**Problem 7** [9 points]
The point of this problem was to implement the A* algorithm for solving the 8-puzzle problem. Since the algorithm was never fully described in the book -- it was necessary to build upon a number of techniques from different parts of the book: avoiding repeated states (AIMA 3.5), best-first search and heuristic search (AIMA 4.1). The main challenge was ensuring correct book keeping in such a way that guarantees that the best solution is the first one found and that states are not revisited unnecessarily. That's where most of the mistakes were made.

*Maintaining the fringe:* when adding new boards to the fringe, it is necessary to check if a board like the one that we are about to add already exists in the fringe but with a smaller cost estimate. In such cases, we ignore the new board and keep the old one. I usually subtracted 3 points for this mistake.

*Maintaining the closed list:* exactly the same applies to the closed list. The difference is, that in this particular case it was not necessary to perform this check because the heuristics were consistent (AIMA p.99). Yet, if you were implementing A* in a general case, this is a case that you would have to take care of. Since very few people got it right, I gave up to two extra points to the few diligent ones. I would have also awarded the extra points if you argued that the extra book keeping was not necessary because our particular heuristics did not require it. I don't think anybody has caught this bit in the book, though.

A sample implementation of the algorithm follows:

```
/**
 * The A* search method
 *
 * @param state the initial state of the board
 * @return the final board
 */
public Board search(Board startState) {
    // initialize open list
    OpenList openList = new OpenList();
    // set the cost estiamte for the start state and add it to the open list
    startState.setCostEstimate(getCostEstimate(startState));
    openList.add(startState);
    // initialize closed list
    ClosedList closedList = new ClosedList();

    // keep getting contents from the open list until it is empty
    while (!openList.isEmpty()) {
        // get the best board from the open list
        Board curBoard = openList.removeFirst();
        System.out.println("Exploring \n" + curBoard);
        // if it is in the goal configuration, return it as solution
        if (curBoard.equals(goalState))
            return curBoard;
        // get successors of the current board
        Iterator successors = curBoard.getSuccessors().iterator();
        // iterate over the successors
        while (successors.hasNext()) {
            Board curSuccessor = (Board) successors.next();
            // calculate cost estimate for the current successor
            curSuccessor.setCostEstimate(getCostEstimate(curSuccessor));
            // if there is a better instance of the successor in the open
            // list, skip to the next successor
            if (openList.contains(curSuccessor)
                && openList.getCostFor(curSuccessor)
                    < curSuccessor.getCostEstimate())
                continue;
            // if there is a better instance in the closed list, skip to
            // the next successor
            if (closedList.contains(curSuccessor)
                && closedList.getCostFor(curSuccessor)
                    < curSuccessor.getCostEstimate())
                continue;
            // remove the successor from the closed list if it's there
            closedList.remove(curSuccessor);
            // add the successor to the open list
            openList.add(curSuccessor);
        }
        // add the current node to the closed list
        closedList.add(curBoard);
    }
    // return null if no solution was found
    return null;
}
```

**Problem 8** [9 points]
The implementation of the two heuristics (misplaced tiles and Manhattan distance) was very easy and almost everybody got it right. The only mistake I saw was forgetting to exclude the blank tile from the calculations.

The point of the exercise was for you to see for yourself what huge difference a good heuristic can make. Anybody who demonstrated the relative differences on a reasonable set of 8-puzzle instances got full credit here. Many of you observed that some of the randomly generated boards were either unsolvable or took a very long time to solve. Most of you followed the instructions in the assignment's preamble and made reasonable assumptions and generated informative results. A few people implemented and documented functions for quickly deciding if a board is solvable -- extra credit was given there.