Robert Pearson Ruiz 1630916
Sergio López Parejo 1634093
Dilluns 10:30-12:30
Tetris

Funcionalitat: Marcar una casella del tauler com a ocupada.

Localització: Arxiu Board.java, classe Board, funció occupyCell(final int x, final int y).

Test1: Arxiu BoardTest.java, classe BoardTest, funció testCellBoundary(). Tipus caixa negra. Utilitza particions equivalents, casos límit i statement coverage.

Test2: Arxiu BoardTest.java, classe BoardTest, funció testOccupyCellWithinBoard() Tipus caixa blanca. Utilitza particions equivalents, statement coverage, decision coverage i condition coverage.

El test1 comprova que s'ocupin i detectin com a ocupades correctament les caselles en els límits del tauler.

El test2 comprova que únicament s'ocupin i detectin com a ocupades les caselles dins els límits del tauler.

```
    /**
    * Marca una celda como ocupada en las coordenadas especificadas.
    * @param x La coordenada x de la celda a ocupar.
    * @param y La coordenada y de la celda a ocupar.*/
    public void occupyCell(final int x, final int y) {
        if (y >= 0 && y < height && x >= 0 && x < width) {
            grid[y][x] = true;
            }
        }
```

Funcionalitat: Saber si una casella està ocupada o no.

Localització: Arxiu Board.java, classe Board, funció isCellOccupied (final int x, final int y).

Test1: Arxiu BoardTest.java, classe BoardTest, funció testIsCellOccupiedAfterOccupyCell(). Tipus caixa blanca. Utilitza particions equivalents, casos límit, statement coverage, decision coverage i condition coverage.

Test2: Arxiu BoardTest.java, classe BoardTest, funció testIsCellOccupiedInitialFalse(). Tipus caixa negra. Utilitza particions equivalents i statement coverage.

El test1 comprova que en ocupar una casella, efectivament es detecta com a ocupada, i la robustesa del mètode davant a diferents entrades.

El test2 comprova que abans de ocupar una casella es detecti com a no ocupada.

```
    /**
     * Verifica si una celda está ocupada.
     * @param x Coordenada x de la celda.
     * @param y Coordenada y de la celda.
     * @return true si la celda está ocupada; false de lo contrario.
     */
    public boolean isCellOccupied(final int x, final int y) {
        if (y >= 0 && y < height && x >= 0 && x < width) {
            return grid[y][x];
        }
        return false;
    }
```

Funcionalitat: Comprovar si una peça col·lisiona amb una altra o amb el tauler.

Localització: Arxiu Board.java, classe Board, funció checkCollision(final Piece piece).

Test: Arxiu BoardTest.java, classe BoardTest, funció testCheckCollisionConditionCoverage(). Tipus caixa blanca. Casos límit, statement coverage, decision coverage, condition coverage i mock objects implementats a mà.

```
    public boolean checkCollision(final Piece piece) {
        boolean[][] shape = piece.getShape();
        int pieceX = piece.getX();
        int pieceY = piece.getY();

        for (int row = 0; row < shape.length; row++) {
            for (int col = 0; col < shape[row].length; col++) {
                if (shape[row][col]) {
                    int boardX = pieceX + col;
                    int boardY = pieceY + row;

                    if (boardX < 0 || boardX >= width
                        || boardY < 0 || boardY >= height) {
                        return true;
                    }
                    if (grid[boardY][boardX]) {
                        return true;
                    }
                }
            }
        }
        return false;
    }
```

Funcionalitat: Fixar una peça a una posició del tauler.

Localització: Arxiu Board.java, classe Board, funció lockPiece(final Piece piece).

Test1: Arxiu BoardTest.java, classe BoardTest, funció testLockPiece(). Tipus caixa blanca. Utilitza particions equivalents i statement coverage.

Test2: Arxiu BoardTest.java, classe BoardTest, testCheckCollisionWithLockedPieces(). Tipus caixa blanca. Utilitza particions equivalents, statement coverage i mock objects implementats a mà.

El test1 comprova que en bloquejar una peça, s'ocupen les caselles corresponents.

El test2 comprova que es pugui detectar una col·lisió amb una peça que hagi estat bloquejada.

Funcionalitat: Eliminar les fileres totalment emplenades.

Localització: Arxiu Board.java, classe Board, funció clearFullLines().

Test: Arxiu BoardTest.java, classe BoardTest, funció testClearFullLines(). Tipus caixa blanca. Utilitza particions equivalents i statement coverage.

Funcionalitat: Comprovar si una fila està totalment emplenada.

Localització: Arxiu Board.java, classe Board, funció isFull(final boolean[] row).

Test: Arxiu BoardTest.java, classe BoardTest, funció testIsFull(). Tipus caixa blanca. Utilitza statement coverage, desision coverage i path coverage.

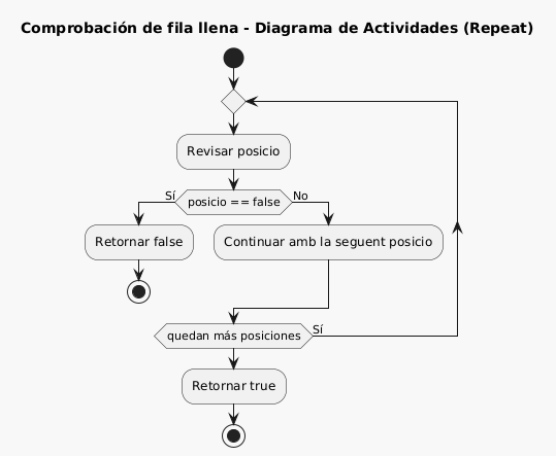

Diagrama de flux de IsFull()

```java
/**
* @param row La línea que se comprueba.
* @return Devuelve si una línea está completamente llena.
*/
public boolean isFull(final boolean[] row) {
    for (boolean cell : row) {
        if (!cell) {
            return false;
        }
    }
    return true;
}
```

Funcionalitat: Baixar les files cap avall quan s'elimina una fila.

Localització: Arxiu Board.java, classe Board, funció prependRow(final boolean[][] newGrid, final boolean[] newRow).

Test: Arxiu BoardTest.java, classe BoardTest, funció testPrependRow(). Tipus caixa negra. Utilitza particions equivalents, comprova casos límit i statement coverage.

Funcionalitat: Transformar el tauler a un string representable.

Localització: Arxiu Board.java, classe Board, funció renderWithPiece(final Piece piece).

Test1: Arxiu BoardTest.java, classe BoardTest, funció testRenderWithPiece(). Tipus caixa negra. Utilitza particions equivalents i statement coverage.

Test2: Arxiu BoardTest.java, classe BoardTest, funció testRenderWithPiece(). Tipus caixa blanca. Utilitza particions equivalents i statement coverage decision coverage i condition coverage.

El test1 comprova que el resultat de transformar el tauler a un string es l'esperat.

El test2 comprova que per a tots els valors possibles d'entrada, es faci la transformació de manera correcta.

```
    public char[][] renderWithPiece(final Piece piece) {
        char[][] rendered = new char[height][width];
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                rendered[y][x] = grid[y][x] ? 'X' : ' ';
            }
        }

        boolean[][] shape = piece.getShape();
        int pieceX = piece.getX();
        int pieceY = piece.getY();

        for (int row = 0; row < shape.length; row++) {
            for (int col = 0; col < shape[row].length; col++) {
                if (shape[row][col]) {
                    int renderX = pieceX + col;
                    int renderY = pieceY + row;

                    if (renderX >= 0 && renderX < width
                        && renderY >= 0 && renderY < height) {
                        rendered[renderY][renderX] = 'O';
                    }
                }
            }
        }

        return rendered;
    }
```

Funcionalitat: Moure una peça a un altre posició.

Localització: Arxiu Piece.java, classe Piece, funció move(final int dx, final int dy).

Test1: Arxiu PieceTest.java, classe PieceTest, funció testMovePiece(). Tipus caixa negra. Utilitza particions equivalents i statement coverage.

Test2: Arxiu PieceTest.java, classe PieceTest, funció testMovePieceNegativeValues(). Tipus caixa blanca. Utilitza particions equivalents i statement coverage.

El test1 comprova que en intentar moure una peça aquesta es mogui a la posició esperada.

El test2 comprova que en passar valors negatius a la funció per desfer un moviment, aquesta ho fa correctament deixant la peça a la posició esperada.

Funcionalitat: Girar una peça 90 graus en el sentit de les agulles del rellotge.

Localització: Arxiu Piece.java, classe Piece, funció rotateClockwise().

Test1: Arxiu PieceTest.java, classe PieceTest, funció testRotateClockwise(). Tipus caixa negra. Utilitza particions equivalents i statement coverage.

Test2: Arxiu PieceTest.java, classe PieceTest, funció testRotateSquareClockwise(). Tipus caixa negra. Utilitza particions equivalents i statement coverage.

Test3: Arxiu PieceTest.java, classe PieceTest, funció testRotateClockwiseLoopCases(). Tipus caixa negra. Utilitza particions equivalents i statement coverage i loop testing aniuat.

El test1 comprova que en intentar rotar una peça aquesta passi a tenir la forma esperada.

El test2 comprova que en intentar rotar una peça quadrada la forma d'aquesta es mantingui igual.

El test3 comprova que la funció rotateClockwise() funciona correctament per a qualsevol combinació d'iteracions.

```
    /** Rotación de la pieza en el sentido de las agujas del reloj.*/
    public void rotateClockwise() {
        int rows = shape.length;
        int cols = shape[0].length;
        boolean[][] rotatedShape = new boolean[cols][rows];

        for (int i = 0; i < rows; i++) {
            for (int j = 0; j < cols; j++) {
                rotatedShape[j][rows - 1 - i] = shape[i][j];
            }
        }
        shape = rotatedShape;
    }
```

Funcionalitat: Girar una peça 90 graus en el sentit contrari a les agulles del rellotge.

Localització: Arxiu Piece.java, classe Piece, funció rotateCounterClockwise().

Test1: Arxiu PieceTest.java, classe PieceTest, funció testRotateCounterClockwise(). Tipus caixa negra. Utilitza particions equivalents i statement coverage.

Test2: Arxiu PieceTest.java, classe PieceTest, funció testRotateSquareCounterClockwise(). Tipus caixa negra. Utilitza particions equivalents i statement coverage.

El test1 comprova que en intentar rotar una peça aquesta passi a tenir la forma esperada.

El test2 comprova que en intentar rotar una peça quadrada la forma d'aquesta es mantingui igual.

Funcionalitat: Generar una peça de forma aleatòria al principi del tauler.

Localització: Arxiu Piece.java, classe Piece, funció generateRandomPiece().

Test: Arxiu PieceTest.java, classe PieceTest, funció testGenerateRandomPiece(). Tipus caixa negra. Statement coverage i casos límit.

Funcionalitat: Posicionar una peça a una posició especifica del tauler.

Localització: Arxiu Piece.java, classe Piece, funció setPosition(final int newX, final int newY).

Test: Arxiu PieceTest.java, classe PieceTest, funció testSetPosition(). Tipus caixa negra. Statement coverage.

Funcionalitat: Generar una nova peça al tauler.

Localització: Arxiu GameController.java, classe GameController, funció spawnNewPiece().

Test1: Arxiu GameControllerTest.java, classe GameControllerTest, funció testSpawnNewPiecePosition().
Tipus caixa negra. Statement coverage.

Test2: Arxiu GameControllerTest.java, classe GameControllerTest, funció testGameOverWhenNoSpaceForNewPiece().
Tipus caixa blanca. Statement coverage, decision coverage i mock objects amb mockito.

El test1 comprova que en generar una peça, aquesta es situa al mig de la primera fila del tauler.

El test2 comprova que si a l'hora de generar una peça no hi ha espai a la posició inicial, el joc s'acaba.

```
    /**
     * Genera una nueva pieza en el tablero.
     * La pieza empieza en el centro superior.
     */
    public void spawnNewPiece() {
        this.currentPiece = Piece.generateRandomPiece();
        currentPiece.setPosition(board.getWidth() / 2, 0);
        if (board.checkCollision(currentPiece)) {
            this.isGameOver = true;
        }
    }
```

Funcionalitat: Moure una peça cap a avall.

Localització: Arxiu GameController.java, classe GameController, funció movePieceDown().

Test1: Arxiu GameControllerTest.java, classe GameControllerTest, funció testMovePieceDown().
Tipus caixa negra. Utilitza particions equivalents i Statement coverage.

Test2: Arxiu GameControllerTest.java, classe GameControllerTest, funció testMovePieceDownPathCoverage()
Tipus caixa negra. Statement coverage, decision coverage i path coverage.

El test1 comprova que en intentar moure una peça cap a avall, aquesta es mogui i passi a estar a la posició esperada.

El test2 comprova el correcte funcionament del programa per a tots els paths de la funció.

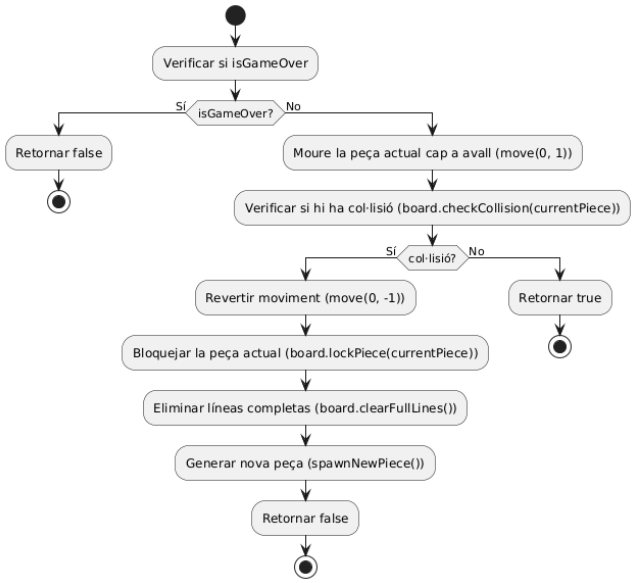

Diagrama de flux de movePieceDown()

```java
/**
 * Mueve la pieza actual hacia abajo una posición.
 * @return true si el movimiento fue exitoso, false si no fue posible.
 */
public boolean movePieceDown() {
    if (isGameOver) {
        return false;
    }

    currentPiece.move(0, 1);
    if (board.checkCollision(currentPiece)) {
        currentPiece.move(0, -1);  // Revertir movimiento
        board.lockPiece(currentPiece);
        board.clearFullLines();
        spawnNewPiece();
        return false;
    }
    return true;
}
```

Funcionalitat: Moure una peça cap a l'esquerra.

Localització: Arxiu GameController.java, classe GameController, funció movePieceLeft().

Test1: Arxiu GameControllerTest.java, classe GameControllerTest, funció testMovePieceLeft(). Tipus caixa negra. Utilitza particions equivalents i Statement coverage.

Test2: Arxiu GameControllerTest.java, classe GameControllerTest, funció testMovePieceLeftLoop().
Tipus caixa negra. Utilitza particions equivalents, Statement coverage.

El test1 comprova que en intentar moure una peça cap a l'esquerra, aquesta es mogui i passi a estar a la posició esperada.

El test2 comprova que en intentar moure una peça cap a l'esquerra des de diverses posicions del tauler cap a la vorera del mateix, aquesta no se surt en cap cas i es mou les posicions esperades.

Funcionalitat: Moure una peça cap a la dreta.

Localització: Arxiu GameController.java, classe GameController, funció movePieceRight().

Test1: Arxiu GameControllerTest.java, classe GameControllerTest, funció testMovePieceRight(). Tipus caixa negra. Utilitza particions equivalents i Statement coverage.

Test2: Arxiu GameControllerTest.java, classe GameControllerTest, funció testMovePieceRightLoop().
Tipus caixa negra. Utilitza particions equivalents, Statement coverage.

El test1 comprova que en intentar moure una peça cap a la dreta, aquesta es mogui i passi a estar a la posició esperada.

El test2 comprova que en intentar moure una peça cap a la dreta des de diverses posicions del tauler cap a la vorera del mateix, aquesta no se surt en cap cas i es mou les posicions esperades.

Funcionalitat: Girar una peça 90 graus.

Localització: Arxiu GameController.java, classe GameController, funció rotatePiece(final boolean clockwise).

Test1: Arxiu GameControllerTest.java, classe GameControllerTest, funció testRotatePiece(); Tipus caixa negra. Utilitza particions equivalents i Statement coverage.

Test2: Arxiu GameControllerTest.java, classe GameControllerTest, funció testRotateClockwisePieceWithCollision().
Tipus caixa Blanca. Statement i mock objects amb mockito.

Test3: Arxiu GameControllerTest.java, classe GameControllerTest, funció testRotateCounterClockwisePieceWithCollision().
Tipus caixa Blanca. Statement i mock objects amb mockito.

El test1 comprova que en intentar girar una peça, el resultat sigui el esperat.

El test2 comprova que en cas de trobar una col·lisio en intentar girar una peça en el sentit horari, aquest gir es reveteixi.

El test3 comprova que en cas de trobar una col·lisio en intentar girar una peça en el sentit antihorari, aquest gir es reveteixi.

Problems Javadoc Declaration Console Coverage

BoardTest (1) (30 nov 2024 17:37:01)

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Board	100,0 %	346	0	346
Board(int, int)	100,0 %	16	0	16
checkCollision(Piece)	100,0 %	66	0	66
clearFullLines()	100,0 %	28	0	28
getHeight()	100,0 %	3	0	3
getWidth()	100,0 %	3	0	3
isCellOccupied(int, int)	100,0 %	21	0	21
isFull(boolean[])	100,0 %	22	0	22
lockPiece(Piece)	100,0 %	45	0	45
occupyCell(int, int)	100,0 %	20	0	20
prependRow(boolean[]	100,0 %	22	0	22
renderWithPiece(Piece)	100,0 %	89	0	89

Cobertura al arxiu Board.java

Problems Javadoc Declaration **Console** Coverage

GameControllerTest.testMovePieceDownPathCoverage (30 nov 2024 16:58:41)

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
Piece.java	100,0 %	386	0	386
Piece	100,0 %	386	0	386
generateRandomPiece(	100,0 %	241	0	241
Piece(boolean[][])	100,0 %	12	0	12
getHeight()	100,0 %	4	0	4
getShape()	100,0 %	3	0	3
getWidth()	100,0 %	6	0	6
getX()	100,0 %	3	0	3
getY()	100,0 %	3	0	3
move(int, int)	100,0 %	13	0	13
rotateClockwise()	100,0 %	47	0	47
rotateCounterClockwise	100,0 %	47	0	47
setPosition(int, int)	100,0 %	7	0	7

Cobertura al arxiu Piece.java

Problems Javadoc Declaration **Console** Coverage

GameControllerTest (30 nov 2024 18:13:44)

Element	Coverage	Covered Instru...	Missed Instruct...	Total Instructio...
controller	100,0 %	156	0	156
GameController.java	100,0 %	156	0	156
GameController	100,0 %	156	0	156
GameController(Board)	100,0 %	11	0	11
getBoard()	100,0 %	3	0	3
getCurrentPiece()	100,0 %	3	0	3
getIsGameOver()	100,0 %	3	0	3
movePieceDown()	100,0 %	35	0	35
movePieceLeft()	100,0 %	20	0	20
movePieceRight()	100,0 %	20	0	20
rotatePiece(boolean)	100,0 %	31	0	31
setIsGameOver(boolea	100,0 %	4	0	4
setPiece(Piece)	100,0 %	4	0	4
spawnNewPiece()	100,0 %	22	0	22

Cobertura al arxiu GameController.java