentaris addicionals de la classe TipusCasella
S'ha fet el 100% de coverage

La classe tipusCasella assegura una estructura modular i extensible per afegir futurs tipus de caselles.

Crear i inicialitzar el tauler de joc

Descripció: S'ha implementat la classe Tauler, que permet crear i gestionar un tauler de joc amb les dimensions i nombre de mines especificats. La classe assegura el correcte posicionament de mines i el manteniment dels comptadors de caselles mostrades i bombes.
Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.model
- Classe: Tauler
- Mètodes principals: Tauler(), setRandomBombs(), mostrarTaulell()

Test: Per la inicialització hem fet varis tests, fent statement coverage i les proves de caixa negra amb valors límit i frontera. Hem fet els tests a TaulerTest per comprovar que és correcte la creació i inicialització del tauler. A més hem afegit el disseny per contracte per assegurar que els paràmetres d'entrada i sortida són correctes.També tenim un pairwise testing en el mètode testSetRandomBombsPairwise() on fem servir les diferents possiblitats i parella de valors per provar més casos. També complim la decision i condition coverage.

```java
@Test
void testConstructorInvalidWidth() {
    Tauler tauler = null;

    System.out.println("La amplada debe ser mayor que 0.");

    assertNull(tauler, "El tablero no debe crearse con amplada inválida.");
}


@Test
void testConstructorInvalidHeight() {
    Tauler tauler = null;

    System.out.println("La altura debe ser mayor que 0.");

    assertNull(tauler, "El tablero no debe crearse con altura inválida.");
}
```

```java
@Test
void testAddBombaIncrementNeighbours() {
    tauler.addBomba(2, 2);
    assertEquals(1, tauler.getCasella(1, 1).getVeins(), "El vecino (1, 1) debe tener 1 bomba cercana.");
    assertEquals(1, tauler.getCasella(3, 3).getVeins(), "El vecino (3, 3) debe tener 1 bomba cercana.");

    tauler.addBomba(0, 0);
    assertEquals(2, tauler.getCasella(1, 1).getVeins(), "El vecino (1, 1) debe tener 2 bomba cercana.");
    assertEquals(1, tauler.getCasella(0, 1).getVeins(), "El vecino (0, 1) debe tener 1 bomba cercana.");

    tauler.addBomba(4, 4);
    assertEquals(2, tauler.getCasella(3, 3).getVeins(), "El vecino (3, 3) debe tener 2 bomba cercana.");


    tauler.addBomba(4, 3);
    assertEquals(3, tauler.getCasella(3, 3).getVeins(), "El vecino (3, 3) debe tener 3 bomba cercana.");


    tauler.addBomba(3, 4);
    assertEquals(4, tauler.getCasella(3, 3).getVeins(), "El vecino (3, 3) debe tener 4 bomba cercana.");

    tauler.addBomba(0, 1);
    assertEquals(1, tauler.getCasella(0, 2).getVeins(), "El vecino (0, 2) debe tener 1 bomba cercana.");


    tauler.addBomba(1, 0);
    assertEquals(4, tauler.getCasella(1, 1).getVeins(), "El vecino (1, 1) debe tener 4 bomba cercana.");
    assertEquals(0, tauler.getCasella(0, 4).getVeins(), "El vecino (0, 4) debe tener 0 bomba cercana.");


}
```

```
*/
public Tauler(int amplada, int altura, int mines) {

        //precondiciones
        //assert amplada > 0 : "La amplada debe ser mayor que 0";
        //assert altura > 0 : "La altura debe ser mayor que 0";
       // assert mines >= 0 : "El número de minas debe ser no negativo";
       // assert mines <= amplada * altura : "El número de minas no puede exceder el total de casillas";

        //implementacion
        this.amplada = amplada;
        this.altura = altura;
        this.nMines = mines;

        caselles = new Casella[amplada][altura];

        for(int y = 0; y < altura; y++) {
            for(int x = 0; x < amplada; x++) {
                caselles[x][y] = new Casella();
            }
        }

        bombCount = 0;

        casellesMostrades = 0;

        //postcondiciones
        //assert amplada > 0 : "La amplada debe ser mayor que 0";
       // assert altura > 0 : "La altura debe ser mayor que 0";
       // assert mines >= 0 : "El número de minas debe ser no negativo";
        //assert mines <= amplada * altura : "El número de minas no puede exceder el total de casillas";
    }
```

```
        //precondiciones
        assert amplada > 0 : "La amplada debe ser mayor que 0";
        assert altura > 0 : "La altura debe ser mayor que 0";
        assert mines >= 0 : "El número de minas debe ser no negativo";
        assert mines <= amplada * altura : "El número de minas no puede exceder el total de casillas";

        //implementacion
        this.amplada = amplada;
        this.altura = altura;
        this.nMines = mines;

        caselles = new Casella[amplada][altura];

        for(int y = 0; y < altura; y++) {
            for(int x = 0; x < amplada; x++) {
                caselles[x][y] = new Casella();
            }
        }

        bombCount = 0;

        casellesMostrades = 0;

        //postcondiciones
        assert amplada > 0 : "La amplada debe ser mayor que 0";
        assert altura > 0 : "La altura debe ser mayor que 0";
        assert mines >= 0 : "El número de minas debe ser no negativo";
        assert mines <= amplada * altura : "El número de minas no puede exceder el total de casillas";
    }
```

```java
@Test
void testSetRandomBombsPairwise() {
    // Combinaciones de parámetros
    int[][] testCases = {
        {1, 1, 0}, // Caso límite: tablero pequeño sin bombas
        {1, 1, 1}, // Caso límite: tablero pequeño con una bomba
        {3, 3, 0}, // Tablero mediano sin bombas
        {3, 3, 1}, // Tablero mediano con una bomba
        {3, 3, 8}, // Tablero mediano con todas las bombas posibles menos una
        {5, 5, 0}, // Tablero grande sin bombas
        {5, 5, 10}, // Tablero grande con algunas bombas
        {5, 5, 24}  // Tablero grande con todas las bombas posibles menos una
    };

    for (int[] testCase : testCases) {
        int amplada = testCase[0];
        int altura = testCase[1];
        int mines = testCase[2];

        Tauler board = new Tauler(amplada, altura, mines);
        board.setRandomBombs();

        // Verificar el número de bombas después de colocarlas
        int expectedBombCount = Math.min(mines, amplada * altura);
        assertEquals(expectedBombCount, board.getBombCount(),
            "Fallo en tablero de tamaño " + amplada + "x" + altura + " con " + mines + " minas.");
    }
}
```

Exemple de pairwise testing.

```java
@Test
void testCondicionMostrarCasellaConVecinos() {
    tauler.addBomba(1, 1); // Agregamos una bomba cerca
    tauler.mostrarCasella(0, 0); // Mostramos una casilla con vecinos
    assertTrue(tauler.getCasella(0, 0).getEsVisible(), "La casilla debe mostrarse.");
    assertFalse(tauler.getCasella(1, 1).getEsVisible(), "La casilla con bomba no debe mostrarse automáticamente.");
}

@Test
void testCondicionMostrarCasellaSinVecinos() {
    tauler.mostrarCasella(2, 2); // Mostramos una casilla sin vecinos
    assertTrue(tauler.getCasella(2, 2).getEsVisible(), "La casilla debe mostrarse.");
    assertTrue(tauler.getCasella(1, 1).getEsVisible(), "Las casillas adyacentes sin bombas deben revelarse.");
}

@Test
void testCondicionRevelaCasellesNoVisible() {
    tauler.mostrarCasella(0, 0); // Intentamos revelar una casilla no visible
    assertTrue(tauler.getCasella(0, 0).getEsVisible(), "La casilla debe mostrarse.");
}

@Test
void testCondicionRevelaCasellesYaVisible() {
    tauler.mostrarCasella(0, 0); // Revelamos la casilla una vez
    tauler.mostrarCasella(0, 0); // Intentamos revelarla nuevamente
    assertTrue(tauler.getCasella(0, 0).getEsVisible(), "La casilla ya revelada debe permanecer visible.");
}
```

test del condition i decision coverage.

```java
if(!caselles[a][b].getEsVisible()) {
```

aplicació del decision i condition coverage

Revelar caselles adjacents

Descripció: El mètode mostrarCasella() permet descobrir una casella del tauler. Si aquesta casella no té bombes adjacents, es revelen també les caselles adjacents mitjançant el mètode privat revelaCasellesAdjuntes().

Localització:

- Classe: Tauler
- Arxiu: main.java.cat.uab.tq.buscamines.model

- Mètodes: mostrarCasella(int x, int y), revelaCasellesAdjuntes(int x, int y)

Test: Els tests es troben a TaulerTest i són principalment els tests: testMostrarCasellaNoNeighbours(), testMostrarCasellaWithNeighbours(), testRevelaCasellesAdjuntesBuides, testRevelaCasellesAdjuntesEsquina, testRevelaCasellesAdjuntesCentre(), testRevelaCasellesAdjuntesBombesProperes(), testRevelaCasellesAdjuntesAmbBombes():
Apliquem Statement coverage i proves de caixa negra amb valors límit i frontera. També fem disseny per contracte al mètode mostrarCasella().

```java
@Test
void testMostrarCasellaNoNeighbours() {
    tauler.mostrarCasella(2, 2);
    assertTrue(tauler.getCasella(2, 2).getEsVisible(), "La casilla (2, 2) debe mostrarse.");

    tauler.mostrarCasella(0, 0);
    assertTrue(tauler.getCasella(0, 0).getEsVisible(), "La casilla (0, 0) debe mostrarse.");

    tauler.mostrarCasella(0, 1);
    assertTrue(tauler.getCasella(0, 1).getEsVisible(), "La casilla (0, 1) debe mostrarse.");

    tauler.mostrarCasella(1, 0);
    assertTrue(tauler.getCasella(1, 0).getEsVisible(), "La casilla (1, 0) debe mostrarse.");

    tauler.mostrarCasella(3, 3);
    assertTrue(tauler.getCasella(3, 3).getEsVisible(), "La casilla (3, 3) debe mostrarse.");

    tauler.mostrarCasella(4, 4);
    assertTrue(tauler.getCasella(4, 4).getEsVisible(), "La casilla (4, 4) debe mostrarse.");

    tauler.mostrarCasella(3, 4);
    assertTrue(tauler.getCasella(3, 4).getEsVisible(), "La casilla (3, 4) debe mostrarse.");


    tauler.mostrarCasella(4, 3);
    assertTrue(tauler.getCasella(4, 3).getEsVisible(), "La casilla (4, 3) debe mostrarse.");



}
```

```java
    @Test
    void testMostrarCasellaWithNeighbours() {
        tauler.addBomba(1, 1);
        tauler.mostrarCasella(0, 0);
        assertTrue(tauler.getCasella(0, 0).getEsVisible(), "La casilla (0, 0) debe mostrarse.");
        assertFalse(tauler.getCasella(1, 1).getEsVisible(), "La casilla (1, 1) no debe mostrarse automáticamente.");


        tauler.addBomba(2, 2);
        tauler.mostrarCasella(3, 3);
        assertTrue(tauler.getCasella(3, 3).getEsVisible(), "La casilla (3, 3) debe mostrarse.");
        assertFalse(tauler.getCasella(2, 2).getEsVisible(), "La casilla (2, 2) no debe mostrarse si tiene una bomba.");


    }

    @Test
    void testRevelaCasellesAdjuntesBuides() {
        tauler.mostrarCasella(2, 2);
        assertTrue(tauler.getCasella(1, 1).getEsVisible(), "La casilla adyacente (1, 1) debe mostrarse.");
        assertTrue(tauler.getCasella(3, 3).getEsVisible(), "La casilla adyacente (3, 3) debe mostrarse.");
    }


    @Test
    void testRevelaCasellesAdjuntesEsquina() {
        tauler.mostrarCasella(0, 0);
        assertTrue(tauler.getCasella(0, 0).getEsVisible(), "La casilla (0, 0) debe mostrarse.");
        assertTrue(tauler.getCasella(1, 1).getEsVisible(), "La casilla (1, 1) debe mostrarse si está vacía.");
    }

    @Test
    void testRevelaCasellesAdjuntesCentre() {
        tauler.mostrarCasella(2, 2);
        assertTrue(tauler.getCasella(1, 1).getEsVisible(), "La casilla (1, 1) debe mostrarse.");
        assertTrue(tauler.getCasella(3, 3).getEsVisible(), "La casilla (3, 3) debe mostrarse.");
        assertFalse(tauler.getCasella(4, 4).getEsVisible(), "Casillas no adyacentes no deben mostrarse.");
    }
```

```java
@Test
void testRevelaCasellesAdjuntesBuides() {
    tauler.mostrarCasella(2, 2);
    assertTrue(tauler.getCasella(1, 1).getEsVisible(), "La casilla adyacente (1, 1) debe mostrarse.");
    assertTrue(tauler.getCasella(3, 3).getEsVisible(), "La casilla adyacente (3, 3) debe mostrarse.");
}


@Test
void testRevelaCasellesAdjuntesEsquina() {
    tauler.mostrarCasella(0, 0);
    assertTrue(tauler.getCasella(0, 0).getEsVisible(), "La casilla (0, 0) debe mostrarse.");
    assertTrue(tauler.getCasella(1, 1).getEsVisible(), "La casilla (1, 1) debe mostrarse si está vacía.");
}

@Test
void testRevelaCasellesAdjuntesCentre() {
    tauler.mostrarCasella(2, 2);
    assertTrue(tauler.getCasella(1, 1).getEsVisible(), "La casilla (1, 1) debe mostrarse.");
    assertTrue(tauler.getCasella(3, 3).getEsVisible(), "La casilla (3, 3) debe mostrarse.");
    assertFalse(tauler.getCasella(4, 4).getEsVisible(), "Casillas no adyacentes no deben mostrarse.");
}

@Test
void testRevelaCasellesAdjuntesBombesProperes() {
    tauler.addBomba(2, 2);
    tauler.mostrarCasella(3, 3);
    assertTrue(tauler.getCasella(3, 3).getEsVisible(), "La casilla (3, 3) debe mostrarse.");
    assertFalse(tauler.getCasella(2, 2).getEsVisible(), "La casilla (2, 2) no debe revelarse automáticamente si tiene una bomba.");
}
@Test
void testRevelaCasellesAdjuntesAmbBombes() {
    tauler.addBomba(1, 1);
    tauler.mostrarCasella(0, 0);
    assertTrue(tauler.getCasella(0, 0).getEsVisible(), "La casilla (0, 0) debe mostrarse.");
    assertFalse(tauler.getCasella(1, 1).getEsVisible(), "La casilla (1, 1) no debe mostrarse automáticamente.");
}
```

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

    //precondiciones
    //assert x >= 0 && x < amplada : "Coordenada x fuera de rango";
    //assert y >= 0 && y < altura : "Coordenada y fuera de rango";

  //implementacion
    if(caselles[x][y].getVeins() != 0) {
        casellesMostrades++;
        caselles[x][y].mostrarCasella();
    }else {
        revelaCasellesAdjuntes(x, y);
    }

    //postcondiciones
    // assert casella[x][y].getEsVisible() : "Postcondición fallida: La casilla no es visible";
    // assert casellesMostrades + bombCount <= amplada * altura : "Postcondición fallida: Casillas mostradas exceden el límite";
    // assert invariants() : "Invariantes fallidas tras mostrar casilla";
}

private void revelaCasellesAdjuntes(int x, int y) {
    // TODO Auto-generated method stub

    int minX = Math.max(0, x-1);
    int maxX = Math.min(amplada - 1, x+1);
    int minY = Math.max(0, y-1);
    int maxY = Math.min(altura-1, y+1);

    for(int b = minY; b <= maxY; b++) {
        for(int a = minX; a <= maxX; a++) {
            if(!caselles[a][b].getEsVisible()) {

                caselles[a][b].mostrarCasella();
                casellesMostrades++;
            }
        }
    }
}
```

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

    //precondiciones
    assert x >= 0 && x < amplada : "Coordenada x fuera de rango";
    assert y >= 0 && y < altura : "Coordenada y fuera de rango";

  //implementacion
    if(caselles[x][y].getVeins() != 0) {
        casellesMostrades++;
        caselles[x][y].mostrarCasella();
    }else {
        revelaCasellesAdjuntes(x, y);
    }

    //postcondiciones
     assert caselles[x][y].getEsVisible() : "Postcondición fallida: La casilla no es visible";
     assert casellesMostrades + bombCount <= amplada * altura : "Postcondición fallida: Casillas mostradas exceden el límite";
    assert invariants() : "Invariantes fallidas tras mostrar casilla";
}

private void revelaCasellesAdjuntes(int x, int y) {
```

Afegir bombes manualment i aleatòriament

Descripció: La classe permet afegir bombes de manera manual mitjançant el mètode addBomba(int x, int y) o de forma aleatòria amb el mètode setRandomBombs().

Localització:

- Classe: Tauler
- Mètodes: addBomba(int x, int y), setRandomBombs()

Test: fem els tests a la classe TaulerTests i fem un statement coverage, a més de proves de caixa negra amb els valors límit i frontera i fem ús del disseny per contracte com es veu a les imatges. Aqui també fem loop testing anidat al test testSetRandomBombsMaxIterations(). Afegim un looping test de loop simple a setRandomBombs i un altre loop simple al afegir bombes manualment. També tenim un Path coverage per afegir les bombes manualment al addBomba().

```java
class TaulerTest {

    private Tauler tauler;


    @BeforeEach
    void setUp() {
        tauler = new Tauler(5, 5, 10); // Tablero 5x5 con 10 minas
    }

    @Test
    void testAddBombaValid() {
        tauler.addBomba(2, 2);
        assertTrue(tauler.getCasella(2, 2).isBomb(), "La casilla (2, 2) debe contener una bomba.");

        tauler.addBomba(0, 0);
        assertTrue(tauler.getCasella(0, 0).isBomb(),"La casilla (0, 0) debe contener una bomba.");

        tauler.addBomba(4, 4);
        assertTrue(tauler.getCasella(4, 4).isBomb(),"La casilla (4, 4) debe contener una bomba.");

        tauler.addBomba(4, 3);
        assertTrue(tauler.getCasella(4, 3).isBomb(),"La casilla (4, 4) debe contener una bomba.");

        tauler.addBomba(3, 4);
        assertTrue(tauler.getCasella(3, 4).isBomb(),"La casilla (4, 4) debe contener una bomba.");

        tauler.addBomba(0, 1);
        assertTrue(tauler.getCasella(0, 1).isBomb(),"La casilla (4, 4) debe contener una bomba.");

        tauler.addBomba(1, 0);
        assertTrue(tauler.getCasella(0, 1).isBomb(),"La casilla (4, 4) debe contener una bomba.");


    }
```

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

      //precondiciones
      //assert x >= 0 && x < amplada : "Coordenada x fuera de rango";
     // assert y >= 0 && y < altura : "Coordenada y fuera de rango";
    //  assert !caselles[x][y].isBomb() : "La casilla ya contiene una bomba";


      //implementacio
      int minX = Math.max(0, x-1);
      int maxX = Math.min(amplada - 1, x+1);
      int minY = Math.max(0, y-1);
      int maxY = Math.min(altura-1, y+1);

      for(int b = minY; b <= maxY; b++) {
          for(int a = minX; a <= maxX; a++) {
              caselles[a][b].addVei();
          }
      }

      caselles[x][y].setBomb();
      bombCount++;

    //postcondiciones
     // assert caselles[x][y].isBomb() : "Postcondición fallida: La casilla no tiene una bomba";
      //assert bombCount <= amplada * altura : "Postcondición fallida: Demasiadas bombas";
     // assert invariants() : "Invariantes fallidas tras añadir bomba";

  }
```

```java
public void setRandomBombs() {

    //precondiciones
    //assert nMines <= amplada * altura : "No hay suficiente espacio para las bombas";

    //implementacio
    Random rand = new Random();
    int x, y;
    for(int i = 0; i < nMines; i++) {
        do {
            x = rand.nextInt(amplada);
            y = rand.nextInt(altura);
        }while(caselles[x][y].isBomb());

        addBomba(x, y);
    }

    //postcondiciones

    //assert bombCount == nMines : "Postcondición fallida: Número incorrecto de bombas";
   // assert invariants() : "Invariantes fallidas tras colocar bombas";
}

public void mostrarTaulell() {
```

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

    //precondiciones
    assert x >= 0 && x < amplada : "Coordenada x fuera de rango";
    assert y >= 0 && y < altura : "Coordenada y fuera de rango";
   assert !caselles[x][y].isBomb() : "La casilla ya contiene una bomba";


    //implementacio
    int minX = Math.max(0, x-1);
    int maxX = Math.min(amplada - 1, x+1);
    int minY = Math.max(0, y-1);
    int maxY = Math.min(altura-1, y+1);

    for(int b = minY; b <= maxY; b++) {
        for(int a = minX; a <= maxX; a++) {
            caselles[a][b].addVei();
        }
    }

    caselles[x][y].setBomb();
    bombCount++;

  //postcondiciones
   assert caselles[x][y].isBomb() : "Postcondición fallida: La casilla no tiene una bomba";
   assert bombCount <= amplada * altura : "Postcondición fallida: Demasiadas bombas";
   assert invariants() : "Invariantes fallidas tras añadir bomba";

}
```

```java
public void setRandomBombs() {

    //precondiciones
    assert nMines <= amplada * altura : "No hay suficiente espacio para las bombas";

    //implementacio
    Random rand = new Random();
    int x, y;
    for(int i = 0; i < nMines; i++) {
        do {
            x = rand.nextInt(amplada);
            y = rand.nextInt(altura);
        }while(caselles[x][y].isBomb());

        addBomba(x, y);
    }

    //postcondiciones

    assert bombCount == nMines : "Postcondición fallida: Número incorrecto de bombas";
    assert invariants() : "Invariantes fallidas tras colocar bombas";
}
```

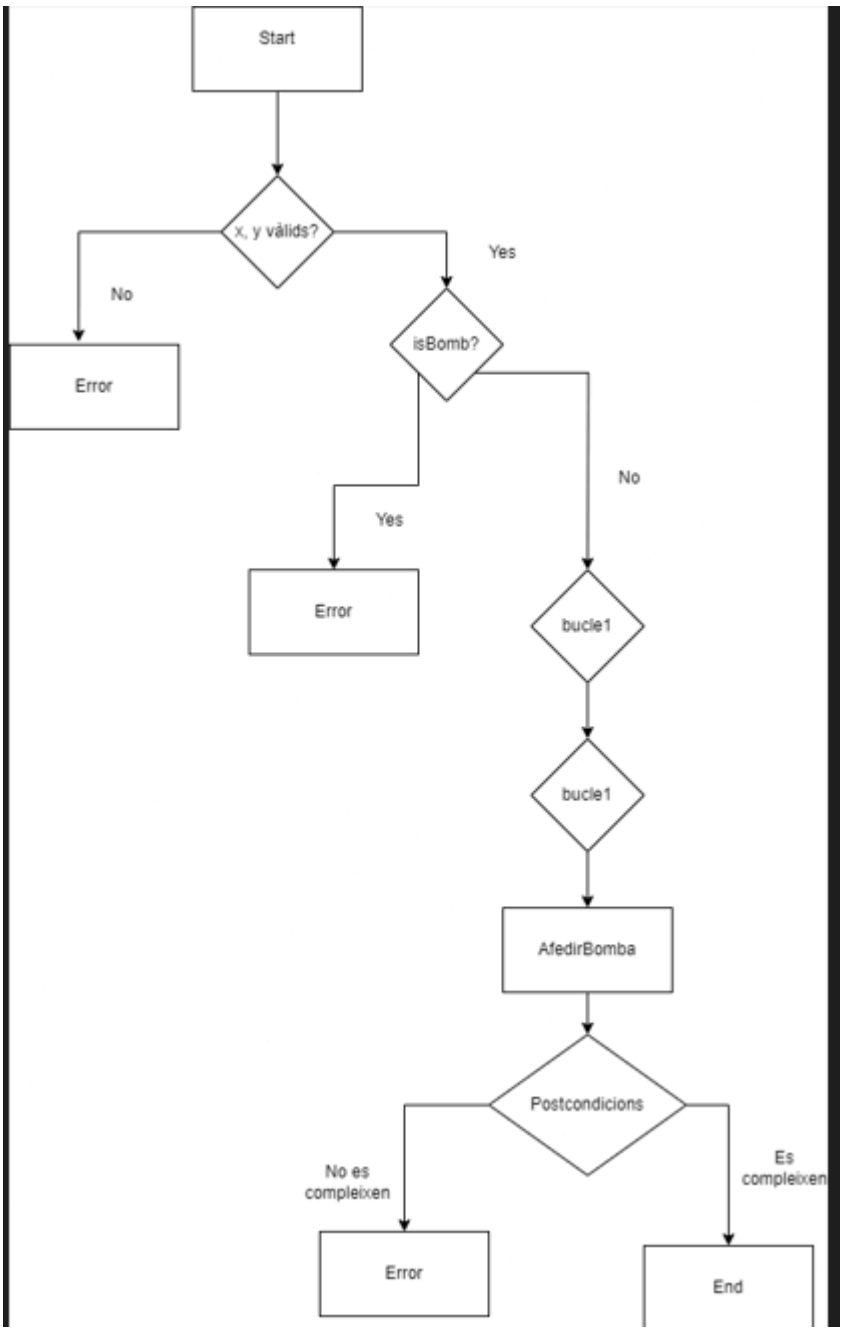
```java
void testSetRandomBombsMaxIterations() {
    Tauler fullBombTauler = new Tauler(5, 5, 25); // Tablero completamente lleno de minas
    fullBombTauler.setRandomBombs();
    assertEquals(25, fullBombTauler.getBombCount(), "Debe haber exactamente 25 bombas, una por cada casilla.");

    // Validamos que todas las casillas son bombas
    for (int y = 0; y < fullBombTauler.getAltura(); y++) {
        for (int x = 0; x < fullBombTauler.getAmplada(); x++) {
            assertTrue(fullBombTauler.isBomb(x, y), "Cada casilla debe ser una bomba.");
        }
    }
}
```

```java
    @Test
    void testSetRandomBombs_SinIteraciones() {
        Tauler tauler = new Tauler(3, 3, 0); // Tablero de 3x3 sin minas.
        tauler.setRandomBombs(); // No debería colocar ninguna bomba.

        assertEquals(0, tauler.getBombCount(), "No debería haber bombas en el tablero.");
    }

    @Test
    void testSetRandomBombs_UnaIteracion() {
        Tauler tauler = new Tauler(3, 3, 1); // Tablero de 3x3 con 1 mina.
        tauler.setRandomBombs();

        assertEquals(1, tauler.getBombCount(), "Debería haber exactamente una bomba en el tablero.");
    }

    @Test
    void testSetRandomBombs_MultiplesIteraciones() {
        Tauler tauler = new Tauler(3, 3, 3); // Tablero de 3x3 con 3 minas.
        tauler.setRandomBombs();

        assertEquals(3, tauler.getBombCount(), "Debería haber exactamente 3 bombas en el tablero.");
    }

}
```

loop simple del looping test.

```java
    @Test
    void testAddBomba_SinVecinos() {
        Tauler tauler = new Tauler(1, 1, 0); // Tablero de 1x1.
        tauler.addBomba(0, 0); // Colocar una bomba en la única casilla.

        assertTrue(tauler.isBomb(0, 0), "La casilla (0,0) debería tener una bomba.");
        assertEquals(1, tauler.getBombCount(), "Debería haber exactamente una bomba en el tablero.");
    }

    @Test
    void testAddBomba_ConVecinos() {
        Tauler tauler = new Tauler(3, 3, 0); // Tablero de 3x3.
        tauler.addBomba(1, 1); // Colocar una bomba en el centro.

        // Verificar que todos los vecinos del centro tienen un vecino bomba.
        for (int x = 0; x < 3; x++) {
            for (int y = 0; y < 3; y++) {
                if (x != 1 || y != 1) { // Evitar la casilla bomba.
                    assertEquals(1, tauler.getCasella(x, y).getVeins(),
                                 "La casilla (" + x + "," + y + ") debería tener exactamente un vecino bomba.");
                }
            }
        }
    }
}
```

un altre loop simple.



path coverage de addBomba().

```java
@Test
void testAddBomba_coordenadaFueraDeRango() {
    Tauler tauler = new Tauler(5, 5, 3);
    assertThrows(AssertionError.class, () -> tauler.addBomba(-1, 0));
    assertThrows(AssertionError.class, () -> tauler.addBomba(5, 0));
    assertThrows(AssertionError.class, () -> tauler.addBomba(0, -1));
    assertThrows(AssertionError.class, () -> tauler.addBomba(0, 5));
}

@Test
void testAddBomba_casillaYaTieneBomba() {
    Tauler tauler = new Tauler(5, 5, 3);
    tauler.addBomba(2, 2);
    assertThrows(AssertionError.class, () -> tauler.addBomba(2, 2));
}

@Test
void testAddBomba_exito() {
    Tauler tauler = new Tauler(5, 5, 3);
    tauler.addBomba(2, 2);

    assertTrue(tauler.isBomb(2, 2), "La casilla (2, 2) debería contener una bomba.");
    assertEquals(1, tauler.getBombCount(), "El número de bombas debería ser 1.");
    assertEquals(1, tauler.getCasella(1, 1).getVeins(), "Los vecinos deberían haberse actualizado.");
}
```

test del path coverage.

Verificar estat del joc

Descripció: El mètode isWon() comprova si el jugador ha descobert totes les caselles sense bomba, indicant que ha guanyat la partida.
Localització:

- Classe: Tauler
- Mètode: isWon()

Test: a la classe TaulerTest amb statement coverage i proves de caixa negra .

```java
@Test
void testIsWonTotesCasellesMostrades() {
    for (int y = 0; y < tauler.getAltura(); y++) {
        for (int x = 0; x < tauler.getAmplada(); x++) {
            if (!tauler.getCasella(x, y).isBomb()) {
                tauler.mostrarCasella(x, y);
            }
        }
    }
    assertTrue(tauler.isWon(), "El juego debe ser ganado si todas las casillas seguras están reveladas.");
}

@Test
void testNotWon() {
    tauler.mostrarCasella(0, 0);
    assertFalse(tauler.isWon(), "El juego no debe ser ganado si quedan casillas seguras sin revelar.");
}
```

```java
public boolean isWon() {
    return casellesMostrades+ bombCount == amplada * altura;
}
```

Comentaris addicionals de la classe Tauler:

Hem fet el 100% de coverage de la classe

Entrada de datos del usuario

Descripció: Es va implementar la classe entryData per gestionar la entrada de dades per part del usuari, incloent la configuració inicial del tauler y las coordenadas de las casillas a descubrir. La clase verifica la validez de la entrada y maneja excepciones comunes.

Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.vista.entryData
- Classes: entryData
- Mètodes: readInpuy i configurarPartida

Test: Es troben a entryDataTest. Tenim statement coverage, proves de caixa negra, disseny per contracte i un 100% de coverage de la classe.

```java
package main.java.cat.uab.tq.buscamines.vista;
import main.java.cat.uab.tq.buscamines.controlador.*;

import java.io.InputStream;
import java.util.Scanner;

public class entryData {
    private Scanner scan = null;

    // Constructor que permite inyección de InputStream
    public entryData(InputStream inputStream) {
        scan = new Scanner(inputStream);
    }

    // Constructor por defecto para uso normal (utiliza System.in)
    public entryData() {
        this(System.in);
    }

    //Get auxiliar para realizar el test del constructor sin parametros
    public Scanner getScan() {
        return scan;
    }

    public int[] readInput() {

        int num1 = 0, num2 = 0;
        boolean valid1 = false, valid2 = false;

        do {

            System.out.println("Entra la coordenada X y Y separadas por un espacio: ");
            String input = scan.nextLine().trim();

            Scanner lineScanner = new Scanner(input); // Escanea la línea para números

            if (lineScanner.hasNextInt()) {
                num1 = lineScanner.nextInt();
                valid1 = true;
            }

            if (lineScanner.hasNextInt()) {
                num2 = lineScanner.nextInt();
                valid2 = true;
            }


            if (!valid1 || !valid2) {
                System.out.println("Coordenada invalida.\n");
            }

            lineScanner.close(); // Cierra el scanner de la línea


        } while (!valid1 || !valid2);

        int[] inputCoordinates = {num1, num2};

        return inputCoordinates;
```

```java
public int[] configurarPartida(int maxFiles, int minFiles, int maxCol, int minCol, int maxMines, int minMines) {

    //precondicions
    assert(minFiles > 0 && maxFiles <= 10) : "El numero maximo o minimo de filas es incorrecto.";
    assert(minCol > 0 && maxCol <= 10) : "El numero maximo o minimo de columnas es incorrecto.";
    assert(minMines > 0 && maxMines <= 10) : "El numero maximo o minimo de minas es incorrecto.";

    System.out.println("Entra las dimensiones deseadas del tablero, "
            + "el numero de filas y columnas separado por un espacio: ");

    int num1 = getIntInput(scan);
    int num2 = getIntInput(scan);

    // Postcondiciones para filas y columnas
    assert (num1 > minFiles && num1 < maxFiles) : "El número de filas debe estar entre los intervalos permitidos";
    assert (num2 > minCol && num2 < maxCol) : "El número de columnas debe estar entre los intervalos permitidos";


    System.out.println("Entra el numero de minas deseado: ");

    int mines = getIntInput(scan);

    //postcondicio
    assert (mines < maxMines && mines > minMines) : "El numero de minas debe estar entre los intervalos permitidos";

    int[] config = {num1, num2, mines};
    return config;
}
```

Disseny per contracte.

```java
    @Test
    void testReadInputValid() {

        //Creamos un input valido simulado y lo guardamos en un ByteaArrayOutputStream
        String automatedInput = "2 3\n";
        ByteArrayInputStream inputStream = new ByteArrayInputStream(automatedInput.getBytes());

        //Creamos una salida para detectar que los mensajes se imprimen correctamente
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        PrintStream originalOut = System.out;
        System.setOut(new PrintStream(outputStream));


        entryData entry = new entryData(inputStream);


        int[] res = entry.readInput();


        System.setOut(originalOut);
        String output = outputStream.toString();

        assertNotNull(res);
        assertEquals(2, res.length);
        assertEquals(2, res[0]);
        assertEquals(3, res[1]);

        assertEquals(output.trim(), "Entra la coordenada X y Y separadas por un espacio:");
    }

    @Test
    void testReadInputInvalid(){
        //Creamos un input simulado que sea invalido y lo guardamos en un ByteaArrayOutputStream
        String automatedInput = "c121A\n" + "1 2";
        ByteArrayInputStream inputStream = new ByteArrayInputStream(automatedInput.getBytes());

        //Creamos una salida para detectar que los mensajes se imprimen correctamente
        ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
        PrintStream originalOut = System.out;
        System.setOut(new PrintStream(outputStream));


        entryData entry = new entryData(inputStream);


        int[] res = entry.readInput();


        System.setOut(originalOut);
        String output = outputStream.toString();

        assertNotNull(res);
        assertEquals(2, res.length);
        assertEquals(1, res[0]);
        assertEquals(2, res[1]);

        assertTrue(output.contains("Entra la coordenada X y Y separadas por un espacio:"));
        assertTrue(output.contains("Coordenada invalida.\n"));
    }
```

Proves de caixa negra.

Mensajes del juego

Descripció: La classe MessageTxt proporciona missatges de text que indican l'estat del joc, com benvinguda, victoria i derrota.
Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.vista.MessageTxt
- Classes: MessageTxt
- Mètodes: presentacio, victory, bomb

Tests: estan realitzats a MesssageTxtTest i tenim statement coverage i proves de caixa negra. No té sentit aplicar disseny per contracte a aquesta classe. També hem realitzat un

mock manual per comprovar que els missatges surten correctament sense necessitat de cridar desde la classe Partida, un mock anomenat MessageTxtMock.

```java
@BeforeEach
void setUp() {
    System.setOut(new PrintStream(outputStream)); // Redirige la salida estándar
}

@Test
void testPresentacio() {
    // Creamos una instancia de MessageTxt
    MessageTxt messageTxt = new MessageTxt();

    // Llamamos al método
    messageTxt.presentacio();

    // Comprobamos si la salida contiene el mensaje esperado
    String output = outputStream.toString().trim();
    assertEquals("¡Bienvenido al juego del Buscaminas!", output);
}

@Test
void testVictory() {
    // Creamos una instancia de MessageTxt
    MessageTxt messageTxt = new MessageTxt();

    // Llamamos al método
    messageTxt.victory();

    // Comprobamos si la salida contiene el mensaje esperado
    String output = outputStream.toString().trim();
    assertEquals("¡Felicidades, has ganado!", output);
}

@Test
void testBomb() {
    // Creamos una instancia de MessageTxt
    MessageTxt messageTxt = new MessageTxt();

    // Llamamos al método
    messageTxt.bomb();

    // Comprobamos si la salida contiene el mensaje esperado
    String output = outputStream.toString().trim();
    assertEquals("¡Boom! Has pisado una mina. Fin del juego.", output);
}
```

```java
package Vista;

public class MessageTxt {

    public void presentacio() {
        System.out.println("¡Bienvenido al juego del Buscaminas!");
    }

    public void victory() {
        System.out.println("¡Felicidades, has ganado!");
    }

    public void bomb() {
        System.out.println("¡Boom! Has pisado una mina. Fin del juego.");
    }
}
```

```java
import main.java.cat.uab.tq.buscamines.vista.MessageTxt;

public class MessageTxtMock extends MessageTxt{
    private StringBuilder output;

    public MessageTxtMock() {
        this.output = new StringBuilder();
    }

    @Override
    public void presentacio() {
        output.append("Benvingut al joc del buscamines!\n");
    }

    @Override
    public void victory() {
        output.append("Victoria!! Has conseguit revelar totes les caselles sense bombes.\n"
    }

    @Override
    public void bomb() {
        output.append("Oops.. Has topat amb una bomba :(\n");
        output.append("GAME OVER\n");
    }

    // Método para acceder al contenido almacenado
    public String getOutput() {
        return output.toString();
    }

    // Método para resetear el contenido almacenado
    public void resetOutput() {
        output.setLength(0);
    }

}
```

mock de MessageTxt i el seu testing.

```java
public class MessageTxtMockTest {

    public static void main(String[] args) {
        // Crear una instancia del mock
        MessageTxtMock mock = new MessageTxtMock();

        // Llamar a los métodos simulados
        mock.presentacio();
        mock.victory();
        mock.bomb();

        // Verificar el contenido generado
        String output = mock.getOutput();
        System.out.println("Contenido del mock:");
        System.out.println(output);

        // Puedes usar assertions para validar
        assert output.contains("Benvingut al joc del buscamines!");
        assert output.contains("Victoria!! Has conseguit revelar totes les caselles sense bombes.");
        assert output.contains("Oops.. Has topat amb una bomba :(");
    }
}
```

Comentaris addicionals de la classe MessageTxt:
Tenim un 100% de coverage de la classe.

Visualización del tablero
Descripció: La classe VistaTauler permet visualitzar l'estat del tauler en consola i validar les posicions seleccionades per l'usuari..
Localització:

- Arxiu: main.java.cat.uab.tq.buscamines.vista.VistaTauler
- Classes: VistaTauler
- Mètodes: printTauler validarPosicio printStatus

Test: Tenim els tests a la classe VistaTaulerTest, també hem implementat un Mock anomenat MockTauler implementat manualment per poder determinar les sortides de VistaTauler sense haver de fer cada prova manualment de partida i poder ficar directament els resultats. També tenim el statement coverage i proves de caixa negra. També tenim implementat 2 condition i 2 decision coverage en un if de validarPosicio() i un loop testing anidat al for de printTauler().

```
public VistaTauler() { }

public void printTauler(Tauler tauler) {
    for(int y = 0; y < tauler.getAltura(); y++) {
        for(int x = 0; x < tauler.getAmplada(); x++) {
            System.out.print(tauler.getCasella(x, y) + " ");
        }
        System.out.println(" |"+ y);
    }

    for(int x = 0; x < tauler.getAmplada(); x++) {
        System.out.print("_ ");
    }
    System.out.println();

    for(int x = 0; x < tauler.getAmplada(); x++) {
        System.out.print(x);
        if(x+1 < 10) {
            System.out.print(" ");
        }
    }
    System.out.println("\n");

}

public boolean validarPosicio(Tauler tauler, int x, int y) {
    if((x < 0 || x >= tauler.getAmplada()) || (y < 0 || y >= tauler.getAltura())) {
        System.out.println("Esta casilla no es correcta! \n");
        return false;
    }

    if(tauler.casellaVisible(x, y)) {
        System.out.println("Esta casilla ya esta destapada! \n");
        return false;
    }

    return true;
}

public void printStatus(Tauler tauler) {
    int a = tauler.getCasellesMostrades();
    int b = tauler.getBombCount();
    int c = tauler.getAmplada() * tauler.getAltura();
```

```java
@Test
public void testPrintTauler() {
    // Caso 1: Tablero vacío
    Tauler tauler = new Tauler(1, 1, 0); // Constructor que inicializa un tablero vacío
    VistaTauler vista = new VistaTauler();


    tauler = new Tauler(1, 1, 0);
    vista.printTauler(tauler);


    tauler = new Tauler(2, 1, 0);
    vista.printTauler(tauler);


    tauler = new Tauler(1, 2, 0);
    vista.printTauler(tauler);


    tauler = new Tauler(2, 3, 0);
    vista.printTauler(tauler);


    tauler = new Tauler(5, 5, 0);
    vista.printTauler(tauler);


    tauler = new Tauler(10, 10, 0);
    vista.printTauler(tauler);


    tauler = new Tauler(15, 15, 0);
    vista.printTauler(tauler);

    tauler = new Tauler(1, 1, 1);
    vista.printTauler(tauler);

    tauler = new Tauler(2, 2, 0);
    vista.printTauler(tauler);

    tauler = new Tauler(2, 2, 1);
    vista.printTauler(tauler);

    tauler = new Tauler(3, 3, 5);
```

Exemple del loop testing anidat.

```java
public boolean validarPosicio(Tauler tauler, int x, int y) {
    if((x < 0 || x >= tauler.getAmplada()) || (y < 0 || y >= tauler.getAltura())) {
```

Decision i condition coverage.

```java
if(x+1 < 10) {
```

Segon condition i decision coverage.

Comentaris addicionals de la classe VistaTauler:
Tenim un 100% de coverage de la classe.

Hem afegit dos decision coverage i 2 condition coverage i un exemple de loop testing anidat. A més de fer servir un mock manual.