Our edited code, before and after.

Group:

- Karl Thomas Hauglid (karltha), Bachelor
- Jostein Hellerud (josteibh), Bachelor
- Simon Jespersen (simonje), Master

Here is our code that we edited in the assignment. First you will see the unedited code, and then the edited code.

Comments:

The first edit we did was giving each method a comment that explains what the method does. We will only show for one method as there is a lot of methods in the program.
Before:

```
private boolean trymove(int randomRow, int randomCol) {
    if(mines[randomRow][randomCol]){
        return false;
    }
    else{
        mines[randomRow][randomCol]=true;
        return true;
    }
}
```

After:

```
//check if it the move is valid
// @param randomRow random row
//@param randomCol random col
// @return returns false if it is. Sets the move too true and returns true if it not taken
private boolean trymove(int randomRow, int randomCol) {
    if(mines[randomRow][randomCol]){
        return false;
    }
    else{
        mines[randomRow][randomCol]=true;
        return true;
    }
}
```

As you can see that code now has a short explanation, for the method as well as parameters and return value.

Max complexity:

The next edit was lowering the max complexity. This was done in the method MineField.drawChar(). We extracted the two for loops that increased the count variable to a

separate method. We also removed some empty lines and unnecessary brackets, and changed the switch case.

Before:

```
private char drawChar(int row, int col) {
    int count=0;
    if(visible[row][col]){
        if(mines[row][col]) return '*';
        for(int irow=row-1;irow<=row+1;irow++){
            for(int icol=col-1;icol<=col+1;icol++){
                if(icol>=0&&icol<colMax&&irow>=0&&irow<rowMax){
                    if(mines[irow][icol]) count++;
                }
            }
        }
    }
    else{
        if(boom){
            return '-';
        }
        {


            return '?';
        }
    }
    switch(count){
    case 0:return '0';
    case 1:return '1';
    case 2:return '2';
    case 3:return '3';
    case 4:return '4';
    case 5:return '5';
    case 6:return '6';
    case 7:return '7';
    case 8:return '8';


    default:return 'X';
    }
}
```

After:

```
//show the current content of position given by row col
// @param row
//@param col
//@return returns what should be located under the postion in form of a char
private char drawChar(int row, int col) {
    int count=0;
    if(visible[row][col]){
        if(mines[row][col]) return '*';
        count = increaseCount(row, col);
    }else{
        if(boom){
            return '-';
        }
        return '?';
    }
    if(count < 9) return (char) (count+48);

    return 'X';
}
```

The extracted method:

```java
/**
*Check number of mines that is surrounding the current cell
*@param row the row number
*@param col the coloum number
*@return int the number of bombs surrounding current cell
*/
private int increaseCount(int row, int col) {
    int count = 0;
    for(int irow=row-1;irow<=row+1;irow++){
        for(int icol=col-1;icol<=col+1;icol++){
            if(icol>=0&&icol<colMax&&irow>=0&&irow<rowMax){
                if(mines[irow][icol]) count++;
            }
        }
    }
    return count;
}
```

Average depth:

The last edit we did was lowering the average depth. After evaluating Ranking.java we found several unecessary curly brackets, so this was the file we decided to edit.

Removing the unecessary brackets,
Before edit:

```java
else{
    name[last]=newName;{
        record[last]=result;{
            last++;
        }
    }
}
```

After:

```java
else{
    name[last]= newName;
    record[last]=result;
    last++;
}
```

This is only a snippet of the method recordName(), as there is no need to show the whole method.

We also changed sort() to get rid of unnecessary brackets to lower the avrage depth
Before:

```
private void sort(){
    if(last<2) return;
    boolean unsorted=true;
    while(unsorted){
        unsorted=false;
        for(int i=0;i<(last-1);i++){
            if(record[i+1]>record[i]){
                int swapR=record[i];{
                    record[i]=record[i+1];{
                        record[i+1]=swapR;
                        String swapN=name[i];
                        name[i]=name[i+1];
                        name[i+1]=swapN;
                        unsorted=true;
                    }
                }
            }
        }
    }
}
```

After:

```
private void sort(){
    if(last<2) return;
    boolean unsorted=true;
    while(unsorted){
        unsorted=false;
        for(int i=0;i<(last-1);i++){
            if(record[i+1]>record[i]){
                swap(i);
                unsorted=true;
            }
        }
    }
}
private void swap(int i){
    int swapR=record[i];
    record[i]=record[i+1];
    record[i+1]=swapR;
    String swapN=name[i];
    name[i]=name[i+1];
    name[i+1]=swapN;
}
```

We extracted the swapping algorithm to a separate method because there were a lot of statements at the deepest block level.