

# **Informe pràctiques**

Silvio Félix Alva Pérez - 1674456  
Daniel Rosa Díaz - 1604158

# Índex

<b>Índex.....</b>	<b>2</b>
<b>Introducció.....</b>	<b>3</b>
<b>Model vista controlador.....</b>	<b>3</b>
<b>Tests.....</b>	<b>4</b>
GameStructure.....	4
BoardPairwiseTest.....	4
BoardTest.....	6
PieceFactoryTest.....	8
PieceTest.....	10
GameControl.....	12
GameTestV1.....	12
GameTestV2.....	14
Diferencias entre GameTestV1 y GameTestV2.....	16
InputsTest.....	18
Statement Coverage.....	19
Loop testing.....	20
<b>CI/CD.....</b>	<b>22</b>
<b>Link al repositori del projecte.....</b>	<b>22</b>

## Introducció

A aquesta pràctica hem desenvolupat el joc Tetris, creat per Alexéi Pázhitnov en 1984. Està desenvolupat aplicant TDD a Java.

## Model vista controlador

En aquesta pràctica hem aplicat el model vista-controlador per organitzar les diferents parts del joc:

**Model:** La part del model, a la qual hem anomenat *GameStructure*, s'encarrega de definir els diferents elements del joc, que inclouen les classes:

- **Board:** defineix el tauler de joc i s'encarrega d'actualitzar-lo.
- **Piece:** s'encarrega de definir les característiques de cada peça, així com de donar-li la capacitat de moure's en les direccions permeses i de girar-la.
- **Piece Factory:** s'encarrega de crear les diferents peces del joc i de generar-les de manera aleatòria.

**Vista:** La part de la vista l'hem anomenada *GraficInterface*. Aquesta s'encarrega de crear la part visual del joc que l'usuari veu. Inclou les classes:

- **GamePanel:** s'encarrega de dibuixar els elements creats en el model: les peces i el tauler.
- **GameWindow:** s'encarrega de crear la finestra que s'obrirà en iniciar el joc.

**Controlador:** Aquesta part l'hem anomenada *GameControl* i s'encarrega de la jugabilitat (*gameplay*) del joc. Inclou les classes:

- **Game:** s'encarrega del funcionament general del joc: iniciar-lo, generar el tauler i les peces, així com les accions perquè l'usuari mogui les peces.
- **Input:** s'encarrega d'assignar els botons que l'usuari prem al teclat als moviments de la peça que descendeix.

# Tests

## GameStructure

### BoardPairwiseTest

Cas	Parametres(x, y, colIX, colY)	Resultat esperat	Explicació de l'escenari
1	(-1, 5, -1, -1)	FALSE	Fora de Límits (X Esquerra): La posició $x=-1$ col·loca la vora esquerra de la peça (en $x=-1$ ) fora del tauler violant el límit esquerre. És una prova de límit estricte. (Posició X: Esquerra, Posició I: Centre, Col·lisió: No).
2	(4, 19, -1, -1)	FALSE	Fora de Límits (Y A baix): La peça $2 \times 2$ està en (4,19). La seva coordenada i més baixa és $19+1=20$ , que està fora del límit inferior del tauler (assumint que el límit inferior és $Y_{max}=19$ ). (Posició X: Centre, Posició Y: A baix/Límit, Col·lisió: No).
3	(9, 10, 9, 10)	FALSE	Col·lisió + Fora de Límits (X Dreta): La posició $x=9$ col·loca la vora dreta de la peça (en $x=9+1=10$ ) fora del tauler (assumint que el límit dret és $X_{max}=9$ ). A mes hi ha una cel·la ocupada. La prova verifica que qualsevol de les condicions d'invalidesa causi la fallada. (Posició X: Dreta/Límit, Posició Y: Centre, Col·lisió: Sí).
4	(0, 0, -1, -1)	TRUE	Posició Vàlida (Cantonada Superior Esquerra): La prova més fonamental per a una posició vàlida. Verifica que els límits superior i esquerre funcionin correctament quan la peça està pegada a ells. (Posició X: Esquerra/Límit, Posició Y: A dalt/Límit, Col·lisió: No).
5	(4, 18, 5, 18)	FALSE	Col·lisió (Prop del Límit Inferior): La peça està en (4,18). Es col·loca una col·lisió en (5,18) que és una de les cel·les que ocuparia la peça. Es verifica la lògica de col·lisió just abans del límit inferior. (Posició X: Centre, Posició Y: A baix, Col·lisió: Sí).
6	(8, 0, -1, -1)	TRUE	Posició Vàlida (Cantonada Superior Dreta): La posició $x=8$ col·loca la peça en (8,0) i (9,0), que és el límit dret (assumint $X_{max}=9$ ). Verifica els límits superior i dret. (Posició X: Dreta/Límit, Posició Y: A dalt/Límit, Col·lisió: No).
7	(9, 5, -1, -1)	FALSE	Fora de Límits (X Dreta): Similar al cas 3 però sense col·lisió. La posició $x=9$ fa que la peça ocipi fins a $x=10$ , que està fora del límit dret del tauler. Aïlla la prova del límit dret. (Posició X: Dreta/Límit, Posició Y: Centre, Col·lisió: No).
8	(0, 19, -1, -1)	FALSE	Fora de Límits (Y A baix): Similar al cas 2 però en la vora esquerra. La posició $i=19$ fa que la peça ocipi fins a $i=20$ , fora del límit inferior. Aïlla la prova del límit inferior. (Posició X: Esquerra/Límit, Posició I: A baix/Límit, Col·lisió: No).
9	(4, 0, 4, 1)	FALSE	Col·lisió (Prop del Límit Superior): La peça està en (4,0). Es col·loca una col·lisió en (4,1) que és una de les cel·les que ocuparia la peça. Verifica la lògica de col·lisió just després del límit superior. (Posició X: Centre, Posició Y: A dalt/Límit, Col·lisió: Sí).

Silvio Félix Alva Pérez - 1674456

Daniel Rosa Díaz - 1604158

En triar punts en els límits (Esquerra, Dreta, A dalt, A baix) i combinar-los amb l'estat de Col·lisió (Sí o No), s'assegura que el mètode `isValidPosition` es provi contra:

- Posicions que toquen els límits vàlids.
- Posicions que excedeixen els límits invàlids.
- Posicions que generen col·lisions vàlides (peces ocupades).
- Posicions que són completament vàlides (el cas true).

## BoardTest

### Tests per a la Inicialització del Tauler (testBoardInitialization)

- Aquesta prova fonamental assegura que l'estructura de dades que representa el tauler (grid) es crea correctament.
- `assertNotNull(grid):` Verifica que el tauler (la matriu grid) no sigui null, garantint que es va inicialitzar.
- `assertEquals(Board.BOARD_HEIGHT, grid.length)` i `assertEquals(Board.BOARD_WIDTH, grid[0].length):` Confirma que el tauler té les dimensions verticals (alt) i horizontals (ample) correctament definides per les constants de la classe Board.
- Loop de verificació (`assertNull(grid[i][j]):`) Recorre cada cel·la de la matriu i assegura que totes les cel·les es van inicialitzar a null. Això és crucial ja que null representa una cel·la buida, llista per a rebre una peça.

### Tests para isValidPosition

- A. Proves Data-Driven (testIsValidPosition\_DataDriven)
- Utilitzant `@parameterizedtest` amb `@csvsource`, es prova el mètode amb diferents combinacions de coordenades per a cobrir els límits del tauler. La peça utilitzada és la shapel (una línia horizontal de 4 blocs).

Cas (x, y)	Esperat	Propòsit
0, 0	TRUE	Vàlid: Cantonada superior esquerra. La prova basi per a una posició permesa.
-1, 0	FALSE	Límit Esquerre: La peça comença en x=-1, fora del tauler.
7, 0	FALSE	Límit Dret: La peça de 4 d'ample comença en x=7 i ocupa fins a x=10 (assumint Xmax=9). Verifica la comprovació del límit dret.
0, 20	FALSE	Límit Inferior: La peça d'1 d'alt comença en y=20 (assumint Ymax=19). Verifica la comprovació del límit inferior.
0, -1	TRUE	Límit Superior (Parcialment Fos): Una peça pot començar amb parts per sobre del tauler (y<0) sempre que no col·lideixi i no estigui fora dels límits x. Això és típic de com apareixen les peces noves.

- B. Prova de Col·lisió (testIsValidPosition\_Collision)
- `board.getGrid()[1][1] = Color.XARXA;`: Es col·loca un bloc de color en una cel·la per a simular una peça ja existent.
- `assertFalse(board.isValidPosition(singlePoint, new Point(1, 1)))`: S'intenta col·locar una peça d'un sol punt en la mateixa posició. La prova verifica que el mètode detecti correctament la col·lisió i retorni false.

### 3. Tests per a placePiece

- Aquestes proves verifiquen que el mètode `placePiece(shape, origin, color)` col·loqui els blocs de la peça en les coordenades correctes del tauler.
- `testPlacePiece (Col·locació normal):`

- Col·loca la peça shapel en (3, 5) amb color CYAN.
- Verifica que les quatre cel·les (5, 3), (5, 4), (5, 5), (5, 6) ara contenen el color CYAN.
- `testPlacePiece_PartiallyAboveBoard` (Col·locació parcial):
  - Utilitza una peça que té una part fora del tauler (en y=-1).
  - `assertDoesNotThrow`: Assegura que el mètode no falla en intentar escriure en una posició fora del array (la cel·la y=-1).
  - Verifica que la part visible de la peça (en y=0) sí que es col·loca correctament.

#### Tests per a `clearLines`

- Aquestes proves són les més complexes i verifiquen que el tauler identifiqui, elimini, i desplaci correctament els blocs després de completar una o més línies.
- `testClearLines_NoLines`: Verifica el cas basi: si només hi ha un bloc solt, no s'elimina cap línia i el mètode retorna 0.
- `testClearLines_OneLine`:
  - Omple l'última fila (la més baixa).
  - Col·loca un bloc de prova en la fila anterior.
  - Verifica que es retorna 1 (una línia eliminada).
  - Crucialment, verifica que la fila eliminada ara estigui null i que el bloc de prova hagi caigut una posició, demostrant el mecanisme de gravetat/desplaçament de files.
- `testClearLines_MultipleLines`:
  - Omple les dues últimes files.
  - Col·loca un bloc de prova en la primera fila.
  - Verifica que es retorna 2.
  - Verifica que el bloc de prova hagi caigut dues posicions (demostrant el desplaçament correcte amb múltiples eliminacions).
- `testClearLines_FullBoard`: Prova el cas extrem d'eliminar totes les línies alhora. Verifica que el comptatge és correcte (\*Board.BOARD\_HEIGHT) i que tot el tauler està null en finalitzar.
- `testClearLines_LineInTheMiddle`:
  - Omple una fila en la meitat del tauler.
  - Col·loca un bloc de prova per sobre d'aquesta línia.
  - Verifica que es retorna 1 i que el bloc de prova cau només fins a la posició immediatament superior a la línia eliminada. Això assegura que el desplaçament de blocs sigui localitzat i només afecti les files superiors a la línia eliminada.
- `testClearLines_EmptyBoard`: Verifica que el mètode s'executi sense errors i retorna 0 quan no hi ha blocs, garantint que no intenta eliminar línies on no hi ha dades.

## PieceFactoryTest

La classe PieceFactoryTest utilitza Mockito (@extendwith(MockitoExtension.class)) per a simular dependències i enfocar el testing exclusivament en la lògica de la fàbrica de peces (PieceFactory).

### 1. Preparació de l'Entorn (Mocking)

- @mock private Board board;: S'utilitza un mock per a la classe Board. Encara que la PieceFactory probablement necessita una referència al Board (potser per a determinar la posició inicial), en aquestes proves no ens importa la funcionalitat del tauler, només la creació de la peça. Mockito assegura que no falli si la fàbrica intenta cridar a un mètode del tauler.

### 2. Control d'Aleatorietat (Predictable Random)

- La funció d'una fàbrica de peces de Tetris és generar una peça aleatòria. Per a provar això de manera determinista, s'usa una tècnica clau: injectar un objecte Random personalitzat (o stub).

#### A. testGetNewPiece\_\*CreatesIShape

- Aquesta prova verifica que la fàbrica pugui generar la peça I i que els seus atributs (color i forma) siguin correctes.

```
Random predictableRandom = new Random() {  
    @Override  
    public int nextInt(int bound) {  
        return 0; // Fuerza la pieza I  
    }  
};
```

Captura 1: Control del Random

Se sobreescrivia el mètode nextInt(bound) perquè sempre retorna 0, forçant a la PieceFactory a seleccionar la peça en l'índex 0 de la seva llista interna.

Verificació:

- assertEquals(Color.CYAN, newPiece.getColor()): Confirma que la peça I s'associa al color CIAN.
- assertArrayEquals(expectedShape, newPiece.getShape()): Confirma que la matriu de punts que defineix la forma de la peça I és correcta: { (0, 1), (1, 1), (2, 1), (3, 1) }.

#### B. testGetNewPiece\_CreatesOShape

- Aquesta prova és anàloga a l'anterior, però verifica la peça O (el quadrat).
- Control del Random: Es força a retornar 3 (assumint que aquest índex correspon a la peça O).
- Verificació:
  - assertEquals(Color.YELLOW, newPiece.getColor()): Confirma el color GROC.

Silvio Félix Alva Pérez - 1674456

Daniel Rosa Díaz - 1604158

- `assertArrayEquals(expectedShape, newPiece.getShape())`: Confirma la forma quadrada { (0, 0), (1, 0), (0, 1), (1, 1) }.

### 3. Prova de Creació Genèrica

#### C. `testGetNewPiece_NotNull`

- Aquesta prova verifica que, sota condicions d'aleatorietat normal, la fàbrica sempre genera una peça vàlida, sense centrar-se en una forma específica.
- Ús de Random real: S'usa el constructor `new PieceFactory(board)` (o el que usi el Random per defecte), permetent que l'elecció de la peça sigui aleatòria.
- Verificació:
  - `assertNotNull(newPiece)`: Es verifica que la fàbrica no retorna null.
  - `assertNotNull(newPiece.getColor())` i `assertNotNull(newPiece.getShape())`: S'assegura que la peça generada tingui tant un color com una forma definida, independentment de la peça triada.

## PieceTest

La classe PieceTest utilitza Mockito (@extendwith(MockitoExtension.class)) per a simular el comportament del tauler (Board), la qual cosa permet provar la lògica interna de la Piece (moviment i rotació) sense preocupar-se per la implementació real de les comprovacions de col·lisió o límits del tauler.

1. Preparació i Inicialització (setUp i testPieceInitialization)
  - @mock private Board board;: El tauler se simula (mockea) perquè el mètode Piece.moveX() depèn de board.isValidPosition(). Al mockearlo, podem controlar exactament el que retorna isValidPosition().
  - Peça de prova: S'usa la forma de la peça L (shapeL) per a les proves: { (0, 0), (0, 1), (0, 2), (1, 2) }.
  - testPieceInitialization:
    - Verifica que la forma (shapeL) i el color (ORANGE) s'assignin correctament.
    - Posició Inicial: És crucial verificar que la peça es col·loca en la posició d'inici estàndard, que sol ser centrada horitzontalment: new Point(Board.BOARD\_WIDTH / 2 - 1, 0).
2. Tests de Moviment (testPieceMovement\_DataDriven)
  - Aquestes proves utilitzen @ParameterizedTest con @CsvSource per a cobrir de manera eficient els diferents casos de moviment (Esquerra, Dreta, Avall) i els dos resultats possibles de la validació del tauler (moviment vàlid i invàlid).
  - El comportament clau controlat pel mock es: when(board.isValidPosition(any(), any(Point.class))).thenReturn(isValid);.

Direcció	isValid	expectedDx	expectedDy	Propòsit
DOWN	TRUE	0	1	Moviment Vàlid: Si el tauler ho permet, la coordenada I ha d'incrementar-se en 1.
DOWN	FALSE	0	0	Moviment Invàlid: Si el tauler no ho permet, la posició no ha de canviar.
LEFT	TRUE	-1	0	Moviment Vàlid: Si el tauler ho permet, la coordenada X deu *decrementarse en 1.
LEFT	FALSE	0	0	Moviment Invàlid: Si el tauler no ho permet, la posició no ha de canviar.
RIGHT	TRUE	1	0	Moviment Vàlid: Si el tauler ho permet, la coordenada X ha d'incrementar-se en 1.
RIGHT	FALSE	0	0	Moviment Invàlid: Si el tauler no ho permet, la posició no ha de canviar.

3. Tests de Rotació
  - La rotació és una operació complexa que canvia els punts interns de la forma. Aquestes proves garanteixen que la rotació només s'execute si és vàlida i que si s'executa, la forma interna canvi.

A. Rotació Vàlida (testRotate\_Valid)

- Setup: Es força al mock a retornar true: when(board.isValidPosition(any(), any(Point.class))).thenReturn(true);.
- Verificació:
  - El codi copia la forma abans de girar.
  - Crida a piece.rotate().
  - Utilitza un bucle per a verificar que almenys un punt en la forma actual és diferent de la forma original (\*assertTrue(changed)). Això demostra que la transformació geomètrica va ocórrer correctament.

B. Rotació Invàlida (testRotate\_Invalid)

- Setup: Es força al mock a retornar false: when(board.isValidPosition(any(), any(Point.class))).thenReturn(false);.
- Verificació:
  - Es guarda la forma inicial (initialShape).
  - Crida a piece.rotate().
  - assertEquals(initialShape, piece.getShape()): Confirma que la forma interna no va canviar perquè el tauler va indicar que la nova posició girada no era vàlida.

4. Tests per a canMoveDown

- Aquesta prova aïlla la lògica de verificació de moviment, que és un pas previ a l'execució real de moveDown().
- testCanMoveDown\_True:
  - Setup: El mock es configura per a retornar true específicament per a la posició un pas a baix ( $i + 1$ ).
  - Verificació: assertTrue(piece.canMoveDown()).
- testCanMoveDown\_False:
  - Setup: El mock es configura per a retornar \*false per a la posició un pas a baix.
  - Verificació: \*assertFalse(piece.canMoveDown()).

## GameControl

### GameTestV1

1. Preparació de l'Entorn (Mocks i setUp)
  - Mocks: S'utilitzen mocks de Board i PieceFactory per a simular el seu comportament, i un mock de Piece (currentPiece) per a simular la peça activa.
  - @beforeeach: Defineix l'estat inicial de cada prova:
    - when(pieceFactory.getNewPiece()).thenReturn(currentPiece): Assegura que cada vegada que la fàbrica és consultada retorna el mock currentPiece.
    - when(board.isValidPosition(any(), any())).thenReturn(true): Assumeix que la posició inicial de qualsevol peça és vàlida (evitant la fi del joc immediat).
    - game = new Game(board, pieceFactory): Inicialitza el joc la qual cosa desencadena la primera anomenada a pieceFactory.getNewPiece().
2. Inicialització del Joc
  - testConstructorGameNoParameters:
    - Prova el constructor sense argument que ha de crear instàncies reals de Board i PieceFactory internament.
    - Verifica que tant el tauler com la peça actual no siguen null.
  - testGameInitialization:
    - Verifica l'estat del joc després de la configuració en setUp.
    - assertEquals(currentPiece, game.getCurrentPiece()): Confirma que la peça activa és la que va retornar el mock.
    - verify(pieceFactory, estafes(1)).getNewPiece(): Assegura que el constructor va cridar a la fàbrica una vegada per a obtindre la primera peça.
    - assertFalse(game.isGameOver()): Confirma que el joc no va acabar a l'inici.
3. Tests de Moviment Bàsic (Delegació)
  - Aquestes proves són senzilles i verifiquen que l'objecte Game simplement delega l'acció de moviment o rotació a la peça activa.
  - testMoveLeft, testMoveRight, testRotatePiece:
    - El patró és idèntic: es crida al mètode del game (game.moveLeft(), game.rotatePiece(), etc.).
    - Es verifica (verify) que el mètode corresponent en el mock currentPiece (currentPiece.moveLeft(), currentPiece.rotate(), etc.) haja sigut anomenat exactament una vegada (estafes(1)).
    - No es requereix stubbing (configurar el mock) perquè només estem verificant que la crida es realitze.
4. Tests de moveDown (Lògica de Fi de Torn)
  - Aquesta és la secció més important, ja que moveDown() implementa la lògica principal del joc: Moure la peça? o Bloquejar la peça i començar un nou torn? .

A. testMoveDown\_CanMove

- Condició: La peça pot moure's cap avall.
- Setup: when(currentPiece.canMoveDown()).thenReturn(true);.
- Verificació:
  - verify(currentPiece, estafes(1)).moveDown(): Es verifica que la peça es va moure.
  - verify(board, never()).placePiece(...): Es verifica que no es va dir a la col·locació de la peça en el tauler, ja que el torn encara no ha acabat.

B. testMoveDown\_CannotMove

- Condició: La peça no pot moure's (ha tocat fons o col·lidit).
- Setup: when(currentPiece.canMoveDown()).thenReturn(false);.
- Lògica de Bloqueig/Nou Torn:
  - Col·locar Peça: El joc ha de col·locar la peça actual en el tauler. verify(board, estafes(1)).placePiece(any(), any(), any()).
  - Netejar Línies: El joc ha d'intentar netejar línies. verify(board, estafes(1)).clearLines().
  - Nova Peça: El joc ha de demanar una nova peça a la fàbrica. verify(pieceFactory, estafes(2)).getNewPiece() (Una crida en setUp + una crida en moveDown).
  - Actualitzar Estat: El joc ha d'establir la nova peça com currentPiece. assertEquals(newPiece, game.getCurrentPiece()).

5. Test de Fi de Joc (testGameOverInitialization)

- Aquesta prova verifica que el joc detecte immediatament una condició de Fi de Joc en intentar generar la primera peça.
- Setup:
  - Se simula que la peça inicial té una forma i posició.
  - when(board.isValidPosition(any(), any())).thenReturn(false): Es força al Board a indicar que la posició inicial de la nova peça és invàlida (p. ex., perquè la zona de spawn ja està ocupada).
- Verificació:
  - Es crea un nou objecte Game amb aquesta configuració.
  - assertTrue(gameOverGame.isGameOver()): Confirma que el joc es marca com acabat immediatament.

## GameTestV2

### 1. Configuració i Aïllament (Mocks)

L'ús de mocks permet que les proves de GameTestV2 se centren únicament en la lògica de Game sense dependre de la correcta implementació de Board o Piece.

- @mock private Board board;
- @mock private PieceFactory pieceFactory;
- @mock private Piece currentPiece;

Aquests mocks simulen les dependències i permeten al desenvolupador controlar exactament el que retornen (ex. si una posició és vàlida) i verificar exactament quins mètodes van ser anomenats (ex. si la peça es va moure).

El mètode setupGameForMethodTests() és un helper que inicialitza el joc amb condicions bàsiques exitoses per a la majoria dels tests d'acció promovent el principi de "Arrange-Act-Assert" net.

### 2. Proves d'Inicialització (testGameInitialization)

Aquestes proves validen l'estat inicial del joc, prestant especial atenció a la condició de Fi de Joc.

- testConstructorGameNoParameters: Verifica que el constructor per defecte cree instàncies reals (no mocks) de Board i Piece, assegurant que el joc puga funcionar sense injecció de dependències.
- testGameInitialization\_Success: Simula que la peça recentment creada té una posició vàlida. Verifica que:
  - Es crida a la fàbrica (verify(pieceFactory).getNewPiece()).
  - El tauler valguda la posició inicial (verify(board).isValidPosition(...)).
  - El joc no està acabat (assertFalse(game.isGameOver())).
- testGameInitialization\_GameOver: Simula que la peça inicial té una posició invàlida (when(board.isValidPosition(...)).thenReturn(false)). Això ocorre si el tauler està ple a l'inici. Verifica que:
  - El joc sí que està acabat (assertTrue(game.isGameOver())).

### 3. Proves d'Acció Condicional (Maneig de GameOver)

Aquesta és una de les principals millores de V2. Totes les proves de moviment i rotació ara cobreixen dos casos: quan el joc està actiu i quan el joc ha acabat. Això assegura la robustesa del controlador.

Mètode d'Acció	Cas d'Èxit (WhenNotGameOver)	Cas de Fallada (WhenGameOver)
moveLeft()	Crida a verify(currentPiece).moveLeft()	Crida a verify(currentPiece, never()).moveLeft()
moveRight()	Crida a verify(currentPiece).moveRight()	Crida a verify(currentPiece, never()).moveRight()
rotatePiece()	Crida a verify(currentPiece).rotate()	Crida a verify(currentPiece, never()).rotate()

La prova garanteix que, si game.isGameOver() és veritable, el Game actua com un guardià i ignora les entrades de l'usuari.

#### 4. Proves de Lògica de Torn (testMoveDown)

El mètode moveDown() encapsula la lògica central del joc (moure, bloquejar, spawnear, netejar línies).

##### A. testMoveDown\_WhenCanMoveDown

- Lògica: La peça pot continuar caient.
- Setup: when(currentPiece.canMoveDown()).thenReturn(true).
- Verificació:
  - Es crida a verify(currentPiece).moveDown().
  - No es crida a verify(board, never()).placePiece(...), ja que el torn encara no acaba.

##### B. testMoveDown\_WhenCannotMoveDown

- Lògica: La peça ha tocat fons i ha de bloquejar-se, iniciant un nou torn.
- Setup Avançat: S'utilitza clearInvocations(pieceFactory, board) per a ignorar les crides fetes pel constructor. Es configura el mock per a simular:
  - currentPiece.canMoveDown() retorna fals.
  - pieceFactory.getNewPiece() retorna una nova peça mockeada (newPiece).
- Verificació (Seqüència de Bloqueig):
  - verify(board).placePiece(...): La peça actual es col·loca en el tauler.
  - verify(board).clearLines(): Es verifica l'intent de netejar línies.
  - verify(pieceFactory.getNewPiece()): Es genera la següent peça.
  - assertEquals(newPiece, game.getCurrentPiece()): La nova peça es converteix en la peça activa.

##### C. testMoveDown\_WhenGameOver

- Lògica: Verifica que la lògica de fi de torn no s'execute si el joc ja va acabar.
- Verificació: Assegura que cap mètode important (canMoveDown, moveDown, placePiece) és anomenat.

## Diferencias entre GameTestV1 y GameTestV2

### 1. Control Explícit del Constructor i Setup

Característica	GameTestV1	GameTestV2	Mejora en V2
Setup General	Utilitza <code>@beforeeach</code> per a inicialitzar el game. Això introduceix dependències entre tests, ja que el mocking del constructor ocorre abans de cada <code>@test</code> .	Introduceix el mètode <code>setupGameForMethodTests()</code> privat. Això permet a les proves d'inicialització ( <code>testGameInitialization_Success/GameOver</code> ) controlar el seu propi setup, mentre que les proves de mètodes usen un setup base net.	Major Aïllament: Les proves d'inicialització tenen control total sobre la lògica de mocking (incloent el <code>isValidPosition</code> que determina la Fi de Joc) sense interferir amb altres proves.
Helper para Pieza	Utilitza <code>game.setCurrentPieceForTest(currentPiece)</code> en diversos tests per a assegurar-se que la peça que està en el mock siga l'activa.	Manté el mateix enfocament, però s'usa de forma més neta en aïllar el setup base en <code>setupGameForMethodTests()</code> .	Manté la utilitat de configurar la peça actual sense passar per la lògica del constructor.

### 2. Proves d'Inicialització i Fi de Joc

V2 divideix i millora les proves d'inicialització per a cobrir la Fi de Joc com un resultat directe del constructor.

- `testGameInitialization` (V1) vs. `testGameInitialization_Success` (V2):
  - Tots dos verifiquen la inicialització exitosa. V2 és més explícit en el `Arrange` (configuració de mocks) abans del `Act` (creació del joc).
- `testGameOverInitialization` (V1) vs. `testGameInitialization_GameOver` (V2):
  - V1 requereix `reset(board, pieceFactory)` per a netejar els mocks del `@beforeeach` i després configura la condició de fi de joc (`when(board.isValidPosition(any(), any())).thenReturn(false)`).
  - V2 és més net: simplement configura els mocks (`when(board.isValidPosition(...)).thenReturn(false)`) i després crida al constructor. L'ús de `setupGameForMethodTests()` per a la resta de mètodes evita la necessitat de cridar a `reset()` ací.

### 3. Cobertura de la condició de isGameOver

La millora més significativa és la introducció de proves per a la condició de Fi de Joc en tots els mètodes de control (moveLeft, moveRight, rotatePiece, moveDown).

Proves Afegides en V2	Propòsit
testMoveLeft_WhenGameOver	Assegura que currentPiece.moveLeft() mai es diu si el joc està acabat.
testMoveRight_WhenGameOver	Assegura que currentPiece.moveRight() mai es diu si el joc està acabat.
testRotatePiece_WhenGameOver	Assegura que currentPiece.rotate() mai es diu si el joc està acabat.
testMoveDown_WhenGameOver	Assegura que la lògica de moviment i fi de torn no s'executa si el joc està acabat.

### 4. Neteja i Rigor en testMoveDown\_WhenCannotMoveDown

El test en V2 per al final del torn (testMoveDown\_WhenCannotMoveDown) és molt més rigorós en el maneig dels mocks:

- Utilitza clearInvocations(pieceFactory, board) per a assegurar-se que les verificacions (verify) al final de la prova només compten les crides fetes dins del mètode moveDown(), aïllant la prova del soroll de les crides fetes durant la inicialització del constructor.
- És més explícit en la configuració de stubbing per a la nova peça que es genera, assegurant que pieceFactory.getNewPiece() retorna el mock newPiece la segona vegada que es diu.

## InputsTest

La classe *InputsTest* utilitza Mockito (@extendwith(MockitoExtension.class)) per simular el comportament de l'objecte *Game*, cosa que permet provar que la classe *Inputs* tradueix correctament les tecles pulsades en crides als mètodes corresponents del joc, sense necessitat d'executar la lògica real del Tetris ni preocupar-se pel moviment, la rotació o la gestió del tauler.

### 1. Preparació de l'entorn (setUp)

- game = mock(Game.class): Ens permet simular els moviments de les peces que es produeixen com a resultat de les tecles que es premen.

### 2. Tests de botó premut (testKeyPressed)

- Amb aquests test es crida a la funció KeyPressed per a cada tecla i comprova:
  - Si la tecla es vàlida, comprova que només s'executa la funció per moure la peça com està previst per aquesta tecla i res mès.
  - Si no es vàlida o no es prem cap tecla, es verifica que no fa res.

### 3. Tests de botó alliberat (testKeyReleased)

- Aquests tests s'encarreguen de comprobar si encara que la acció released no provocarà cap error inesperat y a la clase game no li arriba cap instrucció.

### 4. Test de tipus de tecla (testKeyType)

- Aquest test comproven que:
  - Si el tipus es null, indica que no es prem cap tecla, pel que el joc no fa res.
  - Si el tipus es vàlid, tampoc farà res degut a que el joc només es modifica amb els procediments keyPressed.

## Statement Coverage

Element	Class, %	Method, %	Line, %	Branch, %
cat.uab.tqs	62% (5/8)	76% (33/43)	66% (112/1...)	72% (57/79)
Main	0% (0/1)	0% (0/1)	0% (0/1)	100% (0/0)
GraficInterface	0% (0/2)	0% (0/9)	0% (0/56)	0% (0/16)
GameWindow	0% (0/1)	0% (0/4)	0% (0/20)	0% (0/2)
GamePanel	0% (0/1)	0% (0/5)	0% (0/36)	0% (0/14)
GameStructure	100% (3/3)	100% (17/17)	100% (73/7...)	100% (40/40)
PieceFactory	100% (1/1)	100% (3/3)	100% (12/1...)	100% (2/2)
Piece	100% (1/1)	100% (9/9)	100% (25/25)	100% (10/10)
Board	100% (1/1)	100% (5/5)	100% (36/36)	100% (28/28)
GameControl	100% (2/2)	100% (16/16)	100% (39/39)	73% (17/23)
Inputs	100% (1/1)	100% (4/4)	100% (9/9)	44% (4/9)
Game	100% (1/1)	100% (12/12)	100% (30/30)	92% (13/14)

Captura 2: Imatge on es mostra el percentatge de cobertura de cadascun dels mètodes del projecte.

## Path coverage y condition coverage

```
42 @    public boolean isValidPosition(Point[] pieceShape, Point position) { 24 usages  Danield1707
43 |        for (Point p : pieceShape) {
44 |            int x = position.x + p.x;
45 |            int y = position.y + p.y;
46 |
47 |                // Comprobar limites horizontales
48 |                if (x < 0 || x >= BOARD_WIDTH) {
49 |                    return false;
50 |                }
51 |
52 |                // Comprobar límite inferior
53 |                if (y >= BOARD_HEIGHT) {
54 |                    return false;
55 |                }
56 |
57 |                // Si la coordenada y es negativa, está por encima del tablero, lo cual es válido
58 |                // Solo comprobamos colisión si la celda está dentro del área visible del tablero
59 |                if (y >= 0) {
60 |                    if (grid[y][x] != null) {
61 |                        return false; // Celda ocupada
62 |                    }
63 |                }
64 |
65 |        }
66 |        return true;
67 }
```

Captura 3: Imatge on es mostra una funció amb condicions on es comproven totes les possibilitats

## Loop testing

```
42 @    public boolean isValidPosition(Point[] pieceShape, Point position) { 24 usages & Danird1707
43     for (Point p : pieceShape) {
44         int x = position.x + p.x;
45         int y = position.y + p.y;
46
47         // Comprobar límites horizontales
48         if (x < 0 || x >= BOARD_WIDTH) {
49             return false;
50         }
51
52         // Comprobar límite inferior
53         if (y >= BOARD_HEIGHT) {
54             return false;
55         }
56
57         // Si la coordenada y es negativa, está por encima del tablero, lo cual es válido
58         // Solo comprobamos colisión si la celda está dentro del área visible del tablero
59         if (y >= 0) {
60             if (grid[y][x] != null) {
61                 return false; // Celda ocupada
62             }
63         }
64     }
65     return true;
66 }
```

Captura 4: Loop simple en la función isValidPosition de la clase Board.

```
1 /**
2  * Coloca una pieza en el tablero.
3  * @param pieceShape La forma de la pieza.
4  * @param position La posición de la pieza.
5  * @param color El color de la pieza.
6  */
7 @    public void placePiece(Point[] pieceShape, Point position, Color color) { 8 usages & Danird1707
8     for (Point p : pieceShape) {
9         int x = position.x + p.x;
10        int y = position.y + p.y;
11        if (y >= 0) {
12            grid[y][x] = color;
13        }
14    }
15 }
```

Captura 5: Loop simple en la función placePiece de la clase Board.

```
.5  /**
.6   * Constructor de la clase Board.
.7   * Inicializa la rejilla del tablero a null (vacío).
.8   */
.9  public Board() { &Danird1707
10     grid = new Color[BOARD_HEIGHT][BOARD_WIDTH];
11     for (int i = 0; i < BOARD_HEIGHT; i++) {
12         for (int j = 0; j < BOARD_WIDTH; j++) {
13             grid[i][j] = null; // null representa una celda vacía
14         }
15     }
16 }
```

Captura 6: Loop aniat al constructor sense paràmetres de la classe Board.

```
73  public int clearLines() { 9 usages & Danird1707
74      int linesCleared = 0;
75      Color[][] newGrid = new Color[BOARD_HEIGHT][BOARD_WIDTH];
76      int newRow = BOARD_HEIGHT - 1;
77
78      for (int i = BOARD_HEIGHT - 1; i >= 0; i--) {
79          boolean lineIsFull = true;
80          for (int j = 0; j < BOARD_WIDTH; j++) {
81              if (grid[i][j] == null) {
82                  lineIsFull = false;
83                  break;
84              }
85          }
86
87          if (!lineIsFull) {
88              // Si la línea no está llena, la copiamos a la nueva rejilla
89
90              System.arraycopy(grid[i], 0, newGrid[newRow], 0, BOARD_WIDTH);
91              newRow--;
92
93          } else {
94              linesCleared++;
95          }
96      }
97
98      // Reemplazamos la rejilla antigua con la nueva, que ya tiene las líneas vacías arriba
99      System.arraycopy(newGrid, 0, grid, 0, BOARD_HEIGHT);
100
101      return linesCleared;
102 }
```

Captura 7: Loop aniat a la funció clearLines de la classe Board.

Silvio Félix Alva Pérez - 1674456

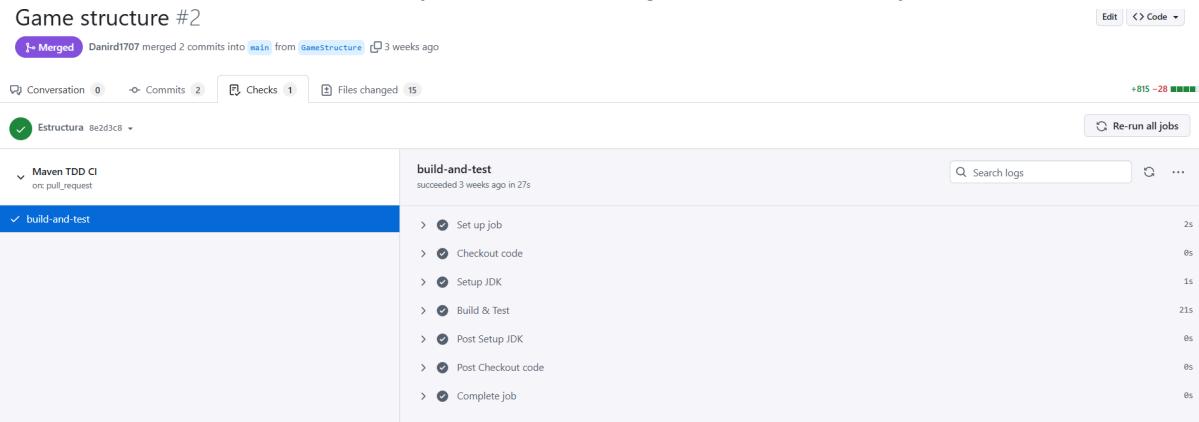
Daniel Rosa Díaz - 1604158

## CI/CD

En aquest projecte hem implementat CI/CD per a que al fer merge de la rama del repositori on es troba cada part del projecte amb la branca pmain no permeti realitzar aquesta acció si no supera tots els test.

Això ho hem fet amb el fitcher ci.yaml ubicado en “.github/workflows/ci.yaml”.

Game structure #2



Captura 8: Imatge de GitHub que indica que el merge aplicant CI s'ha efectuat correctament.

## Link al repositori del projecte

<https://github.com/Danird1707/Practica-Test-i-Qualitat-del-Software>