You should use the following as the goal state:

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>6</td>
<td>5</td>
</tr>
</tbody>
</table>

The following is a good start state to use:

<table>
<thead>
<tr>
<th>2</th>
<th>8</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>5</td>
</tr>
</tbody>
</table>

Note that the available operators are Down, Up, Right, Left. Given any board position, each of these operators is either unapplicable or applicable in one way. Depending on where the blank is, one can always figure out which operators are applicable. For example, in the start state given above, the operators Down, Right, and Left are all applicable.

I would recommend representing the board by three vectors, each of length 3. Each vector represents a row of the board. For example the start state from above is as follows:

```
Positions
0 1 2

Vec0 | 2 | 8 | 3 |

Vec1 | 1 | 6 | 4 |

Vec2 | 7 |   | 5 |
```

Note that we are using 0 to represent the blank. We can then say that the blank is in position (2,1) because it is in Vec2 and position 1.

It is probably useful to define a class Position which has a Row and a Column value.
public class Position {
    int Row;
    int Column;
}

You can define the methods GetRow, GetColumn.

Some Recommendations are as follows:

1. Start with a copy of the simplified version of TwoThreeState and create your own EightPuzzleState class.

2. For debugging purposes create your own driver. Only after you have debugged your state, try the search code.

3. You need to create the following Methods as part of EightPuzzleState.

   applyOperator Must make a copy of the state and then modify it in the appropriate way depending on operator. Find the blank position first and then make necessary moves.

   ValidOperators Depending on the position of the blank the function returns the list of valid operators. Not hard, but a little laborious. Test this function using your own driver.

   successors Easy, no change from TwoThree State

   costOf Easy, always 1

   isGoal Not difficult at all.

   h Only needed if you do the extra credit part.

   getValue Function takes as an argument a Position and returns the value in that position. Easy.

   setValue Function takes a value and a Position and then sets that position to that value. Easy.

   getBlankPos Function returns a Position, the location of the blank. Easy.

   toString You can take this out unless you plan to use it in your printing function.

4. You need to create a function to print out the result. This can be modeled after the one in TestSearch. You should print out each board position and the move used to get there. The board positions can easily be printed out as follows:
<table>
<thead>
<tr>
<th>2</th>
<th>8</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>1</td>
<td>7</td>
<td>5</td>
</tr>
</tbody>
</table>

5. For your Search driver program, you should modify TestSearch.

6. Elimination of duplicates should be done in SearchNode. In method expend, check to see if new state is identical to that of any parent states and if so, reject. Wait till you have implemented everything else before you work on this.