

# **Informe Práctica TQS 2024**

## **CHESS GAME**

Antonio Lozano Larrubia  
Eloi Milego Miralles  
Dimecres 10:30-12:30h  
Chess Game

# Index

<b>Controller</b>	<b>3</b>
<b>Model</b>	<b>5</b>
Board	5
Square	8
Player	10
King	11
Bishop	12
Queen	15
Knight	18
Rook	20
Pawn	22

# Controller

controller	100,0 %	190	0	190
GameController.java	100,0 %	190	0	190

**Funcionalitat:** Inicialització del joc amb els jugadors i el tauler.

**Descripció:** Configura el joc inicialitzant els jugadors, el tauler i assignant el torn inicial al jugador blanc, a més de rebre moviment de la vista i controlar si es pot fer el moviment.

**Localització:**

Arxiu: src/main/java/controller/GameController.java

Classe: GameController

Mètode: GameController() (constructor)

**Test:**

Arxiu: src/test/java/controller/GameControllerTest.java

Classe: GameControllerTest

Mètode: testInitialTurnIsWhite()

Tipus de test: Caixa blanca

Tècnica utilitzada: Path Coverage

```
public GameController(GameView view) {  
    this.view = view;  
    board = new Board();  
    whitePlayer = new Player(Color.WHITE);  
    blackPlayer = new Player(Color.BLACK);  
    actualTurn = whitePlayer;  
    isGameOver = false;  
}
```

---

**Funcionalitat:** Comprovació del final del joc.

**Descripció:** Determina si el joc ha acabat comprovant si un dels reis (blanc o negre) ha estat capturat.

**Localització:**

Arxiu: src/main/java/controller/GameController.java

Classe: GameController

Mètode: checkGameOver()

**Test:**

Arxiu: src/test/java/controller/GameControllerTest.java

Classe: GameControllerTest

Mètodes:

- testGameOverWhenBlackKingsCaptured()
- testGameOverWhenWhiteKingsCaptured()
- testNotGameOver()
- testGameOverWhenBlackKingsCapturedWithMock()
- testGameOverWhenWhiteKingsCapturedWithMock()

Tipus de test: Caixa blanca i Mock Object.

Tècnica utilitzada: Path Coverage

```
public boolean checkGameOver() {  
    // Check if the kings of both colors are still on the board  
    boolean whiteKingExists = board.hasKing(Color.WHITE);  
    boolean blackKingExists = board.hasKing(Color.BLACK);  
    if (!whiteKingExists || !blackKingExists) {  
        //System.out.println(!whiteKingExists ? "Black" : "White") + " player has won.");  
        isGameOver = true;  
        return true;  
    }  
}
```

---

**Funcionalitat:** Gestió del moviment de peces al tauler.

**Descripció:** Valida i executa el moviment d'una peça segons les regles del joc, actualitza el tauler i canvia el torn.

**Localització:**

Arxiu: src/main/java/controller/GameController.java

Classe: GameController

Mètode: makeMove()

**Test:**

Arxiu: src/test/java/controller/GameControllerTest.java

Classe: GameControllerTest

Mètodes:

- testInvalidMoveOutsideBounds()
- testSimulateGame()
- testSimulateGameWithMock()
- testValidMoveWithMockBoardWithMock()

Tipus de test: Caixa negra, Caixa Blanca i amb Mock Object

Tècniques utilitzades: Particions equivalents, Valors Frontera i Path Coverage

```
·public boolean makeMove(int startRow, int startColumn, int destRow, int destColumn) {  
···· Square origin = board.getSquare(startRow, startColumn);  
···· Square destination = board.getSquare(destRow, destColumn);  
  
···· // If both squares are valid, attempt to move the piece && can move piece  
···· if (origin != null && destination != null && board.movePiece(origin, destination)) {  
···· ···· changeTurn();  
···· ···· return true;  
···· }  
···· return false; // Return false if the move could not be made  
·}
```

---

**Funcionalitat:** Control del torn dels jugadors.

**Descripció:** Alterna el torn entre els jugadors blanc i negre després d'un moviment vàlid.

**Localització:**

Arxiu: src/main/java/controller/GameController.java

Classe: GameController

Mètode: changeTurn()

**Test:**

Arxiu: src/test/java/controller/GameControllerTest.java

Classe: GameControllerTest

Mètode: testDisplayTurn()

Tipus de test: Caixa negra

Tècnica utilitzada: Particions equivalents

# Model

## Board

> Board.java 100,0 % 395 0 395

**Funcionalitat:** Inicialització del tauler amb peces en les posicions inicials

**Descripció:** Configura un tauler de 8x8 amb totes les peces col·locades a les seves posicions inicials per a un joc d'escacs estàndard.

**Localització:**

Arxiu: src/main/java/model/Board.java

Classe: Board

Mètodes:

- Board() (constructor per defecte)
- initializeBoard()

Test:

Arxiu: src/test/java/model/BoardTest.java

Classe: BoardTest

Mètode: testInitializeBoard()

Tipus de test: Caixa blanca

Tècnica utilitzada: Path coverage

```
...// Initializes the board with pieces in their starting positions for a standard
...public void initializeBoard(int rows, int cols) {
...    // Create squares for the board.
...    for (int row = 0; row < rows; row++) {
...        for (int column = 0; column < cols; column++) {
...            squares[row][column] = new Square(row, column);
...        }
...    }
...    // Place pawns on the second and seventh rows.
...    for (int column = 0; column < 8; column++) {
...        squares[1][column].setPiece(new Pawn(Color.WHITE));
...        squares[6][column].setPiece(new Pawn(Color.BLACK));
...    }
...    // Place rooks in their starting positions.
...    squares[0][0].setPiece(new Rook(Color.WHITE));
...    squares[0][7].setPiece(new Rook(Color.WHITE));
...    squares[7][0].setPiece(new Rook(Color.BLACK));
...    squares[7][7].setPiece(new Rook(Color.BLACK));
...    // Place knights in their starting positions.
...    squares[0][1].setPiece(new Knight(Color.WHITE));
...    squares[0][6].setPiece(new Knight(Color.WHITE));
...    squares[7][1].setPiece(new Knight(Color.BLACK));
...    squares[7][6].setPiece(new Knight(Color.BLACK));
...    // Place bishops in their starting positions.
...    squares[0][2].setPiece(new Bishop(Color.WHITE));
...    squares[0][5].setPiece(new Bishop(Color.WHITE));
...    squares[7][2].setPiece(new Bishop(Color.BLACK));
...    squares[7][5].setPiece(new Bishop(Color.BLACK));
...    // Place queens in their starting positions.
...    squares[0][3].setPiece(new Queen(Color.WHITE));
...    squares[7][3].setPiece(new Queen(Color.BLACK));
...    // Place kings in their starting positions.
...    squares[0][4].setPiece(new King(Color.WHITE));
...    squares[7][4].setPiece(new King(Color.BLACK));
...}
```

**Funcionalitat:** Comprovar si un rei d'un color està present al tauler

**Descripció:** Itera per totes les caselles del tauler per determinar si hi ha un rei d'un color específic.

**Localització:**

Arxiu: src/main/java/model/Board.java

Classe: Board

Mètodes:

- hasKing()

**Test:**

Arxiu: src/test/java/model/BoardTest.java

Classe: BoardTest

Mètode:

- testHasKing\_WhiteKingPresent()
- testHasKing\_BlackKingPresent()
- testHasKing\_NoKingPresent()
- testHasKingAvoidInnerLoop()
- testHasKingOneIterationInnerLoop()
- testHasKingTwoIterationsInnerLoop()
- testHasKingFiveIterationsInnerLoop()
- testHasKingSevenIterationsInnerLoop()
- testHasKingEightIterationsInnerLoop()
- testHasKingAvoidOuterLoop()
- testHasKingOneIterationOuterLoop()
- testHasKingTwoIterationOuterLoop()
- testHasKingFiveIterationOuterLoop()
- testHasKingSevenIterationOuterLoop()
- testHasKingEightIterationOuterLoop()

Tipus de test: Caixa blanca

Tècnica utilitzada: Loop Testing

```
.....// Checks if the board contains a king of the specified color.
public boolean hasKing(Color color) {
    .....for (int row = 0; row < sizeRows; row++) {
    .....    for (int column = 0; column < sizeCols; column++) {
    .....        Square square = squares[row][column];
    .....        // Check if the square contains a king of the specified color.
    .....        if (square.getPiece().instanceof King && square.getPiece().getColor() == color) {
    .....            return true;
    .....        }
    .....    }
    .....}
    .....return false;
}
```

---

**Funcionalitat:** Comprovar si es pot fer un moviment

**Descripció:** Es comprova que la peça pot fer un moviment dins del tauler, i si pot, es fa

**Localització:**

Arxiu: src/main/java/model/Board.java

Classe: Board

Mètodes:

- movePiece(Square origin, Square destination)

**Test:**

Arxiu: src/test/java/model/BoardTest.java

Classe: BoardTest

Mètode:

- testMovePiece\_ValidMove()
- testMovePiece\_InvalidMove()
- testMovePiece\_Pairwise()

Tipus de test: Caixa negra i caixa blanca

Tècniques utilitzades: Partició equivalent amb valors fronteres i límits, Path coverage i Pairwise testing.

```
...// Moves a piece from the origin square to the destination square if the move is valid.
...// Returns true if the move was successful, false otherwise.
...public boolean movePiece(Square origin, Square destination) {
...    if (origin.getPiece() != null && origin.getPiece().validMovement(destination, this)) {
...        destination.setPiece(origin.getPiece());
...        origin.setPiece(null);
...        return true;
...    } else {
...        return false;
...    }
...}
```

---

**Funcionalitat:** Obtenir una casella del tauler per coordenades

**Descripció:** Retorna la casella corresponent a les coordenades especificades, o null si les coordenades estan fora dels límits del tauler.

**Localització:**

Arxiu: src/main/java/model/Board.java

Classe: Board

Mètode: getSquare()

**Test:**

Arxiu: src/test/java/model/BoardTest.java

Classe: BoardTest

Mètodes:

- testGetSquare\_ValidCoordinates()
- testGetSquare\_InvalidCoordinates()

Tipus de test: Caixa negra

Tècnica utilitzada: Particions equivalents amb valors fronteres i límits.

---

**Comentari de la classe Board:** No fiquem design by contract ja que necessitem modificar el tauler per fer per exemple loop testing.

# Square

>  Square.java 100,0 % 43 0 43

---

**Funcionalitat:** Inicialitzar una casella amb coordenades i peça

**Descripció:** Crea una casella del tauler especificant la fila, columna i, opcionalment, una peça que l'ocupi. Si no s'especifica una peça, la casella queda buida.

**Localització:**

Arxiu: src/main/java/model/Square.java

Classe: Square

Mètodes:

Square(int row, int column, Piece piece)

Square(int row, int column)

**Test:**

Arxiu: src/test/java/model/SquareTest.java

Classe: SquareTest

Mètode: testSquare()

Tipus de test: Caixa blanca

Tècnica utilitzada: Path coverage

```
public Square(int fila, int columna, Piece piece) {  
    this.row = fila;  
    this.column = columna;  
    this.piece = piece;  
}  
  
/** Constructor: initializes a square with specified row and column,  
 * Parameters:  
 * -- row: the row index of the square.  
 * -- column: the column index of the square.  
 */  
public Square(int row, int column) {  
    this(row, column, null);  
}
```

---

**Funcionalitat:** Verificar si una casella està ocupada per una peça

**Descripció:** Determina si la casella conté una peça retornant true o false.

**Localització:**

Arxiu: src/main/java/model/Square.java

Classe: Square

Mètode: isOccupied()

**Test:**

Arxiu: src/test/java/model/SquareTest.java

Classe: SquareTest

Mètode: testIsOccupied()

Tipus de test: Caixa negra

Tècnica utilitzada: Particions equivalents amb valors límit i frontera.

---



**Funcionalitat:** Assignar una peça a una casella

**Descripció:** Col·loca una peça en la casella, actualitzant la seva posició. Si la peça és null, la casella queda buida.

**Localització:**

Arxiu: src/main/java/model/Square.java

Classe: Square

Mètode: setPiece()

**Test:**

Arxiu: src/test/java/model/SquareTest.java

Classe: SquareTest

Mètode: testIsOccupied()

Tipus de test: Caixa blanca

Tècnica utilitzada: Path coverage

```
// Setter: sets a piece to occupy this square.
// Parameters:
// - piece: the piece to place on the square.
// If the piece is not null, its position is updated to this square.
public void setPiece(Piece piece) {
    this.piece = piece;
    if (piece != null) {
        piece.setPosition(this);
    }
}
```

---

**Funcionalitat:** Recuperar informació sobre una casella

**Descripció:** Obté informació com les coordenades (fila i columna) i la peça que ocupa la casella, si n'hi ha.

**Localització:**

Arxiu: src/main/java/model/Square.java

Classe: Square

Mètodes:

- getRow()
- getColumn()
- getPiece()

**Test:**

Arxiu: src/test/java/model/SquareTest.java

Classe: SquareTest

Mètode: testSquare()


Tipus de test: Caixa blanca

Tècnica utilitzada: Statement coverage

---

**Comentari de la classe Square:** No fiquem design by contract ja que necessitem provar peces en diferents llocs del tauler en altres classes.

# Player

>  Player.java

100,0 %

9

0

9

---

**Funcionalitat:** Inicialitzar un jugador amb un color assignat

**Descripció:** Crea un jugador assignant-li un color específic, blanc o negre, per determinar quines peces controla durant la partida.

**Localització:**

Arxiu: src/main/java/model/Player.java

Classe: Player

Mètode: Player(Color color) (constructor)

**Test:**

Arxiu: src/test/java/model/PlayerTest.java

Classe: PlayerTest

Mètodes:

- testPlayerColorWhite()
- testPlayerColorBlack()

Tipus de test: Caixa blanca

Tècnica utilitzada: Path Coverage

---

**Funcionalitat:** Recuperar el color del jugador

**Descripció:** Proporciona el color assignat al jugador per identificar quines peces controla.

**Localització:**

Arxiu: src/main/java/model/Player.java

Classe: Player

Mètode: getColor()

**Test:**

Arxiu: src/test/java/model/PlayerTest.java

Classe: PlayerTest

Mètodes:

- testPlayerColorWhite()
- testPlayerColorBlack()
- testColorIsNotNull()

Tipus de test: Caixa blanca

Tècnica utilitzada: Path Coverage

---

```
...// Constructor: initializes the player with a specific color.
...// Parameters:
...// - color: the color assigned to the player, determining which pieces they control.
public Player(Color color) {
    ...this.color = color;
}

...// Getter: returns the color of the player.
...// This method is used to identify which pieces belong to the player.
public Color getColor() {
    ...return color;
}
```

# King

> King.java

100,0 %

107

0

107

**Funcionalitat:** Retornar nom de la peça

**Descripció:** Implementació d'un mètode que retorna el nom del rei segons el color que tingui.

**Localització:**

Arxiu: src/main/java/model/King.java

Classe: King

Mètode:

- King(Color color) (Constructor)
- getName()

**Test:**

Arxiu: src/test/java/model/KingTest.java

Classe: KingTest

Mètodes:

- testKingGetColors()
- testKingGetPositionInBoard()
- testKingGetName()

Tipus de test: Caixa blanca

Tècniques utilitzades: Path coverage

```
// Constructor to initialize the King with a specific color.
public King(Color color) {
    super(color);
}
@Override
public String getName() {
    return (this.color == Color.WHITE ? "W.King" : "B.King");
}
```

**Funcionalitat:** Verificar si un moviment del rei és vàlid

**Descripció:** Determina si un moviment és vàlid segons les regles dels escacs: un sol quadrat en qualsevol direcció (horitzontal, vertical o diagonal), sempre que la casella de destinació no estigui ocupada per una peça amiga o fora del tauler.

**Localització:**

Arxiu: src/main/java/model/King.java

Classe: King

Mètode: validMovement(Square destination, Board board)

**Test:**

Arxiu: src/test/java/model/KingTest.java

Classe: KingTest

Mètodes:

- testWhiteKingCanMoveOneSquare()
- testWhiteKingCannotMoveTwoSquares()
- testBlackKingCannotMoveToSamePosition()
- testWhiteKingCanCaptureBlackPiece()
- testBlackKingCannotCaptureSameColorPiece()
- testKingCannotMoveThroughPieces()
- testKingCannotMoveToInvalidPosition()
- testOutOfBound()

Tipus de test: Caixa negra i Caixa Blanca

Tècniques utilitzades: Particions equivalents amb valors frontes i límits, Path Coverage.

```

@Override
public boolean validMovement(Square destination, Board board) {
    // Preconditions and invariant
    assert destination != null : "Destination square cannot be null.";
    assert destination.getRow() >= 0 && destination.getRow() < board.getSizeRows() : "Row is out of bounds.";
    assert destination.getColumn() >= 0 && destination.getColumn() < board.getSizeCols() : "Column is out of bounds.";
    assert checkInvariants() : "King's state invariant violated: color cannot be null.";

    // Calculate the row and column differences between the current position and the destination.
    int rowDelta = Math.abs(destination.getRow() - this.position.getRow());
    int colDelta = Math.abs(destination.getColumn() - this.position.getColumn());

    // Check that movement is one square in any direction
    if ((rowDelta <= 1 && colDelta <= 1) && !(rowDelta == 0 && colDelta == 0)) {
        Piece destinationPiece = destination.getPiece();

        // Check if the destination square is occupied.
        if (destinationPiece != null && destinationPiece.getColor() == this.color) {
            return false;
        }
        return true;
    }

    // If the move is more than one square in any direction, return false
    return false;
}

private boolean checkInvariants() {
    return this.color != null;
}

```

## Bishop

>  Bishop.java  100,0 % 150 0 150

**Funcionalitat:** Retornar nom de la peça

**Descripció:** Implementació d'un mètode que retorna el nom del alfil segons el color que tingui.

**Localització:**

Arxiu: src/main/java/Bishop.java

Classe: Bishop

Mètode:

- Bishop(Color color) (Constructor)
- getName()

**Test:**

Arxiu: src/test/java/BishopTest.java

Classe: BishopTest

Mètodes:

- testBishopGetColors()
- testBishopGetPositionInBoard()
- testBishopGetName()

Tipus de test: Caixa blanca

Tècnica utilitzada: Path coverage

```

    public Bishop(Color color) {
        super(color);
    }

    @Override
    public String getName() {
        // Returns the name of the piece, prefixed with its color.
        return (this.color == Color.WHITE ? "W.Bishop" : "B.Bishop");
    }

```

---

**Funcionalitat:** Verificar si el moviment de l'alfil és vàlid

**Descripció:** Determina si un moviment és vàlid segons les regles dels escacs: en aquesta classe el moviment és vàlid si aquest és en diagonal, sempre que la casella de destinació no estigui ocupada per una peça amiga o fora del tauler.

**Localització:**

Arxiu: src/main/java/Bishop.java

Classe: Bishop

Mètode: validMovement(Square destination, Board board)

**Test:**

Arxiu: src/test/java/BishopTest.java

Classe: BishopTest

Mètodes:

- testBishopCanMoveDiagonally()
- testBishopCannotMoveStraight()
- testBishopCanCaptureBlackBishop()
- testBishopCannotCaptureSameColorPiece()
- testBishopCannotMoveThroughPieces()
- testBishopCannotMoveToInvalidPosition()
- testBishopCannotMoveHorizontallyOrVertically()
- testBishopWithoutColor()
- testBishopInvalidMoveOutOfBounds()
- testMockBishopCanMoveDiagonally()
- testMockBishopCannotMoveStraight()
- testMockBishopCanCaptureEnemyPiece()
- testMockBishopCannotCaptureSameColorPiece()
- testMockBishopCannotMoveThroughPieces()
- testMockBishopRemainsWithinBoardBounds()

Tipus de test: Caixa negra i caixa blanca

Tècniques utilitzades: Particions equivalents amb valors frontes i límits, Path Coverage i Mock Objects amb Mockito.

```

@Override
public boolean validMovement(Square destination, Board board) {
    // Preconditions and invariant
    assert destination != null : "Destination square cannot be null.";
    assert destination.getRow() >= 0 && destination.getRow() < board.getSizeRows()
        : "Row is out of bounds.";
    assert destination.getColumn() >= 0 && destination.getColumn() < board.getSizeCols()
        : "Column is out of bounds.";
    assert checkInvariants() : "Bishop's state invariant violated: color cannot be null.";

    // Calculate the absolute row and column differences between current and destination positions.
    int rowDelta = Math.abs(destination.getRow() - this.position.getRow());
    int colDelta = Math.abs(destination.getColumn() - this.position.getColumn());

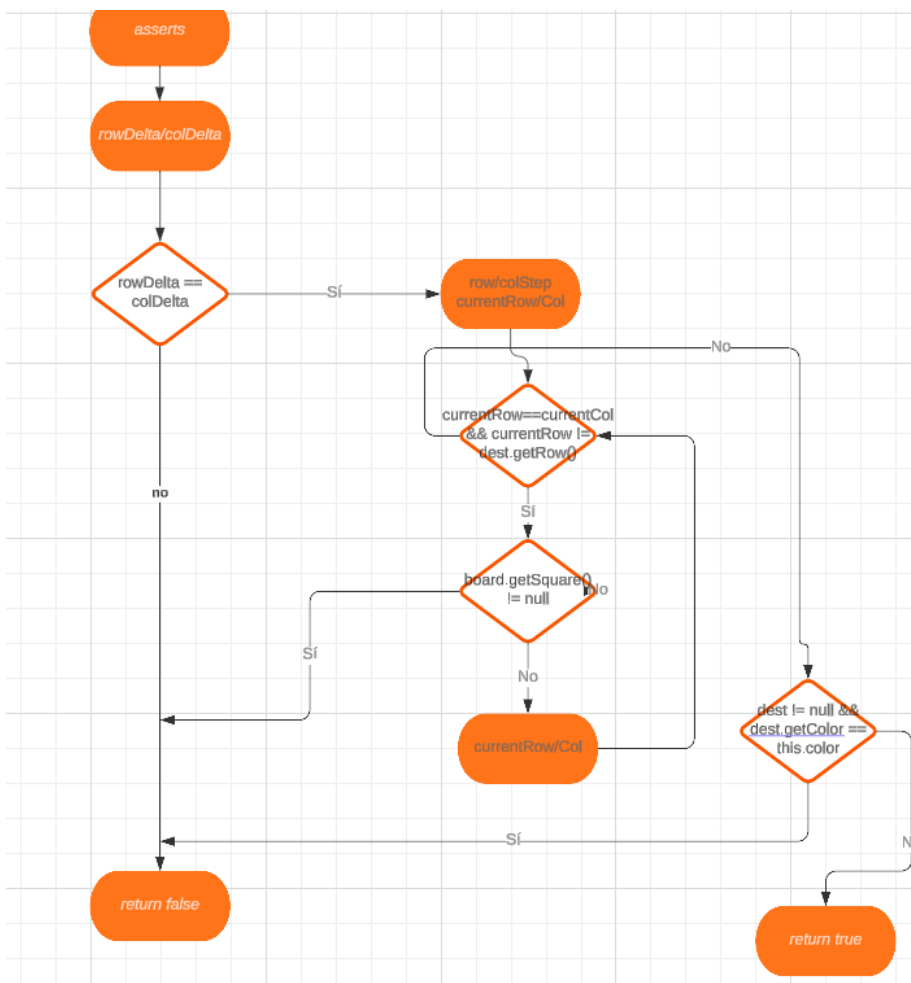
    // A bishop moves diagonally, so row and column deltas must be equal.
    if (rowDelta == colDelta) {
        // Determine the step direction for rows and columns.
        int rowStep = Integer.compare(destination.getRow(), this.position.getRow());
        int colStep = Integer.compare(destination.getColumn(), this.position.getColumn());

        // Traverse the diagonal path from the current position to the destination.
        int currentRow = this.position.getRow() + rowStep;
        int currentCol = this.position.getColumn() + colStep;
        while (currentRow == currentCol && currentRow != destination.getRow()) {
            // If any square along the path contains a piece, the move is invalid.
            if (board.getSquare(currentRow, currentCol).getPiece() != null) {
                return false;
            }
            currentRow += rowStep;
            currentCol += colStep;
        }


        // Ensure the destination square does not contain a piece of the same color.
        Piece destinationPiece = destination.getPiece();
        if (destinationPiece != null && destinationPiece.getColor() == this.color) {
            return false;
        }
        return true;
    }
    return false;
}

```

Diagrama:



# Queen

>  Queen.java

 100,0 %

155

0

155

---

**Funcionalitat:** Retornar nom de la peça

**Descripció:** Implementació d'un mètode que retorna el nom de la reina segons el color que tingui.

**Localització:**

Arxiu: src/main/java/Queen.java

Classe: Queen

Mètode:

- Queen(Color color) (Constructor)
- getName()

**Test:**

Arxiu: src/test/java/QueenTest.java

Classe: QueenTest

Mètodes:

- testQueenGetColors()
- testQueenGetPositionInBoard()
- testQueenGetName()

Tipus de test: Caixa blanca

Tècnica utilitzada: Path coverage

```
> public Queen(Color color) {  
    super(color);  
}  
  
@Override  
public String getName() {  
    return (this.color == Color.WHITE ? "W.Queen" : "B.Queen");  
}
```

---

**Funcionalitat:** Verificar si el moviment de la reina és vàlid

**Descripció:** Determina si un moviment és vàlid segons les regles dels escacs: el moviment d'una reina és vàlid per qualsevol destinació, sempre que la casella de destinació no estigui ocupada per una peça amiga o fora del tauler.

**Localització:**

Arxiu: src/main/java/Queen.java

Classe: Queen

Mètode: validMovement(Square destination, Board board)

**Test:**

Arxiu: src/test/java/QueenTest.java

Classe: QueenTest

Mètodes:

- testWhiteQueenCanMoveStraight()
- testWhiteQueenCanMoveHorizontal()

- testWhiteQueenCanMoveDiagonally()
- testWhiteQueenCanCaptureBlackQueen()
- testWhiteQueenCannotCaptureSameColorPiece()
- testQueenCannotMoveOutOfBounds()
- testQueenCannotMoveThroughPieces()
- testQueenCannotMoveDiagonallyThroughPieces()
- testBlackQueenCannotMoveToASpecificPosition()
- testQueenOutOfBound()
- testMockWhiteQueenCanMoveStraight()
- testMockWhiteQueenCanMoveDiagonally()
- testMockWhiteQueenCannotCaptureSameColorPiece()
- testMockQueenCannotMoveThroughPieces()
- testMockQueenRemainsWithinBoardBounds()

Tipus de test: Caixa negra i caixa blanca

Tècniques utilitzades: Particions equivalents amb valors fronteres i límits, Path Coverage i Mock Objects amb Mockito.

```
@Override
public boolean validMovement(Square destination, Board board) {
    // Preconditions and invariant
    assert destination != null : "Destination square cannot be null.";
    assert destination.getRow() >= 0 && destination.getRow() < board.getSizeRows()
        : "Row is out of bounds.";
    assert destination.getColumn() >= 0 && destination.getColumn() < board.getSizeCols()
        : "Column is out of bounds.";
    assert checkInvariants() : "Queen's state invariant violated: color cannot be null.";

    // Calculate the row and column differences between the current position and the destination.
    int rowDelta = Math.abs(destination.getRow() - this.position.getRow());
    int colDelta = Math.abs(destination.getColumn() - this.position.getColumn());

    // Can move in straight lines or diagonals
    if (rowDelta == colDelta || rowDelta == 0 || colDelta == 0) {
        // Determine the direction of movement in rows
        int rowStep = Integer.compare(destination.getRow(), this.position.getRow());
        // Determine the direction of movement in columns
        int colStep = Integer.compare(destination.getColumn(), this.position.getColumn());

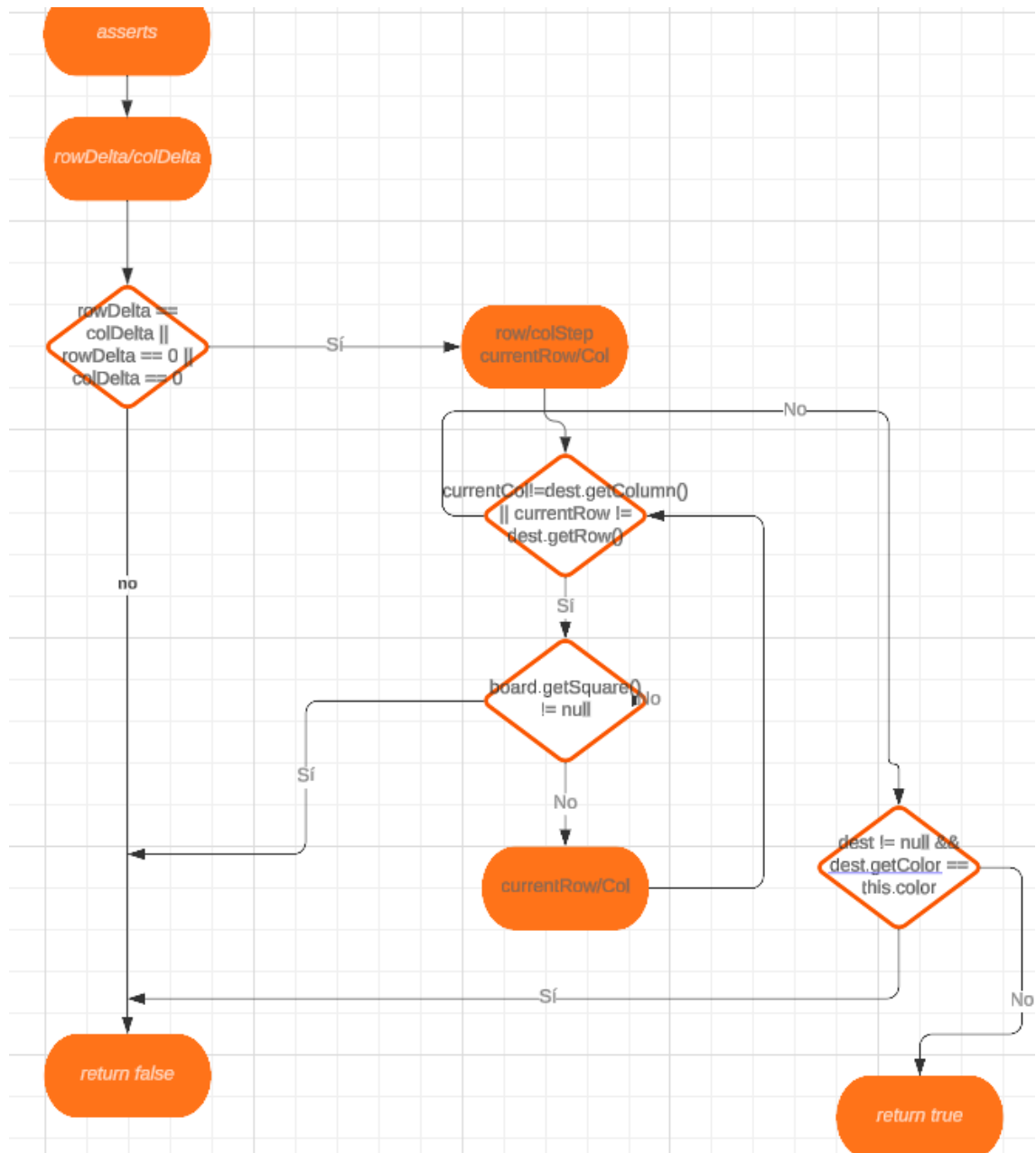
        // Check if there are pieces blocking the path
        int currentRow = this.position.getRow() + rowStep;
        int currentCol = this.position.getColumn() + colStep;
        while (currentRow != destination.getRow() || currentCol != destination.getColumn()) {
            if (board.getSquare(currentRow, currentCol).getPiece() != null) {
                return false; // Path is blocked
            }
            currentRow += rowStep;
            currentCol += colStep;
        }

        // Ensure the destination square is not occupied by a piece of the same color
        Piece destinationPiece = destination.getPiece();
        if (destinationPiece != null && destinationPiece.getColor() == this.color) {
            return false; // Cannot capture a friendly piece
        }

        return true;
    }
    // If the movement is not a valid diagonal or straight line, return false
    return false;
}
```



Diagrama:



# Knight

> Knight.java 100,0 % 114 0 114

---

**Funcionalitat:** Retornar nom de la peça

**Localització:**

Arxiu: src/main/java/Knight.java

Classe: Knight

Mètode:

- Knight(Color color) (Constructor)
- getName()

**Test:**

Arxiu: src/test/java/KnightTest.java

Classe: KnightTest

Mètodes:

- testKnightGetColors()
- testKnightGetPositionInBoard()
- testKnightGetName()

Tipus de test: Caixa blanca

Tècnica utilitzada: Path coverage

```
public Knight(Color color) {  
    super(color);  
}  
  
@Override  
public String getName() {  
    // Returns the name of the piece, prefixed with its color.  
    return (this.color == Color.WHITE ? "W.Knight" : "B.Knight");  
}
```

---

**Funcionalitat:** Verificar si el moviment del cavall és vàlid

Determina si un moviment és vàlid segons les regles dels escacs: en aquesta classe el moviment és vàlid si el moviment és en L, sempre que la casella de destinació no estigui ocupada per una peça amiga o fora del tauler.

**Localització:**

Arxiu: src/main/java/Knight.java

Classe: Knight

Mètode: validMovement(Square destination, Board board)

**Test:**

Arxiu: src/test/java/KnightTest.java

Classe: KnightTest

Mètodes:

- testBlackBoxValidLShapedMoveRowDelta2ColDelta1()
- testBlackBoxValidLShapedMoveRowDelta1ColDelta2()
- testBlackBoxInvalidStraightMove()
- testBlackBoxInvalidNonLShapedMove()

- testBlackBoxInvalidNonLShapedMove2()
- testBlackBoxInvalidNonLShapedMove3()
- testBlackBoxCannotMoveOutOfBoundsRowNegative()
- testBlackBoxCannotMoveOutOfBoundsRowExceedsBoard()
- testBlackBoxCannotMoveOutOfBoundsColumnNegative()
- testBlackBoxCannotMoveOutOfBoundsColumnExceedsBoard()
- testNullDestination()
- testWhiteBoxCanCaptureBlackKnight()
- testWhiteBoxCannotCaptureSameColorPiece()
- testInvalidLShapeMove()
- testCaptureNullColorInvariantCheck()

Tipus de test: Caixa negra i caixa blanca

Tècniques utilitzades: Particions equivalents amb valors fronteres i límits, Path Coverage.

```
@Override
public boolean validMovement(Square destination, Board board) {
    // Preconditions and invariant
    assert destination != null : "Destination square cannot be null.";
    assert destination.getRow() >= 0 && destination.getRow() < board.getSizeRows()
        : "Row is out of bounds.";
    assert destination.getColumn() >= 0 && destination.getColumn() < board.getSizeCols()
        : "Column is out of bounds.";
    assert checkInvariants() : "Knight's state invariant violated: color cannot be null.";

    // Calculate the absolute differences in rows and columns between the current position and destination.
    int rowDelta = Math.abs(destination.getRow() - this.position.getRow());
    int colDelta = Math.abs(destination.getColumn() - this.position.getColumn());

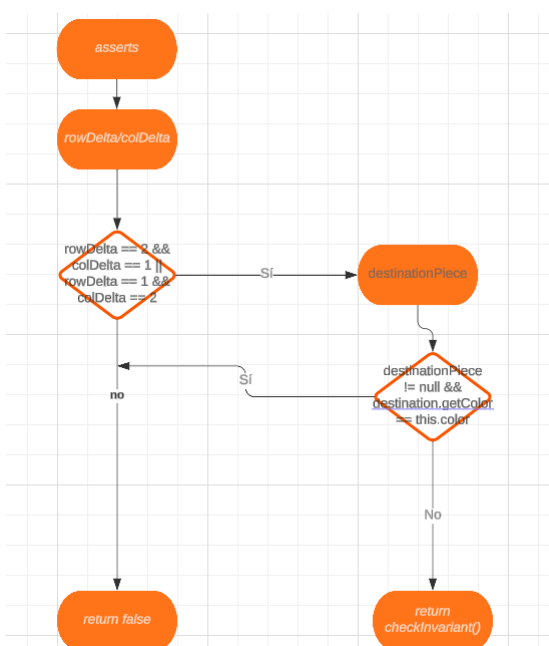
    // A knight moves in an "L" shape: two squares in one direction and one in the perpendicular direction.
    if ((rowDelta == 2 && colDelta == 1) || (rowDelta == 1 && colDelta == 2)) {
        // Retrieve the piece at the destination square, if any.
        Piece destinationPiece = destination.getPiece();

        // Ensure the destination square does not contain a piece of the same color.
        if (destinationPiece != null && destinationPiece.getColor() == this.color) {
            return false; // Cannot capture a piece of the same color.
        }


        // Check invariants for the Knight.
        return checkInvariants();
    }


    // If none of the valid movement conditions are met, the move is invalid.
    return false;
}
```

Diagrama:



# Rook

>  Rook.java

 100,0 %

152

0

152

---

**Funcionalitat:** Retornar nom de la peça

**Localització:**

Arxiu: src/main/java/Rook.java

Classe: Rook

Mètode:

- Rook(Color color) (Constructor)
- getName()

**Test:**

Arxiu: src/test/java/RookTest.java

Classe: RookTest

Mètodes:

- testRookGetColors()
- testRookGetPositionInBoard()
- testRookGetName()

Tipus de test: Caixa blanca

Tècnica utilitzada: Path coverage

```
// Constructor to initialize the Rook with a specific color.
public Rook(Color color) {
    super(color);
}

@Override
public String getName() {
    // Returns the name of the piece, prefixed with its color.
    return (this.color == Color.WHITE ? "W.Rook" : "B.Rook");
}
```

---

**Funcionalitat:** Verificar si el moviment de la torre és vàlid

Determina si un moviment és vàlid segons les regles dels escacs: en aquesta classe el moviment és vàlid si aquest és vertical o horitzontal, sempre que la casella de destinació no estigui ocupada per una peça amiga o fora del tauler.

**Localització:**

Arxiu: src/main/java/Rook.java

Classe: Rook

Mètode: validMovement(Square destination, Board board)

**Test:**

Arxiu: src/test/java/RookTest.java

Classe: RookTest

Mètodes:

- testWhiteRookCanMoveStraightHorizontally()
- testWhiteRookCanMoveStraightVertically()
- testWhiteRookCanCaptureBlackRook()
- testWhiteRookCannotCaptureSameColorPiece()
- testRookCannotMoveThroughPieces()

- testKingCannotMoveToInvalidPosition()
- testRookCannotMoveDiagonally()
- testRookCannotMoveOutOfBound()

Tipus de test: Caixa negra i caixa blanca

Tècniques utilitzades: Particions equivalents amb valors fronteres i límits, Path Coverage.

```
@Override
public boolean validMovement(Square destination, Board board) {
    // Preconditions and invariant
    assert destination != null : "Destination square cannot be null.";
    assert destination.getRow() >= 0 && destination.getRow() < board.getSizeRows()
        : "Row is out of bounds.";
    assert destination.getColumn() >= 0 && destination.getColumn() < board.getSizeCols()
        : "Column is out of bounds.";
    assert checkInvariants() : "Rook's state invariant violated: color cannot be null.";

    // Calculate the absolute row and column differences between the current position and the destination.
    int rowDelta = Math.abs(destination.getRow() - this.position.getRow());
    int colDelta = Math.abs(destination.getColumn() - this.position.getColumn());

    // A Rook can only move in straight lines (either rows or columns must remain constant).
    if (rowDelta == 0 || colDelta == 0) {
        // Determine the step direction for rows or columns.
        int rowStep = Integer.compare(destination.getRow(), this.position.getRow());
        int colStep = Integer.compare(destination.getColumn(), this.position.getColumn());

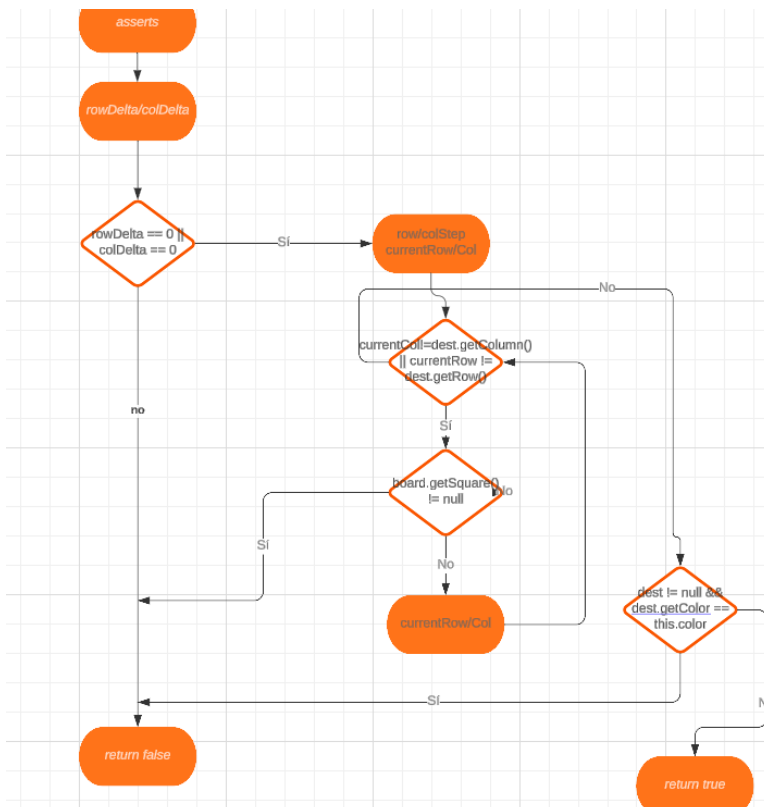
        // Check each square along the path to the destination.
        int currentRow = this.position.getRow() + rowStep;
        int currentCol = this.position.getColumn() + colStep;
        while (currentRow != destination.getRow() || currentCol != destination.getColumn()) {
            // If any square along the path contains a piece, the move is invalid.
            if (board.getSquare(currentRow, currentCol).getPiece() != null) {
                return false; // Path is blocked
            }
            currentRow += rowStep;
            currentCol += colStep;
        }

        // Ensure the destination square does not contain a piece of the same color.
        Piece destinationPiece = destination.getPiece();
        if (destinationPiece != null && destinationPiece.getColor() == this.color) {
            return false; // Cannot capture a friendly piece
        }


        return true; // The move is valid.
    }

    // The move is invalid if it is not in a straight line.
    return false;
}
```

Diagrama:



# Pawn

>  Pawn.java	 100,0 %	192	0	192
---	---	-----	---	-----

---

**Funcionalitat:** Retornar nom de la peça

**Localització:**

Arxiu: src/main/java/Pawn.java

Classe: Pawn

Mètode:

- Pawn(Color color) (Constructor)
- getName()

**Test:**

Arxiu: src/test/java/PawnTest.java

Classe: PawnTest

Mètodes:

- testPawnGetColors()
- testPawnGetPositionInBoard()
- testPawnGetName()

Tipus de test: Caixa blanca

Tècnica utilitzada: Path coverage

```
// Constructor to initialize the Pawn with a specific color.
public Pawn(Color color) {
    super(color);
}

@Override
public String getName() {
    // Returns the name of the piece, prefixed with its color.
    return (this.color == Color.WHITE ? "W.Pawn" : "B.Pawn");
}
```

---

**Funcionalitat:** Verificar si el moviment del peó és vàlid

Determina si un moviment és vàlid segons les regles dels escacs, moviment vàlid sempre cap endavant: Si estàs en posició inicial, pots avançar 1 o 2 caselles, sinó d'1 en 1, i es poden fer atacs en diagonal. Es prova tant el color blanc com el negre ja que aquí el color importa.

**Localització:**

Arxiu: src/main/java/Pawn.java

Classe: Pawn

Mètode: validMovement(Square destination, Board board)

**Test:**

Arxiu: src/test/java/PawnTest.java

Classe: PawnTest

Mètodes:

- testBlackPawnValidMovementForwardOneSquare()

- testBlackPawnValidMovementForwardTwoSquares()
- testBlackPawnDiagonalAttack()
- testBlackPawnCannotMoveBackward()
- testBlackPawnCannotCaptureSameColorPiece()
- testBlackPawnInvalidDiagonalMove()
- testBlackPawnCannotMoveToOutOfBounds()
- testBlackPawnDoubleForwardBlocked()
- testBlackPawnCannotMoveForwardForBeingBlocked()
- testBlackPawnCannotMoveTwoForwardInDiagonal()
- testBlackPawnCannotMoveTwoForwardForNotBeingInStartingPosition()
- testBlackPawnCannotMoveTwoForwardForBeingBlockBySameColor()
- testWhitePawnValidMovementForwardOneSquare()
- testWhitePawnValidMovementForwardTwoSquares()
- testWhitePawnDiagonalAttack()
- testWhitePawnCannotMoveBackward()
- testWhitePawnCannotCaptureSameColorPiece()
- testWhitePawnInvalidDiagonalMove()
- testPawnCannotMoveInvalidPosition()
- testWhitePawnDoubleForwardBlocked()
- testWhitePawnCannotMoveForwardForBeingBlocked()
- testWhitePawnCannotMoveTwoForwardInDiagonal()
- testWhitePawnCannotMoveTwoForwardForNotBeingInStartingPosition()
- testWhitePawnCannotMoveTwoForwardForBeingBlockBySameColor()
- testPawnCannotMoveBackward()
- testOutOfBound()
- testPawnWithoutColor()

Tipus de test: Caixa negra i caixa blanca

Tècniques utilitzades: Particions equivalents amb valors frontes i límits, Path Coverage.

```
@Override
public boolean validMovement(Square destination, Board board) {
    // Preconditions and invariant
    assert destination != null : "Destination square cannot be null.";
    assert destination.getRow() >= 0 && destination.getRow() < board.getSizeRows()
        : "Row is out of bounds.";
    assert destination.getColumn() >= 0 && destination.getColumn() < board.getSizeCols()
        : "Column is out of bounds.";
    assert checkInvariants() : "Pawn's state invariant violated: color cannot be null.";

    // Basic logic for pawn movement (only forward, or diagonal attack)
    int rowDelta = destination.getRow() - this.position.getRow();
    int colDelta = Math.abs(destination.getColumn() - this.position.getColumn());

    // If the pawn is black, it moves downward the board
    if (color == Color.BLACK) {
        // Move forward one square if the destination is not occupied
        if (rowDelta == -1 && colDelta == 0 && !destination.isOccupied()) {
            return true; // Move forward
        }
        // First move: move forward two squares if no pieces are in the way
        else if (rowDelta == -2 && colDelta == 0 && this.position.getRow() == 6 &&
            !destination.isOccupied() && !board.getSquare(this.position.getRow() - 1,
                this.position.getColumn()).isOccupied()) {
            return true; // First move
        }
        // Diagonal attack: move one square diagonally to capture an opponent's piece
        else if (rowDelta == -1 && colDelta == 1 && destination.isOccupied()
            && destination.getPiece().getColor() == Color.WHITE) {
            return true; // Diagonal attack
        }
    } else {
        // If the pawn is white, it moves upward the board
        // Move forward one square if the destination is not occupied
        if (rowDelta == 1 && colDelta == 0 && !destination.isOccupied()) {
            return true; // Move forward
        }
        // First move: move forward two squares if no pieces are in the way
        else if (rowDelta == 2 && colDelta == 0 && this.position.getRow() == 1 &&
            !destination.isOccupied() && !board.getSquare(this.position.getRow() + 1,
                this.position.getColumn()).isOccupied()) {
            return true; // First move
        }
        // Diagonal attack: move one square diagonally to capture an opponent's piece
        else if (rowDelta == 1 && colDelta == 1 && destination.isOccupied()
            && destination.getPiece().getColor() == Color.BLACK) {
            return true; // Diagonal attack
        }
    }
    // If none of the above conditions are met, the move is invalid
    return false;
}
```