Sergio Ávila Torres - 1636391
Carlos Molina Santamaria - 1631872
Dilluns 15:00-17:00
TETRIS
https://github.com/satorres8/TQS_PLAB_422

Funcionalidad: Manejar el movimiento de Tetrominos (izquierda, derecha y abajo).
Localización: `GameController.java`, clase `GameController`, métodos `keyPressed(KeyEvent e)`, `keyReleased(KeyEvent e)`.
Test:

- `GameControllerTest.java`, clase `GameControllerTest`, método `testMoveTetrominoLeft`.
  Descripción del test: Caja negra. Evalúa particiones equivalentes para verificar que el Tetromino se mueve a la izquierda.
- `GameControllerTest.java`, clase `GameControllerTest`, método `testMoveTetrominoRight`.
  Descripción del test: Caja negra. Evalúa particiones equivalentes para verificar que el Tetromino se mueve a la derecha.
- `GameControllerTest.java`, clase `GameControllerTest`, método `testMoveTetrominoDown`.
  Descripción del test: Caja negra. Evalúa particiones equivalentes para verificar que el Tetromino desciende correctamente.

Funcionalidad: Controlar la rotación de Tetrominos.
Localización: `GameController.java`, clase `GameController`, método `keyPressed(KeyEvent e)`.
Test:

- `GameControllerTest.java`, clase `GameControllerTest`, método `testRotateTetromino`.

Descripción del test: Caja negra. Evalúa particiones equivalentes para verificar que la forma del Tetromino cambia al rotar.

Funcionalidad: Gestionar el movimiento continuo y su detención.
Localización: `GameController.java`, clase `GameController`, métodos `keyPressed(KeyEvent e)`, `keyReleased(KeyEvent e)`.
Test:

- `GameControllerTest.java`, clase `GameControllerTest`, método `testContinuousDownMovement`.
  Descripción del test: Caja blanca. Evalúa bucles simples para verificar que el movimiento hacia abajo es continuo mientras la tecla está presionada.
- `GameControllerTest.java`, clase `GameControllerTest`, método `testKeyReleasedStopsContinuousMovement`.
  Descripción del test: Caja negra. Evalúa valores límite para garantizar que el movimiento continuo se detiene al liberar la tecla.

Funcionalidad: Prevenir que el Tetromino cruce los límites del tablero (izquierda y derecha).
Localización: `GameController.java`, clase `GameController`, método `keyPressed(KeyEvent e)`.
Test:

- `GameControllerTest.java`, clase `GameControllerTest`, método `testBoundaryLeftMovement`.
  Descripción del test: Caja negra. Evalúa valores límite para garantizar que el Tetromino no cruza el límite izquierdo.
- `GameControllerTest.java`, clase `GameControllerTest`, método `testBoundaryRightMovement`.
  Descripción del test: Caja negra. Evalúa valores límite para garantizar que el Tetromino no cruza el límite derecho.

Funcionalidad: Manejar combinaciones de teclas y pulsaciones rápidas.
Localización: `GameController.java`, clase `GameController`, método `processKeyQueue()`.
Test:

- `GameControllerTest.java`, clase `GameControllerTest`, método `testPairwiseKeyCombinations`.
  Descripción del test: Caja negra. Evalúa Pairwise Testing para verificar combinaciones como izquierda + abajo o derecha + rotación.
- `GameControllerTest.java`, clase `GameControllerTest`, método `testRapidKeyPresses`.
  Descripción del test: Caja negra. Evalúa valores límite para garantizar que las pulsaciones rápidas de teclas afectan correctamente al Tetromino.

Funcionalidad: Procesar múltiples teclas en la cola de eventos.
Localización: `GameController.java`, clase `GameController`, método `processKeyQueue()`.
Test:

- `GameControllerTest.java`, clase `GameControllerTest`, método `testProcessKeyQueueWithMultipleKeys`.
  Descripción del test: Caja blanca. Evalúa Path Coverage para garantizar que la cola de teclas se procesa correctamente.
- `GameControllerTest.java`, clase `GameControllerTest`, método `testMultipleKeysProcessing`.
  Descripción del test: Caja blanca. Evalúa bucles anidados para verificar que el estado del tablero es válido tras procesar múltiples teclas.

Funcionalidad: Verificar la interacción entre el controlador y el tablero al mover el Tetromino hacia la izquierda.
Localización: `GameController.java`, clase `GameController`, método `keyPressed(KeyEvent e)`.
Test:

- `GameControllerTestWithMocks.java`, clase `GameControllerTestWithMocks`, método `testMoveTetrominoLeft_MockGameBoard`.
  Descripción del test: Caja negra con mocks (Mockito). Evalúa la interacción del controlador con el método `moveTetrominoLeft()` de `GameBoard` y la invocación del callback de repintado.

Funcionalidad: Manejar teclas no válidas para prevenir acciones indebidas.
Localización: `GameController.java`, clase `GameController`, método `keyPressed(KeyEvent e)`.
Test:

- `GameControllerTestWithMocks.java`, clase `GameControllerTestWithMocks`, método `testInvalidKeyPress`.
  Descripción del test: Caja negra con mocks (Mockito). Evalúa robustez y manejo de entradas inválidas asegurando que no se realizan acciones en `GameBoard` para teclas no válidas.

Funcionalidad: Inicializar el tablero con dimensiones válidas.
Localización: `GameBoard.java`, clase `GameBoard`, constructor `GameBoard(int rows, int cols)`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testGameBoardInitialization`.
  Descripción del test: Caja negra. Evalúa particiones equivalentes para verificar que el tablero se inicializa correctamente con las dimensiones dadas.

Funcionalidad: Manejar dimensiones de tablero inválidas.
Localización: `GameBoard.java`, clase `GameBoard`, constructor `GameBoard(int rows, int cols)`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testInvalidGameBoardDimensions`.
  Descripción del test: Caja negra. Evalúa valores límite para garantizar que se lanza una excepción al inicializar un tablero con dimensiones inválidas.

Funcionalidad: Consultar el valor de celdas en el tablero.
Localización: `GameBoard.java`, clase `GameBoard`, método `getCell(int row, int col)`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testGetCell`.
  Descripción del test: Caja negra. Evalúa particiones equivalentes para verificar que las celdas del tablero contienen los valores iniciales correctos.

Funcionalidad: Manejar accesos a celdas fuera de los límites.
Localización: `GameBoard.java`, clase `GameBoard`, método `getCell(int row, int col)`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testInvalidCellAccess`.
  Descripción del test: Caja negra. Evalúa valores límite para garantizar que se lanza una excepción al intentar acceder a celdas fuera del rango válido.

Funcionalidad: Prevenir la asignación de valores inválidos a las celdas del tablero.
Localización: `GameBoard.java`, clase `GameBoard`, método `setCell(int row, int col, int value)`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testSetCellInvalidValue`.
  Descripción del test: Caja blanca. Evalúa cobertura de declaraciones (Statement Coverage) para verificar que no se pueden establecer valores negativos en las celdas.

Funcionalidad: Limpiar múltiples líneas completas y desplazar filas superiores.
Localización: `GameBoard.java`, clase `GameBoard`, método `clearCompleteLines()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testClearLineWithMultipleRows`.
  Descripción del test: Caja blanca. Evalúa bucles anidados para verificar que las filas superiores se desplazan correctamente al limpiar múltiples líneas.

Funcionalidad: Detectar colisiones entre el Tetromino y celdas ocupadas del tablero.
Localización: `GameBoard.java`, clase `GameBoard`, método `canMove(int newX, int newY, int[][] shape)`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testCanMoveCollision`.
  Descripción del test: Caja blanca. Evalúa cobertura de declaraciones para garantizar que el método detecta correctamente colisiones con celdas ocupadas.

Funcionalidad: Mover el Tetromino hacia abajo hasta que alcance el límite inferior del tablero.
Localización: `GameBoard.java`, clase `GameBoard`, método `moveTetrominoDown()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testMoveTetrominoDown`.
  Descripción del test: Caja blanca. Evalúa bucles simples para verificar que el Tetromino desciende hasta el límite inferior antes de generar uno nuevo.

Funcionalidad: Permitir el movimiento lateral del Tetromino.
Localización: `GameBoard.java`, clase `GameBoard`, métodos `moveTetrominoLeft()` y `moveTetrominoRight()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testMoveTetrominoLeftRight`.
  Descripción del test: Caja negra. Evalúa particiones equivalentes para garantizar que el Tetromino se mueve correctamente hacia la izquierda y la derecha.

Funcionalidad: Controlar la rotación del Tetromino en el tablero.
Localización: `GameBoard.java`, clase `GameBoard`, método `rotateTetromino()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testRotateTetromino`.
  Descripción del test: Caja negra. Evalúa particiones equivalentes para garantizar que la rotación del Tetromino cambia su forma correctamente.

Funcionalidad: Incrementar el puntaje tras limpiar líneas completas.
Localización: `GameBoard.java`, clase `GameBoard`, método `clearCompleteLines()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testScoreAfterClearingLines`.
  Descripción del test: Caja negra. Evalúa particiones equivalentes para verificar que el puntaje aumenta correctamente tras limpiar líneas.

Funcionalidad: Limpiar múltiples líneas completas del tablero y actualizar el puntaje.
Localización: `GameBoard.java`, clase `GameBoard`, método `clearCompleteLines()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testClearMultipleCompleteLines`.
  Descripción del test: Caja blanca. Evalúa bucles anidados para verificar que el puntaje se incrementa correctamente al limpiar varias líneas consecutivas.

Funcionalidad: Generar un nuevo Tetromino tras colocar el actual.
Localización: `GameBoard.java`, clase `GameBoard`, método `spawnTetromino()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testSpawnNewTetromino`.

Descripción del test: Caja negra. Evalúa valores límite para garantizar que un nuevo Tetromino se genera tras colocar el actual.

Funcionalidad: Prevenir que el Tetromino cruce los límites del tablero.
Localización: `GameBoard.java`, clase `GameBoard`, métodos `moveTetrominoLeft()` y `moveTetrominoRight()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testBoundaryMovements`.
  Descripción del test: Caja negra. Evalúa valores límite para garantizar que el Tetromino no puede moverse fuera del tablero por los bordes.

Funcionalidad: Detectar colisiones complejas entre el Tetromino y otras piezas en el tablero.
Localización: `GameBoard.java`, clase `GameBoard`, método `canMove(int newX, int newY, int[][] shape)`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testComplexCollisions`.
  Descripción del test: Caja blanca. Evalúa cobertura de declaraciones para garantizar que las colisiones complejas se manejan correctamente.

Funcionalidad: Validar el estado del tablero tras colocar un Tetromino.
Localización: `GameBoard.java`, clase `GameBoard`, método `placeTetromino()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testBoardStateAfterPlacement`.
  Descripción del test: Caja blanca. Evalúa Path Coverage para garantizar que el tablero mantiene un estado válido tras colocar un Tetromino.

Funcionalidad: Verificar combinaciones de posiciones y formas para Tetrominos en el tablero.
Localización: `GameBoard.java`, clase `GameBoard`, método `canMove(int newX, int newY, int[][] shape)`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testCanMovePairwise`.
  Descripción del test: Caja blanca. Evalúa Pairwise Testing para verificar combinaciones de posiciones y formas del Tetromino.

Funcionalidad: Colocar Tetrominos en el tablero y validar todas las rutas posibles.
Localización: `GameBoard.java`, clase `GameBoard`, método `placeTetromino()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testPlaceTetrominoPathCoverage`.
  Descripción del test: Caja blanca. Evalúa Path Coverage para validar todas las rutas del método `placeTetromino`.

Funcionalidad: Usar un Tetromino mockeado para probar la colocación en el tablero.
Localización: `GameBoard.java`, clase `GameBoard`, método `placeTetromino()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testPlaceTetrominoWithMockedTetromino`.
  Descripción del test: Caja blanca con mocks. Asegura que la colocación del Tetromino en el tablero se realiza correctamente usando un MockObject.

Funcionalidad: Detectar colisiones usando un Tetromino mockeado.
Localización: `GameBoard.java`, clase `GameBoard`, método `canMove(int newX, int newY, int[][] shape)`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testCollisionDetectionWithMockedTetromino`.
  Descripción del test: Caja blanca con MockObjects manuales. Verifica la detección de colisiones al mover un Tetromino mockeado en el tablero.

Funcionalidad: Manejar el estado de fin de juego cuando no hay espacio para un nuevo Tetromino.
Localización: `GameBoard.java`, clase `GameBoard`, método `spawnTetromino()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testGameOverWhenNoSpaceForNewTetromino`.
  Descripción del test: Caja blanca con mocks. Verifica que se lanza una excepción cuando no hay espacio para generar un nuevo Tetromino.

Funcionalidad: Gestionar el estado de juego cuando un Tetromino no puede moverse ni generarse.
Localización: `GameBoard.java`, clase `GameBoard`, método `moveTetrominoDown()`.
Test:

- `GameBoardTest.java`, clase `GameBoardTest`, método `testTetrominoCannotMove_ExpectException`.
  Descripción del test: Caja blanca con mocks. Evalúa el manejo de excepciones al intentar mover un Tetromino que no puede desplazarse ni generar uno nuevo.

Funcionalidad: Inicializar Tetrominos con tipos válidos.
Localización: `Tetromino.java`, clase `Tetromino`, constructor `Tetromino(TetrominoType type)`.
Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testTetrominoInitializationValidTypes`.
  Descripción del test: Caja negra. Evalúa particiones equivalentes para garantizar que cada tipo de Tetromino (I, O, T, S, Z, J, L) tiene la forma correcta tras la inicialización.

Funcionalidad: Manejar inicialización con tipos de Tetromino no válidos.
Localización: `Tetromino.java`, clase `Tetromino`, constructor `Tetromino(TetrominoType type)`.
Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testTetrominoInitializationInvalidType`.
  Descripción del test: Caja negra. Evalúa valores límite para garantizar que se lanza una excepción al intentar inicializar un Tetromino con un tipo nulo.

Funcionalidad: Validar rotaciones de todos los Tetrominos.
Localización: `Tetromino.java`, clase `Tetromino`, método `rotate()`.
Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testAllRotationsForAllTetrominoes`.
  Descripción del test: Caja negra. Evalúa cobertura de rutas para garantizar que todas las rotaciones de los Tetrominos coincidan con las formas esperadas.

Funcionalidad: Contar las celdas ocupadas en un Tetromino.
Localización: `Tetromino.java`, clase `Tetromino`, método `getOccupiedCells()`.
Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testGetOccupiedCells`.
  Descripción del test: Caja negra. Evalúa particiones equivalentes para garantizar que el número de celdas ocupadas es correcto para cada tipo de Tetromino.

Funcionalidad: Evitar cambios de forma al rotar piezas de tipo O.
Localización: `Tetromino.java`, clase `Tetromino`, método `rotate()`.
Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testRotateTetrominoO`.
  Descripción del test: Caja blanca. Evalúa cobertura de decisiones para garantizar que las piezas tipo O mantienen su forma tras la rotación.

Funcionalidad: Garantizar cambios de forma al rotar piezas que no son de tipo O.
Localización: `Tetromino.java`, clase `Tetromino`, método `rotate()`.
Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testRotateTetrominoNonO`.
  Descripción del test: Caja blanca. Evalúa cobertura de decisiones para garantizar que los Tetrominos que no son de tipo O cambian su forma correctamente tras la rotación.

Funcionalidad: Validar la actualización de formas en Tetrominos.
Localización: `Tetromino.java`, clase `Tetromino`, método `setShape(int[][] shape)`.
Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testSetShapePairwise`.
  Descripción del test: Caja negra. Evalúa Pairwise Testing para verificar que el método `setShape` maneja correctamente entradas válidas e inválidas.

Funcionalidad: Manejar inicialización con tipos válidos y prevenir excepciones innecesarias.
Localización: `Tetromino.java`, clase `Tetromino`, constructor `Tetromino(TetrominoType type)`.
Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testConstructorWithValidType`.
  Descripción del test: Caja blanca. Evalúa cobertura de decisiones para garantizar que no se lanza una excepción al inicializar un Tetromino con un tipo válido.

Funcionalidad: Prevenir inicialización con tipos nulos.
Localización: `Tetromino.java`, clase `Tetromino`, constructor `Tetromino(TetrominoType type)`.
Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testConstructorWithNullType`.
  Descripción del test: Caja blanca. Evalúa cobertura de decisiones para garantizar que se lanza una excepción cuando el tipo del Tetromino es nulo.

Funcionalidad: Contar celdas ocupadas en Tetrominos con formas vacías.
Localización: `Tetromino.java`, clase `Tetromino`, método `getOccupiedCells()`.
Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testGetOccupiedCellsEmptyShape`.
  Descripción del test: Caja blanca. Evalúa pruebas de bucles para verificar que el método retorna 0 cuando la forma del Tetromino es vacía.

Funcionalidad: Contar celdas ocupadas en Tetrominos con formas mixtas.
Localización: `Tetromino.java`, clase `Tetromino`, método `getOccupiedCells()`.
Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testGetOccupiedCellsMixedShape`.
  Descripción del test: Caja blanca. Evalúa bucles anidados para garantizar que el método cuenta correctamente las celdas ocupadas en formas mixtas.

Funcionalidad: Verificar el comportamiento del método `rotate` utilizando un Spy.

Localización: `Tetromino.java`, clase `Tetromino`, método `rotate()`.

Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testRotateTetrominoWithSpy`.
  Descripción del test: Caja blanca con Spy. Evalúa aislamiento y cobertura para garantizar que el método `rotate` actualiza correctamente la forma del Tetromino.

Funcionalidad: Manejar excepciones en `getOccupiedCells` cuando la forma del Tetromino es nula.

Localización: `Tetromino.java`, clase `Tetromino`, método `getOccupiedCells()`.

Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testGetOccupiedCellsWithNullShape`.
  Descripción del test: Caja blanca con mocks. Evalúa robustez para verificar que se lanza una excepción cuando la forma del Tetromino es nula.

Funcionalidad: Contar correctamente celdas ocupadas en Tetrominos con formas irregulares.

Localización: `Tetromino.java`, clase `Tetromino`, método `getOccupiedCells()`.

Test:

- `TetrominoTest.java`, clase `TetrominoTest`, método `testTetrominoWithIrregularShape`.
  Descripción del test: Caja blanca con MockObjects manuales. Evalúa robustez para garantizar que el método cuenta las celdas ocupadas en formas no estándar.

***Las múltiples versiones tanto de las funcionalidades como tests del código se pueden ver a través de los commits del repositorio de GitHub**