```
package hw3;

import static api.Direction.*;

import java.util.ArrayList;

import api.Cell;
import api.Direction;
import api.Move;
import api.Orientation;

/**
 * Represents a board in the Block Slider game. A board contains a 2D grid of
 * cells and a list of blocks that slide over the cells.
 * @author jcluse
 */
public class Board {
      /**
       * 2D array of cells, the indexes signify (row, column) with (0, 0)
representing
       * the upper-left cornner of the board.
       */
      private Cell[][] grid;

      /**
       * A list of blocks that are positioned on the board.
       */
      private ArrayList<Block> blocks;

      /**
       * A list of moves that have been made in order to get to the current
position
       * of blocks on the board.
       */
      private ArrayList<Move> moveHistory;

      private int moveCount;

      private boolean over;

      private Block grabbedBlock;

      private Cell grabbedCell;

      private boolean canPlaceBlock = false;

      /**
       * Constructs a new board from a given 2D array of cells and list of blocks.
The
       * cells of the grid should be updated to indicate which cells have blocks
       * placed over them (i.e., setBlock() method of Cell). The move history
should
       * be initialized as empty.
       *
       * @param grid   a 2D array of cells which is expected to be a rectangular
shape
       * @param blocks list of blocks already containing row-column position which
       *               should be placed on the board
       */
      public Board(Cell[][] grid, ArrayList<Block> blocks) {
            this.grid = grid;
            this.blocks = blocks;
            int row = 0;
            int col = 0;
            int length = 0;
            Orientation ori = null;

            for(int i = 0; i < blocks.size(); i++) { //creates new blocks and
places them on the grid


                  row = blocks.get(i).getFirstRow();
                  col = blocks.get(i).getFirstCol();
                  length = blocks.get(i).getLength();
                  ori = blocks.get(i).getOrientation();

                  Block block = new Block(row,col,length,ori);

                  if(block.getOrientation() == Orientation.HORIZONTAL) {

                        for(int j = 0; j < length; j++) { //horizontal blocks

                              this.grid[row][col + j].setBlock(block);

                        }

                  }

                  else { //vertical blocks

                        for(int j = 0; j < length; j++) {

                              this.grid[row + j][col].setBlock(block);

                        }

                  }




                  //go through blocks using block.getfirst methods
                  //place the blocks onto the grid using setblock method
            }



            moveCount = 0;
            over = false;
            moveHistory = new ArrayList<Move>();
            reset();

      }

      /**
       * Constructs a new board from a given 2D array of String descriptions.
       * <p>
       * DO NOT MODIFY THIS CONSTRUCTOR
       *
       * @param desc 2D array of descriptions
       */
      public Board(String[][] desc) {
            this(GridUtil.createGrid(desc), GridUtil.findBlocks(desc));
      }

      /**
       * Models the user grabbing a block over the given row and column. The
purpose
       * of grabbing a block is for the user to be able to drag the block to a new
       * position, which is performed by calling moveGrabbedBlock(). This method
       * records two things: the block that has been grabbed and the cell at which
it
       * was grabbed.
       *
       * @param row row to grab the block from
       * @param col column to grab the block from
       */
      public void grabBlockAtCell(int row, int col) {
            releaseBlock();
            grabbedCell = grid[row][col];
            if (grabbedCell.hasBlock()) {

                  grabbedBlock = grabbedCell.getBlock();

            }
      }

      /**
       * Set the currently grabbed block to null.
       */
      public void releaseBlock() {
            grabbedBlock = null;
      }

      /**
       * Returns the currently grabbed block.
       *
       * @return the current block
       */
      public Block getGrabbedBlock() {

            return grabbedBlock;
      }

      /**
       * Returns the currently grabbed cell.
       *
       * @return the current cell
       */
      public Cell getGrabbedCell() {

            return grabbedCell;
      }

      /**
       * Returns true if the cell at the given row and column is available for a
block
```

```
 * to be placed over it. Blocks can only be placed over floors and exits. A
 * block cannot be placed over a cell that is occupied by another block.
 *
 * @param row row location of the cell
 * @param col column location of the cell
 * @return true if the cell is available for a block, otherwise false
 */
public boolean canPlaceBlock(int row, int col) {

      canPlaceBlock = false;

      if(!grid[row][col].isWall()) {
            if(!grid[row][col].hasBlock()) {

            canPlaceBlock = true;

            }
      }
      return canPlaceBlock;
}

/**
 * Returns the number of moves made so far in the game.
 *
 * @return the number of moves
 */
public int getMoveCount() {

      return moveCount;
}

/**
 * Returns the number of rows of the board.
 *
 * @return number of rows
 */
public int getRowSize() {
      // TODO
      return grid.length;
}

/**
 * Returns the number of columns of the board.
 *
 * @return number of columns
 */
public int getColSize() {

      return grid[0].length;
}

/**
 * Returns the cell located at a given row and column.
 *
 * @param row the given row
 * @param col the given column
 * @return the cell at the specified location
 */
public Cell getCell(int row, int col) {
```

```
            return grid[row][col];
      }

      /**
       * Returns a list of all blocks on the board.
       * 
       * @return a list of all blocks
       */
      public ArrayList<Block> getBlocks() {

            return blocks;
      }

      /**
       * Returns true if the player has completed the puzzle by positioning a block
       * over an exit, false otherwise.
       * 
       * @return true if the game is over
       */
      public boolean isGameOver() {

            return over;
      }
      /**
       * Helper method used to determine the index of the grabbed
       * block within blocks array
       * 
       * @return index of grabbed block
       */
      private int currentblockIndex() {

            int row = grabbedBlock.getFirstRow();

            int col = grabbedBlock.getFirstCol();

            int blockNum = 0;

            for(int i = 0; i < blocks.size(); i++) {

                  if(blocks.get(i).getFirstRow() == row
                              && blocks.get(i).getFirstCol() == col) {

                        blockNum = i;
                  }
            }

            return blockNum;
      }

      /**
       * Moves the currently grabbed block by one cell in the given direction. A
       * horizontal block is only allowed to move right and left and a vertical
block
       * is only allowed to move up and down. A block can only move over a cell
that
       * is a floor or exit and is not already occupied by another block. The
method
       * does nothing under any of the following conditions:
```

```
       * <ul>
       * <li>The game is over.</li>
       * <li>No block is currently grabbed by the user.</li>
       * <li>A block is currently grabbed by the user, but the block is not allowed
to
       * move in the given direction.</li>
       * </ul>
       * If none of the above conditions are meet, the method does the following:
       * <ul>
       * <li>Moves the block object by calling its move method.</li>
       * <li>Sets the block for the grid cell that the block is being moved
into.</li>
       * <li>For the grid cell that the block is being moved out of, sets the block
to
       * null.</li>
       * <li>Moves the currently grabbed cell by one cell in the same moved
direction.
       * The purpose of this is to make the currently grabbed cell move with the
block
       * as it is being dragged by the user.</li>
       * <li>Adds the move to the end of the moveHistory list.</li>
       * <li>Increment the count of total moves made in the game.</li>
       * </ul>
       *
       * @param dir the direction to move
       */
      public void moveGrabbedBlock(Direction dir) {

            int num = 0;

            if(dir == LEFT || dir == UP) {
                  num = -1;
            }
            else {
                  num = 1;
            }

            if(!over && grabbedBlock != null) {

                  int row = grabbedBlock.getFirstRow();

                  int col = grabbedBlock.getFirstCol();

                  int originalRow = grabbedBlock.getFirstRow();

                  int originalCol = grabbedBlock.getFirstCol();

                  int lastCol = originalCol + grabbedBlock.getLength() - 1;

                  int lastRow = originalRow + grabbedBlock.getLength() - 1;

                  int grabbedBlockIndex = currentblockIndex();

                  if((grabbedBlock.getOrientation() == Orientation.HORIZONTAL))
{ //HORIZONTAL MOVE

                        if(dir == RIGHT) { //Right possibility

                              if(canPlaceBlock(row,col + grabbedBlock.getLength()))
```

```
{
                                    grabbedBlock.move(dir);

      blocks.get(grabbedBlockIndex).setFirstCol(originalCol + num);
                                    col = col + num;

                                    this.grid[row][originalCol +
grabbedBlock.getLength()].setBlock(grabbedBlock);
                                    this.grid[row][originalCol].clearBlock();
                                    grabbedCell = this.grid[grabbedCell.getRow()]
[grabbedCell.getCol() + num];
                                    moveHistory.add(new Move(grabbedBlock, dir));
                                    moveCount++;

                              }

                        }

                        else if(dir == LEFT) { //Left possibility

                              if(canPlaceBlock(row,col + num)){

                                    grabbedBlock.move(dir);

      blocks.get(grabbedBlockIndex).setFirstCol(originalCol + num);
                                    col = col + num;

                                    this.grid[row][originalCol +
num].setBlock(grabbedBlock);
                                    this.grid[row][lastCol].clearBlock();
                                    grabbedCell = this.grid[grabbedCell.getRow()]
[grabbedCell.getCol() + num];
                                    moveHistory.add(new Move(grabbedBlock, dir));
                                    moveCount++;

                              }
                        }


                        if(this.grid[row][col + grabbedBlock.getLength() -
1].isExit()) { //checks if game is over
                              over = true;

                        }

                  }

                  else if ((grabbedBlock.getOrientation() == Orientation.VERTICAL))
{ //VERTICAL MOVE

                        if(dir == DOWN) { //Up possibility

                              if((canPlaceBlock(row +
grabbedBlock.getLength(),col))){

                                    grabbedBlock.move(dir);
                                    row = row + num;
      blocks.get(grabbedBlockIndex).setFirstRow(originalRow + num);

                                    this.grid[originalRow + 
grabbedBlock.getLength()][col].setBlock(grabbedBlock);
                                    this.grid[originalRow][col].clearBlock();
                                    grabbedCell = this.grid[grabbedCell.getRow() + 
num][grabbedCell.getCol()];
                                    moveHistory.add(new Move(grabbedBlock, dir));
                                    moveCount++;

                              }

                        }

                        else if(dir == UP) { //Down possibility

                              if(canPlaceBlock(row + num, col)){

                                    grabbedBlock.move(dir);
                                    row = row + num;

      blocks.get(grabbedBlockIndex).setFirstRow(originalRow + num);

                                    this.grid[originalRow + num]
[col].setBlock(grabbedBlock);
                                    this.grid[lastRow][col].clearBlock();
                                    grabbedCell = this.grid[grabbedCell.getRow() + 
num][grabbedCell.getCol()];
                                    moveHistory.add(new Move(grabbedBlock, dir));
                                    moveCount++;

                              }

                        }




                        if(this.grid[row + grabbedBlock.getLength() - 1]
[col].isExit()) {
                              over = true;
                        }

                        }
            }

      }

      /**
       * Resets the state of the game back to the start, which includes the move
       * count, the move history, and whether the game is over. The method calls 
the
       * reset method of each block object. It also updates each grid cells by 
calling
       * their setBlock method to either set a block if one is located over the 
cell
       * or set null if no block is located over the cell.
       */
      public void reset() {
            for(int row = 0; row < this.grid.length; row++) { //clears all blocks

                  for(int col = 0; col < this.grid[0].length; col++) {

                        this.grid[row][col].clearBlock();

                  }
            }

            int row = 0;
            int col = 0;
            int length = 0;
            Orientation ori = null;

            for(int i = 0; i < blocks.size(); i++) {

                  blocks.get(i).reset();

                  row = blocks.get(i).getFirstRow();
                  col = blocks.get(i).getFirstCol();
                  length = blocks.get(i).getLength();
                  ori = blocks.get(i).getOrientation();

                  Block block = new Block(row,col,length,ori);

                  if(block.getOrientation() == Orientation.HORIZONTAL) {

                        for(int j = 0; j < length; j++) {

                              this.grid[row][col + j].setBlock(block);

                        }

                  }

                  else {

                        for(int j = 0; j < length; j++) {

                              this.grid[row + j][col].setBlock(block);

                        }

                  }
            }

            moveCount = 0;
            over = false;
            moveHistory = new ArrayList<Move>();
      }

      /**
       * Returns a list of all legal moves that can be made by any block on the
       * current board. If the game is over there are no legal moves.
       *
       * @return a list of legal moves
```

```
       */
      public ArrayList<Move> getAllPossibleMoves() {

            ArrayList<Move> possibleMoves = new ArrayList<Move>();

            for(int i = 0; i < blocks.size(); i++) {

                  if(blocks.get(i).getOrientation() == Orientation.HORIZONTAL) {

                        if(canPlaceBlock(blocks.get(i).getFirstRow(),
blocks.get(i).getFirstCol() + 2)) {

                        possibleMoves.add(new Move(blocks.get(i), RIGHT));

                        }

                        else if(canPlaceBlock(blocks.get(i).getFirstRow(),
blocks.get(i).getFirstCol() - 1)) {

                        possibleMoves.add(new Move(blocks.get(i), LEFT));

                        }

                  }

                  if(blocks.get(i).getOrientation() == Orientation.VERTICAL) {

                        if(canPlaceBlock(blocks.get(i).getFirstRow() - 1,
blocks.get(i).getFirstCol())) {

                        possibleMoves.add(new Move(blocks.get(i), UP));

                        }

                        else if(canPlaceBlock(blocks.get(i).getFirstRow() + 1,
blocks.get(i).getFirstCol())) {

                        possibleMoves.add(new Move(blocks.get(i), DOWN));

                        }

                  }

            }
            return possibleMoves;
      }

      /**
       * Gets the list of all moves performed to get to the current position on the
       * board.
       *
       * @return a list of moves performed to get to the current position
       */
      public ArrayList<Move> getMoveHistory() {
            return moveHistory;
      }

      /**
       * EXTRA CREDIT 5 POINTS
       * <p>
       * This method is only used by the Solver.
       * <p>
       * Undo the previous move. The method gets the last move on the moveHistory
list
       * and performs the opposite actions of that move, which are the following:
       * <ul>
       * <li>grabs the moved block and calls moveGrabbedBlock passing the opposite
       * direction</li>
       * <li>decreases the total move count by two to undo the effect of calling
       * moveGrabbedBlock twice</li>
       * <li>if required, sets is game over to false</li>
       * <li>removes the move from the moveHistory list</li>
       * </ul>
       * If the moveHistory list is empty this method does nothing.
       */
      public void undoMove() {
            // TODO
      }

      @Override
      public String toString() {
            StringBuffer buff = new StringBuffer();
            boolean first = true;
            for (Cell row[] : grid) {
                  if (!first) {
                        buff.append("\n");
                  } else {
                        first = false;
                  }
                  for (Cell cell : row) {
                        buff.append(cell.toString());
                        buff.append(" ");
                  }
            }
            return buff.toString();
      }
}
```