Please attach this coversheet to your report, including all lab records and workings

CAMBRIDGE UNIVERSITY ENGINEERING DEPARTMENT

ENGINEERING TRIPOS PART IIA

MODULE EXPERIMENT REPORT

Module no: 3F6 Experiment Title: Object Oriented Programming, Android App Development

Date Experiment Performed: 1st March Date Submitted for Marking: 6th March

Student Name: ZHUOYU ZHU Email: ZZ265 College: Wolfson
[PLEASE PRINT]

Marker: RICHARD TURNER Date: 8th March

[PLEASE PRINT NAME] *[MARKED REPORT TO BE RETURNED TO STUDENT WITHIN 15 DAYS INCLUSIVE OF HAND-IN DEADLINE]*

Percentage:	100%	90%	80%	70%	60%	50%	40%	30%	20%	10%	0%
Mark:	5	4½	4	3½	3	2½	2	1½	1	½	0

[POSITIVE CREDIT - CIRCLE AS APPROPRIATE (GUIDELINES OVERLEAF)]

Markers Comments:

See report

Total hours spent on this write-up 10

NB: Students should use both sides of the paper, and remember to keep all work for submission to the External Examiners in the Easter Term

Part IIA-Information and Computing Engineering

3F6 Android App Development

By Zhuoyu Zhu

Wolfson College

March 6, 2013

Plagiarism statement

The coursework presented and data calculated is the sole work of myself. Representing any part of works of others as my own is Plagiarism.

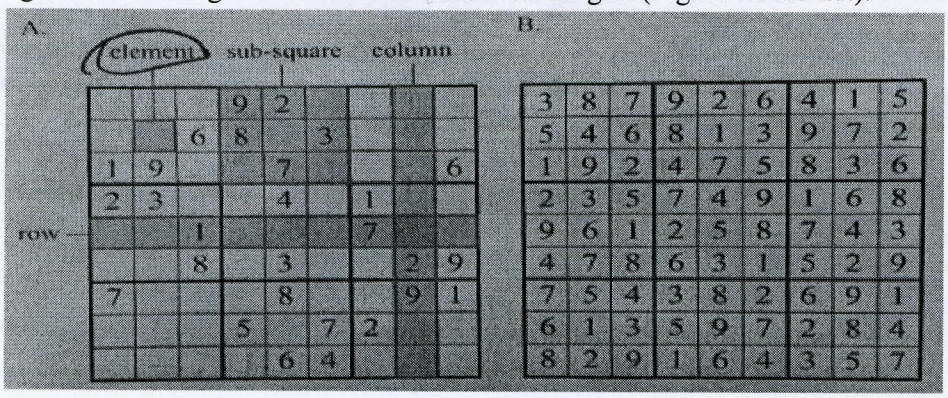
Summary

In this experiment, the object-oriented programming techniques were introduced throughout the processing of designing Sudoku puzzle for Android application. In addition, the interaction between graphical user interface and Java programming language was explored as well under the Unix Environment. Finally, the organisation of an Android mobile phone application was investigated during the whole process.

Contents

Introduction

Sudoku puzzle is a logic puzzle consisting of a nine-by-nine grid which is demonstrated in the figure below and given some initial values on the grid (Figure on the left).



			9	2				
		6	8		3			
1	9			7				6
2	3			4		1		
		1				7		
		8		3			2	9
7				8			9	1
			5		7	2		
				6	4			

3	8	7	9	2	6	4	1	5
5	4	6	8	1	3	9	7	2
1	9	2	4	7	5	8	3	6
2	3	5	7	4	9	1	6	8
9	6	1	2	5	8	7	4	3
4	7	8	6	3	1	5	2	9
7	5	4	3	8	2	6	9	1
6	1	3	5	9	7	2	8	4
8	2	9	1	6	4	3	5	7

The purpose is to fill each row, column and sub-square with the digits 1 to 9. Furthermore, the constraints are that there can be no repeated values in the elements belonging to each row, column and sub-square (Figure on the right).

Experimental work and method

The method of solving the Sudoku is to maintain a list of candidates for each element and to eliminate candidates using the constraints mentioned in the previous section. Moreover, each element on the grid contains the final value of each element and a set of possible values the element might take. The final value of each element is assigned to be zero before there is only one possible value remaining in this element.

Result and Discussion

1. Adding a button

A button is added to the app GUI which the user clicks on in order for the app to provide a solution to the current Sudoku grid. The hierarchy of objects used in the Sudoku app GUI is illustrated in the Figure 1 in Appendix A. All the "LinearLayout" are the "ViewGroup" objects while all the "EditText", "TextView" and "Button" are the "View" objects. In addition, the first "LinearLayout" on the top in Figure 1 represents the whole Sudoku grid structure while the remaining ones stand for the each row on the grid. Moreover, each "EditView" represents the each element which enables us to enter the possible values and each "TextView" stands for the gap between each 3*3 sub-square.

The text defined on the button is specified in a resource file strings.xml. It is useful to do this because of the following reasons:
1. This resource file is connected to the application in such a way that it is easy to modify the string text without needing to change the source code or recompile the application. ✓ will need to re-compile to debug
2. Using resources file is very useful in localization and internationalization of the application when the multilingual applications are developed because all that is required to do is to point to the different string files for different languages. ✓
3. Android pareses the XML resource files automatically when they are used therefore the values defined in these file can be referenced directly without extra efforts. ✓

2/2

2. Adding a method to the Element class

In this part, the getCandidateVals() method used to find the possible values which an undetermined element on the grid can take on is written inside the Element class. The added codes for this method in Element.java are shown in the Figure 2 in Appendix A. First of all, this method uses the existing method countCandidates() to return the number of true candidates for an element. Then using a loop to find out where the candidates are true. Finally, the getCandidatVals() method return an array of the possible values that the undetermined element can take on.

Other classes can modify the candidate field of an Element object by using the constructor in that class which can help the user to initialize the values for each element inside the Boolean array. This is beneficial for the design of object oriented programming because it prevents the arbitrary changes to the candidate field from the user. In addition, since the access identifier of candidate field is specified as private so the user can only access and alter the content by using the provided packaged function which helps the program to hide the raw data from the user.

Finally, the unit test is done before writing other parts of program. This illustrates an example of test-driven design where the tests are written before the programs that will pass them.

3. Writing a Sudoku solver method in Java

The recursiveSolveSudoku() is written in this part of the experiment. The added codes for this method are illustrated in the Figure 3 in Appendix A.

First of all, an undetermined element with fewest possible candidates remaining is located by using the findLeastCandidates() method. It returns an array consisting of two elements which indicates the row and column position of the undetermined element. Using the method which returns the fewest number of candidates can improve and minimise execution time inside the loop. Then after the undetermined element is found, the getCandidateVals() method from the Element class is used to return all the possible values that the element can take on.

Secondly, all the possible values of the undetermined element is checked one by one inside the loop to see if the Sudoku board is solved or not. Since there is only one undetermined element selected for each iteration of the loop so the recursiveSolveSudoku() method needs to call itself inside to start from the beginning in order to solve the other undetermined elements. The process described above continues until the program solves all the undetermined elements by calling this method repeatedly.

Thirdly, a deep copy of the current board is made at the beginning of the loop because it is essential to keep old values for back tracking. In this example, the boardOld array object get a copy of the same properties as the board array object without sharing the these properties so the modification can be made to the attribute fields independently without affecting the original board object. For instance, if there is one possible value of the undetermined element that violates the consistency criteria of Sudoku, all the elements on the grid can be restored with the original value at the end of the loop. Then this undetermined element is able to test the other possible values. Furthermore, the restore operation only needs to make a copy of the reference which is called shallow copy to the object because in this example, it might not be desirable to have two different objects which describe the same board and it is logical for the program to have one unique board object from each Sudoku grid.

4.Connecting the Solve! Button to the solver.

In this final part of the experiment, the Java codes are connected to the Android App. It involves hooking up the button to the Java solver and displaying the result. The method called solveSudoku(View view) is added into the MainActivity.java which is called when the Solve! Button is clicked. The added codes for this method are demonstrated in the Figure 4 in Appendix A. The name of this method should be the same as the android: onClick signature.

Furthermore, a second activity called DisplaySolveSudoku.java that display the solution is added. The codes added are given in the handout in Appendix B. Based on the given codes for the above method, block 1 instantiates an intent object which provides runtime binding between separate activities. In this example, it binds the MainActivity and DisplaySolvedSudoku. It assigns the SOLVE_BOARD from MainActivity as a string to the solvedBoardString. Moreover, the block 2 instantiates a textView object whose size is set to 35 and it assigns the solvedBoardString to this textView. Then the block 3 is used to set and display the result in textView. ✓ 2/2

In the end, all the activities in the manifest file, AndroidManifest.xml have to be declared and a new string is also added to the string resource file which shows the title when the Sudoku displays the result. Under the circumstance with no given values, the app is still able to solve the puzzle. The final solution makes sense because the algorithm for solving Sudoku is to find the element with the least possible candidate recursively row by row starting from the first element on the left. Therefore in the first row number displayed is 1-9 by the incrementing order while the first element on the second row is 4 due to the existing number 1-3 in the first sub-square on the top. Repeating this logic then it is concluded that the solution does make sense. 2/3 Is this a good behaviour? What happens in general if the user misses a given value & therefore might a sudoku without a unique solution? How could we alter the code to provide more informative feedback?

However, the app only shows the input numbers on the grid and leaves the other elements as zero when the Sudoku is solved by inputting some contradictory values. This is because when the first undetermined element with fewest possible values is located and it starts to try all the possible values, for each possible value the program checks if the board is consistent or not with the existing values on the board. Since the original inputs are contradictory so after trying each possible value, the program do not enter either isSudokuSolved() method or isConsistent() method and it restore the original values on the board which is the initial state that all the element are zero except the contradictory values. This process continues until it finishes checking all the possible values for the undetermined element. Therefore, the grid show all the undetermined element to be equal to zero except the contradictory values.

If the feedback is required, the codes in the MainActivity.java have to be modified in such a way that after getting the board, it is desirable to check the consistency about the initial state before starting to solve the puzzle. If the puzzle is not consistent, the program should return some messages which display the corresponding problem. The feedback messages are added in the strings.xml of resource files. 3/5

- here's a 3 line modification to MainActivity that would address this problem.

Conclusion

Throughout this experiment, the Android App development techniques are introduced by solving the Sudoku puzzle applying the GUI and Java. In addition, the principles of oriented object programming and the basic structure of Android App are explored as well during the whole process.

good presentation, could have been more incisive & clearer in places 6/8

total 37/40 = 4

Reference

Appendix A

Figure 1 – The XML tree: Demonstration of the hierarchy of objects used in the Sudoku app GUI

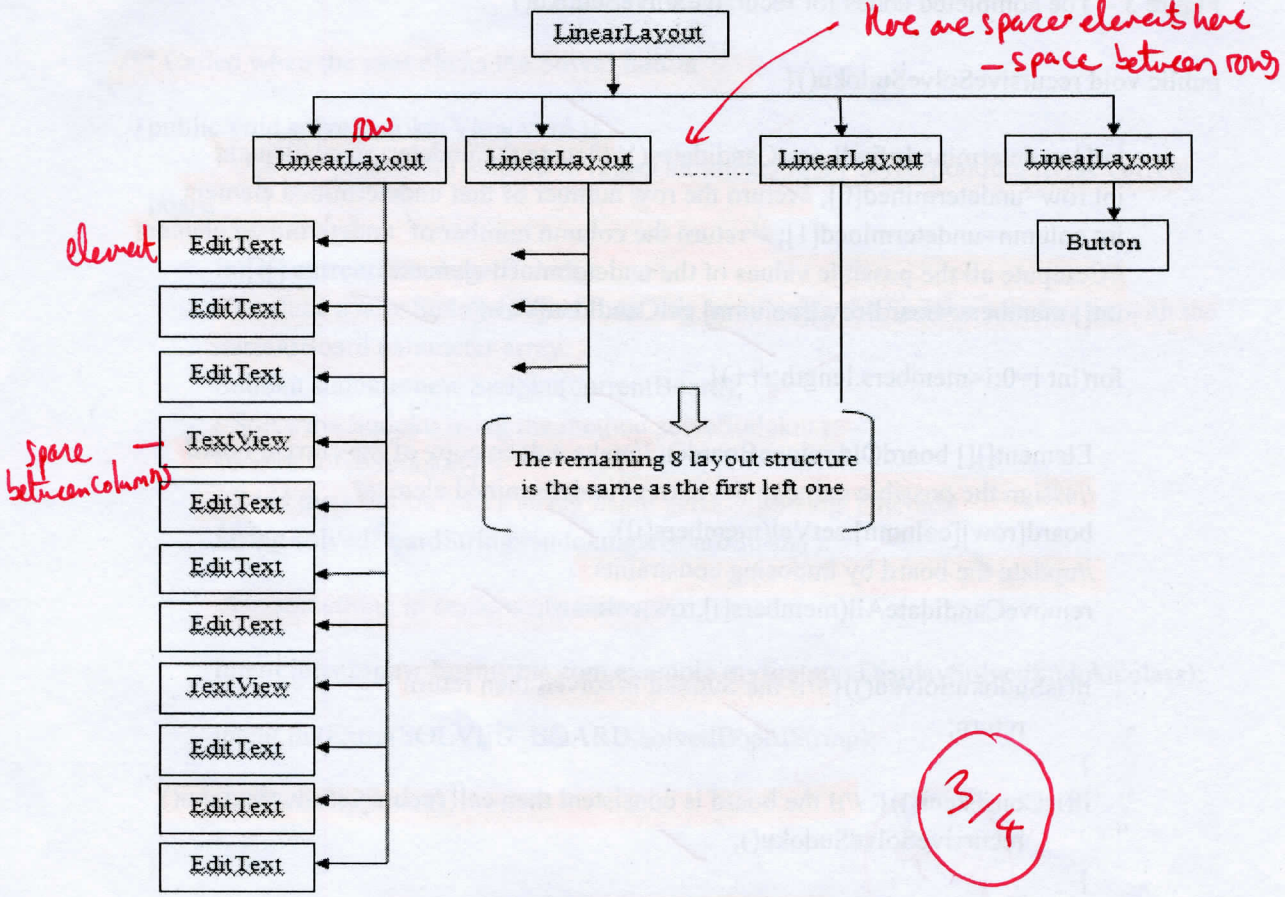


Figure 2 – The completed codes for getCandidateVals()

```
//return the possible values of the undetermined element that can take on
public int[] getCandidateVals(){

    int num=countCandidates(); //countCandidates() return the number of possible values
    int[] validCandidates=new int[num]; //define an array to store these possible values
    int j=0; //define and initialize a counter

    //iterate all the elements inside the Boolean candidates array
    for(int i=0;i<9;i++){

      if(candidates[i]){ //if the value of candidates[i] is true

          validCandidates[j]=i+1;//assign possible value to each element inside array
          j++; //increment the counter for the array
          }

    }
    return validCandidates; //return all the possible values of the undetermined element
}
```

Figure 3 – The completed codes for recursiveSolveSudoku()

```
public void recursiveSolveSudoku(){

        int[] undetermined=findLeastCandidates(); //locate the undetermined element
        int row=undetermined[0]; //return the row number of that undetermined element
        int column=undetermined[1]; //return the column number of  undetermined element
        //Compute all the possible values of the undetermined element
        int[] members=board[row][column].getCandidateVals();

        for(int i=0;i<members.length;i++){

          Element[][] boardOld=cloneBoard(); //make a deep copy of the current board
          //assign the possible value to the current undetermined element
          board[row][column].setVal(members[i]);
          //update the board by imposing constraints
          removeCandidateAll(members[i],row,column);

          if(isSudokuSolved()){ //if the Sudoku is solved then return
                return;
          }
          if(isConsistent()){  //if the board is consistent then call recursiveSolveSudoku()
                recursiveSolveSudoku();
          }
          if(isSudokuSolved()){ //if the Sudoku is solved then return
                return;
          }
          board=boardOld; //restore the board to the copy in boardOld
        }
}
```

good code.

Figure 4 – The completed codes for the added method in MainActivity.java

```
/** Called when the user clicks the Solve! button */

    public void solveSudoku(View view){
        //Use the getBoard() method to return an integer array corresponding to the current
    board

        int[][] currentBoard=getBoard();
        //Produce a new Sudoku object from this array using the Sudoku constructor with the
        currentBoard parameter array.
        Sudoku sudoku=new Sudoku(currentBoard);
        //Solve the Sudoku using the method solveSudoku()
        sudoku.solveSudoku();
        //Read the solution into a string using getBoardString() method
        String solvedBoardString=sudoku.getBoardString();

        //Do something in response to button

        Intent intent=new Intent(this,com.example.myfirstapp.DisplaySolvedSudoku.class);

        intent.putExtra(SOLVED_BOARD,solvedBoardString);

        startActivity(intent);

    }
```

4/4