CS21120: Data Structures and Algorithm Analysis
Assignment 1 – Sudoku

dac46@aber.ac.uk
25/10/13

Description Of Solver Algorithms

Main loop:

1. Do 10 times:
    2. Update pencil marks for each cell
    3. Solve Pointing Pairs for Rows and Columns
    4. Update pencil marks for each cell
    5. Solve Naked Pairs for Rows, Columns and Squares
    6. Do 10 times:
        7. Update pencil marks for each cell
        8. Solve Hidden Singles for Rows, Columns and Squares
        9. Solve Naked Singles for Rows, Columns and Squares

Pointing Pairs:

1. For every block:
    2. For every possible number in a block (1-9):
        3. Get the number of times that number occurs in the block
        4. Check if all those occurrences are all in a different block together
        5. Remove all occurrences of the number from the pencil marks in the original block EXCEPT those that also appear in the second block

Naked Pairs:

1. For every block:
    2. Get every cell that has 2 pencil marks in
    3. For every possible pair of those:
        4. Check if they have the same set of pencil marks
        5. If they do:
            6. For each pencil mark they have the same:
                7. Remove all occurrences of the number from the pencil marks in the original block EXCEPT the two cells in the pair

Hidden Singles:

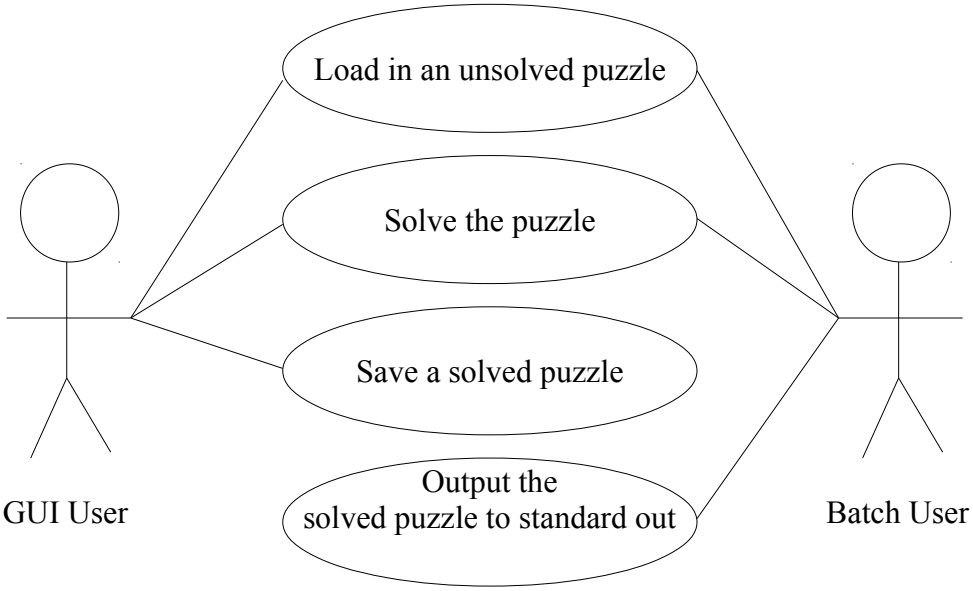
1. For every possible number(0-9):
    2. For every block:
        3. Get all occurrences of the number in the block
        4. If there is only one occurrence:
            5. Set the value of that square to be the number

Naked Singles:

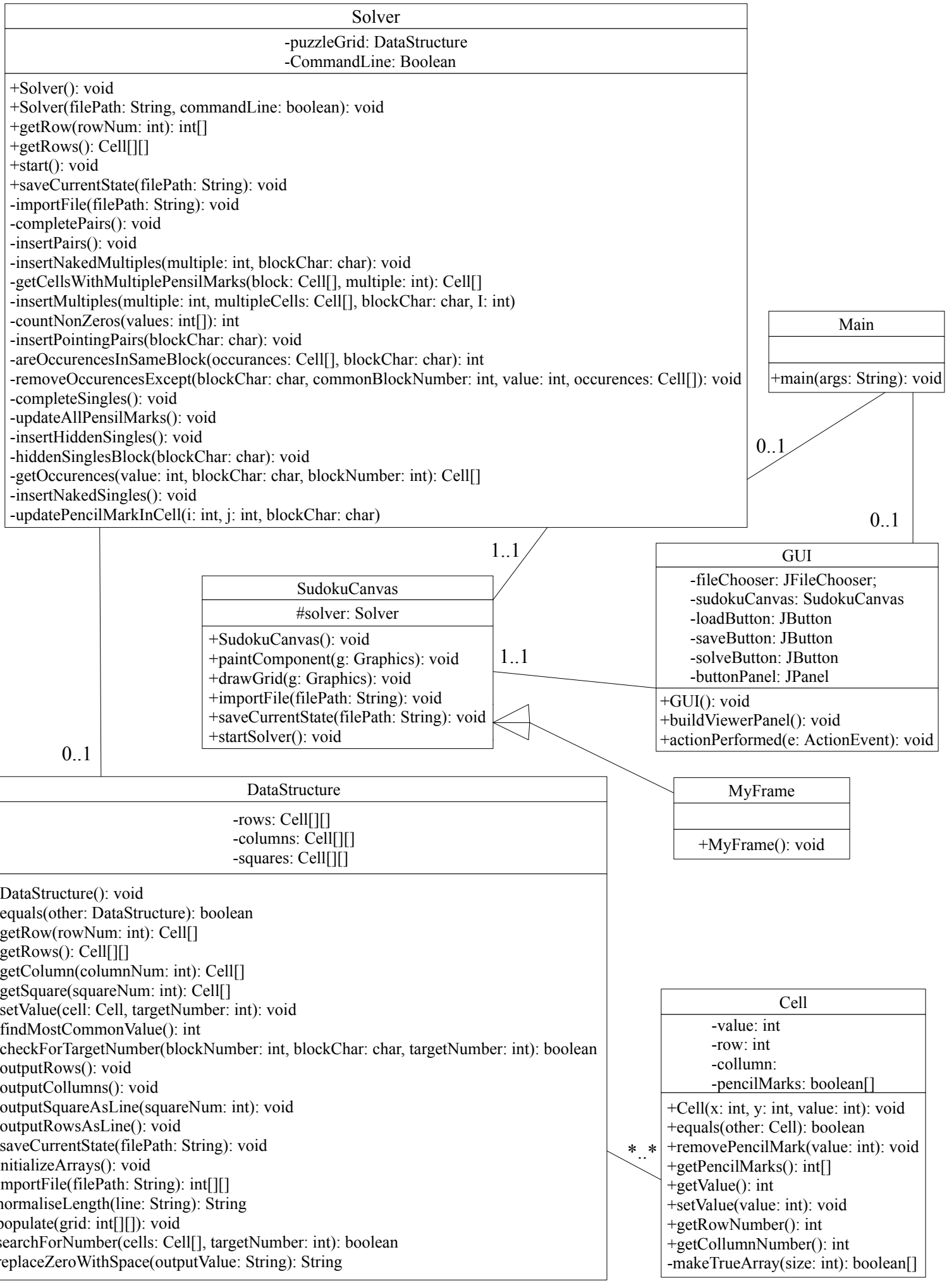
1. For every cell in the grid:
    2. If there is only one pencil mark:
        3. Set the value of the square to be that number

UML Diagrams

Use Case Diagram:



Class Diagram:



Design Decisions

I decided to use 3 2D arrays for my data structure, which all contain the same data, just in a different order. This means that it is very easy to simply traverse each row, each column or each square.

I decide to have each Cell object hold each cell, which would hold the row and column it's in, as well as it's value and the pencil marks for that cell.

For the GUI I decided to use the 3 most distinct colours, Red(255,0,0) for the solved numbers, Green(0,255,0) for the square colour, and Blue(0,0,255) for the pencil marks. It doesn't look amazingly user friendly, but it's good for debugging.

Testing

Test Plan:

I will use both Black Box and unit testing (via JUnit) to test that my code works, is robust and is maintainable.

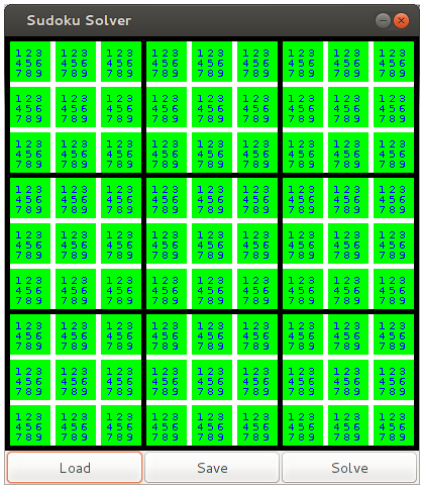
Black Box:

Test No	Test Description	Input	Expected Output	Actual Output	Pass/Fail	Evidence
1	Start program	Start the program	Program Starts	Program Starts	Pass	1
2	Load in correct data via gui	Click Load Choose web.sud	Loaded puzzle is displayed on screen	Loaded puzzle is displayed on screen	Pass	2
3	Load in wrong data via gui	Click Load Try to choose Main.class	Unable to find the file in the file finder	Unable to find the file in the file finder	Pass	3
4	Solve puzzle: book70.sud	Load the puzzle Solve it	Solve puzzle displayed	Solve puzzle displayed	Pass	4
5	Solve puzzle: web.sud	Load web.sud Solve it	Solve puzzle displayed	Solve puzzle displayed	Pass	5
6	Solve puzzle: book58.sud	Load book58.sud Solve it	Solve puzzle displayed	Solve puzzle displayed	Pass	6
7	Solve puzzle: book62.sud	Load book62.sud Solve it	Solve puzzle displayed	Only partially solved	Fail	7
8	Save the solved sudoku	Load web.sud Solve it Save it to web1.sud	Outputted file represents solved grid correctly	File created, but empty, after intensive debugging, I cannot solve the problem	Fail	8
9	Load in single	Run “java	Solved sudoku is		Pass	9

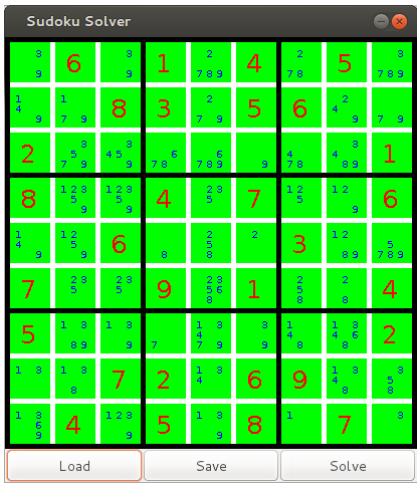
	correct data via command line	Main ../data/web.sud”	outputted to standard out as a line			
10	Load in single incorrect data via command line	Run “java Main ./Main.class”	Program ends		Pass	10
11	Load in multiple correct data via command line	Run “java Main ../data/*”	Solved sudokus outputted as lines		Pass	11

Evidence:

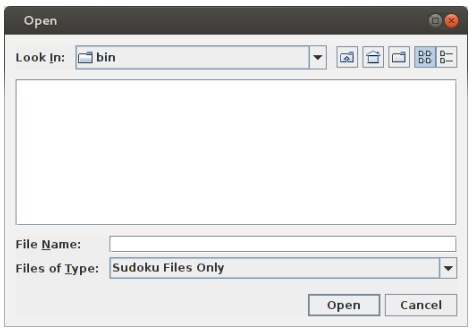
1.



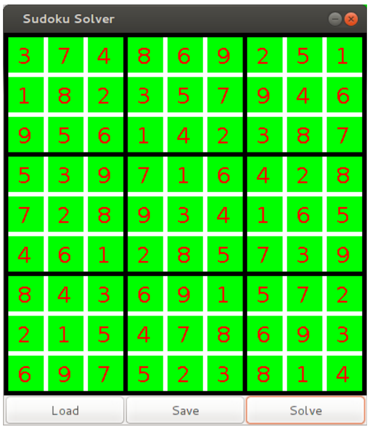
2.



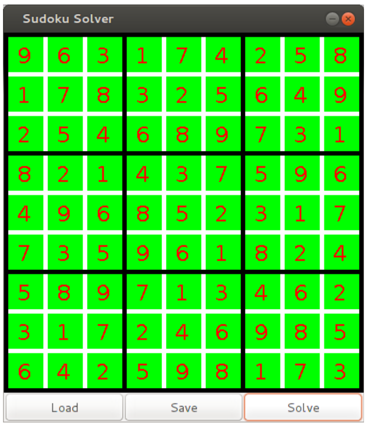
3.



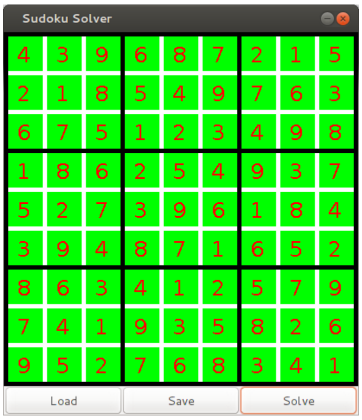
4.


Sudoku Solver
3 7 4 8 6 9 2 5 1
1 8 2 3 5 7 9 4 6
9 5 6 1 4 2 3 8 7
5 3 9 7 1 6 4 2 8
7 2 8 9 3 4 1 6 5
4 6 1 2 8 5 7 3 9
8 4 3 6 9 1 5 7 2
2 1 5 4 7 8 6 9 3
6 9 7 5 2 3 8 1 4
Load
Save
Solve

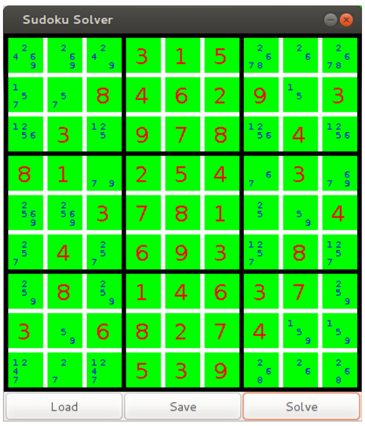
5.


Sudoku Solver
9 6 3 1 7 4 2 5 8
1 7 8 3 2 5 6 4 9
2 5 4 6 8 9 7 3 1
8 2 1 4 3 7 5 9 6
4 9 6 8 5 2 3 1 7
7 3 5 9 6 1 8 2 4
5 8 9 7 1 3 4 6 2
3 1 7 2 4 6 9 8 5
6 4 2 5 9 8 1 7 3
Load
Save
Solve

6.

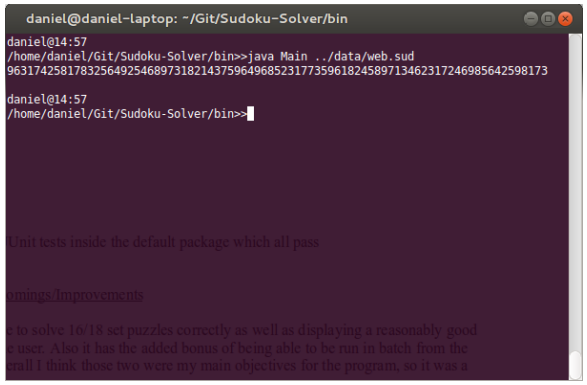

Sudoku Solver
4 3 9 6 8 7 2 1 5
2 1 8 5 4 9 7 6 3
6 7 5 1 2 3 4 9 8
1 8 6 2 5 4 9 3 7
5 2 7 3 9 6 1 8 4
3 9 4 8 7 1 6 5 2
8 6 3 4 1 2 5 7 9
7 4 1 9 3 5 8 2 6
9 5 2 7 6 8 3 4 1
Load
Save
Solve

7.

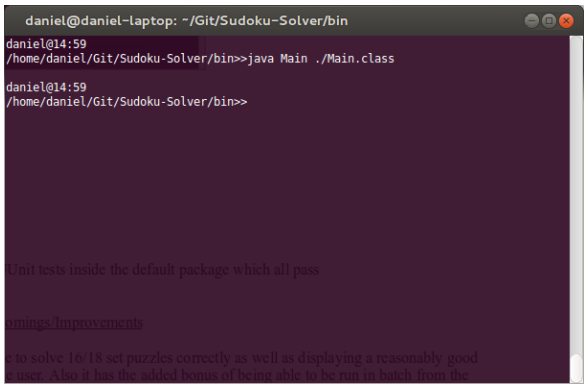

Sudoku Solver
Load
Save
Solve

8.

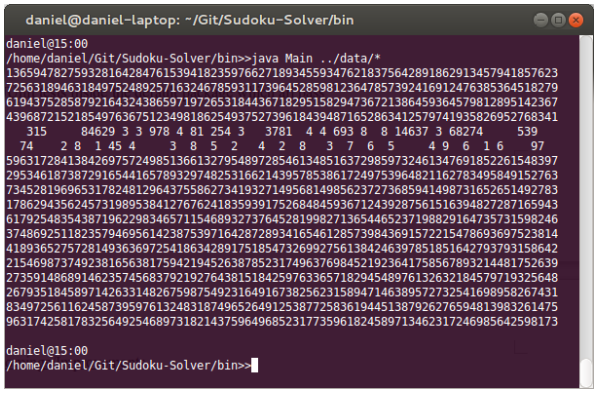
9.



10.



11.



JUnit:

There are a set of JUnit tests inside the default package which all pass

Conclusion/Shortcomings/Improvements

My program is able to solve 16/18 set puzzles correctly as well as displaying a reasonably good user interface to the user. Also it has the added bonus of being able to be run in batch from the command line. Overall I think those two were my main objectives for the program, so it was a success.

The program will not save correctly for an unknown reasons, the advisors figure it out either.

The program only decides it's finished after it has looped 10 times, which is a really inefficient way of doing it, if I had more time I would have implemented something like the following:

```
while(startStateOfGrid != currentStateOfGrid){
        startStateOfGrid = currentStateOfGrid;
        solveTheSudoku;
}
```