```
package hw3;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.Map;

import api.Cell;
import api.Move;

/**
 * A puzzle solver for the the Block Slider game.
 * <p>
 * THE ONLY METHOD YOU ARE CHANGING IN THIS CLASS IS solve().
 */
public class Solver {
	/**
	 * Maximum number of moves allowed in the search.
	 */
	private int maxMoves;

	/**
	 * Associates a string representation of a grid with the move count required
to
	 * reach that grid layout.
	 */
	private Map<String, Integer> seen = new HashMap<String, Integer>();

	/**
	 * All solutions found in this search.
	 */
	private ArrayList<ArrayList<Move>> solutions = new
ArrayList<ArrayList<Move>>();


	private boolean isgameOver;

	private Cell[][] grid;

	/**
	 * Constructs a solver with the given maximum number of moves.
	 *
	 * @param givenMaxMoves maximum number of moves
	 */
	public Solver(int givenMaxMoves) {
		maxMoves = givenMaxMoves;
		solutions = new ArrayList<ArrayList<Move>>();
	}

	/**
	 * Returns all solutions found in the search. Each solution is a list of
moves.
	 *
	 * @return list of all solutions
	 */
	public ArrayList<ArrayList<Move>> getSolutions() {
		return solutions;
	}

	/**
```

```
	 * Prints all solutions found in the search.
	 */
	public void printSolutions() {
		for (ArrayList<Move> moves : solutions) {
			System.out.println("Solution:");
			for (Move move : moves) {
				System.out.println(move);
			}
			System.out.println();
		}
	}

	/**
	 * EXTRA CREDIT 15 POINTS
	 * <p>
	 * Recursively search for solutions to the given board instance according to
the
	 * algorithm described in the assignment pdf. This method does not return
	 * anything its purpose is to update the instance variable solutions with
every
	 * solution found.
	 *
	 * @param board any instance of Board
	 */
	public void solve(Board board) {
		// TODO
		int i = 0;
		int j = 0;

	//base case
		if(board.getMoveCount() >= maxMoves) {
			return;
		}else if(board.isGameOver() == true) {
			board.getMoveHistory();
		}else if(grid[i][j] == grid[i - 1][j - 1]) {
			if(board.getMoveCount() >= board.getMoveCount()-1) {
				return;
			}
		}else {
			board.getMoveCount();
		}

		//getting the list of all possible moves
		board.getAllPossibleMoves();


		//for each possible move
		for(i = 0; i < board.getAllPossibleMoves().size(); i++) {

			//grab the block to move
			board.grabBlockAtCell(i, j);
			board.moveGrabbedBlock(null);
			solve(board);
			board.undoMove();
		}


	}
}
```