UNIVERSITY OF HERTFORDSHIRE

School of Computer Science

Modular BSc Honours in Computer Science

6COM0282 – Computer Science Project

Final Report

April 2013

Sudoku Generator and Solver

S R Foster

Supervised by: Peter Lane

Abstract

The aim of this report is to discuss the development process of producing a Sudoku Puzzle Generator and Solver. The report will go into detail about each stage of the implementation process and the testing that was conducted during each implementation stage to ensure the task was working as it was intended to, without any errors.

This report will identify the aims and objectives of the project, as well as the strengths and weaknesses of the program that has been produced. Many tasks were undertaken within the development process, many of which have been successful apart from choosing a difficulty level to play at.

An overall evaluation can be seen at the end of the report as well as a brief evaluation of each of the tasks that were implemented in the program. This evaluation will also include the schedule of the project compared to the original proposed Gantt Chart as well as stating what future work could be done on the project, including improvements and extensions to the Sudoku Puzzle program that has been produced.

Acknowledgements

I would like to thank my project supervisor, Peter Lane who has been a great help throughout the entire project; he gave me motivation at the beginning of the project to keep me going, and provided me with valuable advice.

I would also like to thank my friends and family who provided help throughout this project.

Table of Contents

Introduction
History 1
Game/Rules 1
Target Audience 2
Project Aim 2
Objectives of the Project 2
Deliverables 2
Project Plan 3
Report Structure 3
Chapter 1 – Project Research
1.1 Programming Language 4
1.2 Software Application 4
1.3 Search Algorithm 5
Chapter 2 – Sudoku Completion Checker
2.1 Implementation 6
2.2 Testing 9
2.3 User Interaction 11
Chapter 3 – Generate Random Completed Puzzle
3.1 Implementation 12
3.2 Testing 13
Chapter 4 – One Possible Solution Method
4.1 Implementation 14
4.2 Testing 15
Chapter 5 – Sudoku Puzzle Solver
5.1 Implementation 16
5.2 Testing 17
Chapter 6 – Sudoku Puzzle Generator
6.1 Implementation 19
6.2 Testing 20
Chapter 7 – Evaluation
7.1 Sudoku Completion Checker 21
7.2 Generate Random Completed Puzzle 21
7.3 One Possible Solution 22
7.4 Sudoku Puzzle Solver 22
7.5 Sudoku Puzzle Generator 22
7.6 Overall Evaluation 22
7.6.1 Project Schedule 23
7.6.2 Future Work 24
References 25
Bibliography 26
Appendices
Appendix 1: Test Plan Document 28
Appendix 2: Solve Method Printout 29
Appendix 3: Generator Method Printout 30
Appendix 4: Flip Method Printouts 32
Appendix 5: Program Code 33

Introduction

History

The original thought for the game of Sudoku was based on a game called ‘Latin Squares’ created by a Swiss mathematician named LeonHard Euler in 1783. ‘Latin Squares’ was a game in which every number only appears once in each row or column. The only difference between the Sudoku puzzle and Latin Squares is that Sudoku is not only divided into rows and columns but also into regional squares of 9 cells.

Two centuries later in the 1970’s Dell Puzzle Magazine introduced a new puzzle called not Sudoku, but instead ‘Number Place’. The magazine published the puzzle along with other brain teasers for years, and after receiving much positive feedback they decided to produce a book filled with ‘Number Place’ puzzles. In the 1980’s Japan spotted the puzzle produced by Dell Puzzle Magazine and decided to make small improvements to the concept and named it Sudoku which is the name that it is globally known by to this day.

For twenty years the puzzle remained popular only in the Far East until Wayne Gould from New Zealand found the puzzle in Hong Kong and loved it so much that he decided to develop a computer program that created Sudoku Puzzles instantly. He then travelled to the United Kingdom and showed the puzzle to the Times who published it on the 12 November, 2004, Smith (2005).

As you can see Sudoku has become a phenomenon around the world and is available in various different formats. The puzzle can be found in numerous books, magazines and newspapers that have been published worldwide. The puzzle is also found in video games, computer programs, and is available on many mobile phones. Hand held devices of the game can also be purchased from high street retailers.

Game/Rules

Sudoku is a number puzzle that is represented by a 9x9 grid. Each grid cell can contain a number ranging from 1-9. This grid is then divided into nine separate grids of 3x3 which contain 9 squares. The object of the puzzle is to place the numbers 1-9 so that each column, row, and region contains no duplicates of the numbers 1-9 and does not contain any empty spaces.

When the puzzle is started, a few numbers are instantly given randomly throughout the puzzle, with the rest of the spaces being empty. The amount of numbers that are shown varies with the puzzle difficulty. A puzzle with an easy difficulty will start with more numbers displayed than a harder puzzle. The game is considered complete when all of the 81 cells of the puzzle have been filled with a number 1-9 with no empty spaces, and there are no duplicates in any of the rows, columns, and regions throughout the puzzle.

Target Audience

This Sudoku puzzle program will be targeted towards people that like to play logic puzzles and enjoy completing brain teasers. The puzzle can be played by new players that are just learning the game and will include features as the Solver to help them learn how the game is played. The puzzle is also targeted towards intermediate and expert players who enjoy a challenge. The player can play a generated game by the program, or instead insert their own puzzle into the program to play manually or to have the program solve the puzzle for them.

Project Aim

The aim of this project is to produce a Sudoku program that will generate different Sudoku puzzles at various difficulties. This program should be able to detect whether or not the puzzle has been completed successfully or not without containing any duplicates or empty spaces. The program will also be able to solve a Sudoku puzzle that has not yet been completed.

Objectives of the Project

In order to achieve the aim of this project the following list of objectives was to be the focus during implementation:

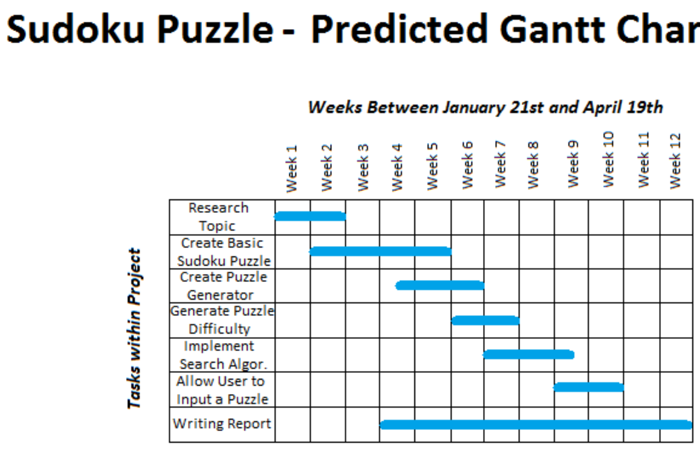
- A text based GUI was to be used to provide a grid for representing the puzzle.
- Allow the user to enter a number 1-9 into a position in the puzzle grid.
- Implement a puzzle generator that will generate random puzzles for the user to solve manually.
- Provide different levels of difficulty for the user to choose from (Easy, Medium and Hard).
- The result of a user's game can be checked to ensure it is correctly completed.
- Allow a user to input a puzzle into the puzzle grid to be played.
- Implement a puzzle solver that will allow the user to solve a puzzle that is unfinished.
- A user friendly GUI should be created if time allows.

Deliverables

The main deliverable will be a program that will allow the user to play a Sudoku Puzzle within a text-based environment. It should include functions such as a puzzle generator, different levels of difficulty, a completion checker, a puzzle solver and an optional user friendly GUI-interface. A report will also be included containing the research and evaluation of the completed project.

Project Plan

In order to finish my project by the given deadline, I decided to create a Gantt chart that would provide a schedule for me to see when certain objectives should be completed, and how long each task should take. I first broke down my tasks in a similar way to the objectives that can be seen above. Each task contains a bar which shows how long the given task should take and contained an estimated time period that should be allowed for each task. The Gantt chart that has been created can be seen below.



Report Structure

The chapters in this report are separated into separate tasks/objectives explaining the implementation of each task. At the end of each chapter there is a small paragraph explaining the testing that has been done on the given task and what the results of the test were. The 1st chapter contains the research that was done for this project. The 2nd chapter explains the implementation of the method that checks if the solved puzzle has been completed successfully or not. The 3rd chapter explains the implementation of the puzzle generator that generates different completed puzzles. The 4th chapter explains the implementation of the method that checks if a given position only has one possible solution. The 5th chapter explains the implementation of the puzzle solver. The 6th chapter explains the implementation of the puzzle generator that removes numbers from the puzzle grid. The 7th chapter contains the evaluation of the project as a whole, an evaluation of each task within the project, future work that could be done on the program and the final project schedule.

The 1st appendix contains a Test Plan document that holds all of the tests that are required in order to fully test each method to ensure there are no errors. The 2nd appendix contains a printout of the Solve method when it is called. The 3rd appendix contains a printout of the Generator method removing numbers from the puzzle. The fourth appendix contains a printout of each of the flip methods being called on a single puzzle.

Chapter 1 – Project Research

1.1 Programming Language

Many different programming languages could have been used for the implementation of this project, but I decided to narrow this down to Python and Java as I have had experience using both languages prior to this project; both options and their benefits can be seen below:

Java – Java could have been used to implement this puzzle as the language was designed to be very easy. Not only is it easy to read and write but also to debug and compile. Java is also platform independent which means it can be easily moved from one computer system to a different one. Java is also known for its robustness and security. Java compilers can detect certain problems that other programming languages would not detect until the execution of the program, MS (2007). Out of all programming languages I have had the most experience using Java and have experience using it for three years, Patel ([n.d.]).

Python – Python could have been also used for the implementation of the Sudoku puzzle as it is very easy to understand and write, Bach (2007). Python also requires far less lines of code to express certain concepts, Rossum (1997). According to Dawson, code is often three to five times shorter than the same program written in Java, Dawson (2003). The language can also be used on projects of a small or a large scale. Python allows for search algorithms to easily be added to code in order to solve a given puzzle. I have had some experience using Python to code smaller puzzles in my Artificial Intelligence module.

After I had conducted my research, I decided to use the programming language Java to implement my puzzle. I felt that with the time constraints it would be better to use a programming language that I felt most comfortable using. I have had experience using Java for three years, and although I have used Python previously for coding puzzles, I only had around five months of experience using the language. Because of this I felt that using Python would require a lot of my time learning the language and learning how to implement certain features. I also felt that Java would still be capable of implementing all of the features I wanted within my puzzle so this seemed like the best programming language to use.

1.2 Software Application

Once I figured out I was going to use Java for my programming language, I then needed to decide which software application I would use to create my Sudoku Puzzle program. After researching I found that the most common programming IDE's were Eclipse, NetBeans, and JBuilder X. When I attended my supervisor meeting, I brought up the different IDE's that I had found and we both felt that NetBeans would be a great IDE for me to choose as it included a test framework called JUnit and the program was easy to use and learn. JUnit allows you to run multiple tests that you have created at the same time. This can be useful not only to test each method but to also ensure that when you create new methods, your old methods are still working as they are intended to. The software is easy to use and can fully test each method that has been created, Programmer (2012).

1.3 Search Algorithm

The next thing I needed to research was different search algorithms that could be used in the solver task. This method would need to be efficient and produce a correct Sudoku puzzle without any errors, and be somewhat fast. Although testing will be required to determine which search algorithm would be used, I looked into the most common search algorithms and noticed how they worked as well as looking at their advantages and disadvantages. A list of the most common search algorithms used for Sudoku puzzles can be seen below along with their advantages and some disadvantages.

Depth First Search – This search algorithm starts at the top root and explores down each branch as far as possible and then will backtrack up the branch again. Backtracking seems to be the most common search algorithm used amongst other Sudoku puzzle programs I have found, Matuszek (2002).

Brute Force Algorithm – This algorithm can also be used amongst Sudoku puzzles but requires the code to be designed towards using this solving algorithm. The algorithm enumerates all possible candidates for the solution and checks if they are valid or not. The solution is always guaranteed but the time required to solve the puzzle can vary depending on the puzzles degree of difficulty, Webopedia (2013).

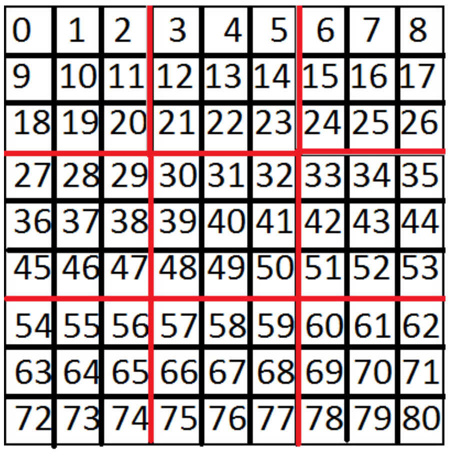
Stochastic Search – this search uses a random number search method. It fills random numbers into the empty cells in the grid and will then calculate the number of errors in the puzzle. The search algorithm will then shuffle these numbers until the puzzle contains no errors. This type of search is known to be fast, but can be slow compared to some other search algorithms that are available, Spall (2004).

Chapter 2 – Sudoku Completion Checker

2.1 Implementation

The object of this task was to create a basic program that would create a pre-set Sudoku puzzle and contain a method that would check each row, column and square of the Sudoku puzzle to determine if the puzzle was completed successfully or not. This required checking that each row, column, and square has no duplicate numbers and that every square was filled with no empty spaces (0's). If the puzzle has been completed successfully "Puzzle Completed Successfully" will be printed to the terminal, otherwise "This Puzzle is Invalid" will be printed. This task requires no user interaction, but will instead use a pre-inserted puzzle to test the methods for debugging. User Interaction will be added once the method is implemented and tested successfully.

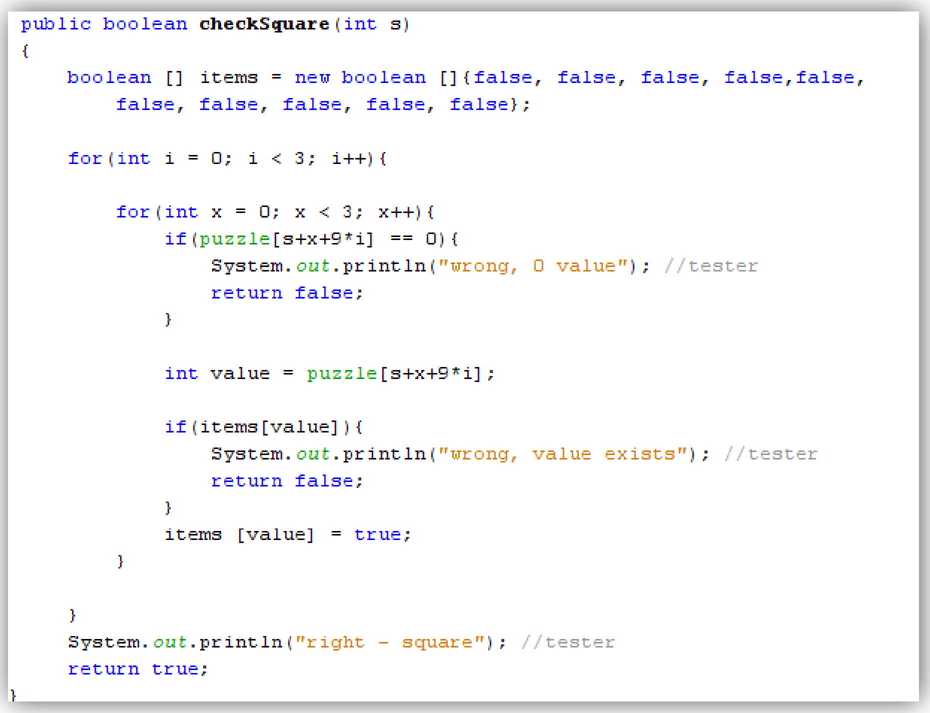
To start this task I first researched different data structures in order to determine which would be the best suited for my project. I decided to store the values of my puzzle inside an array, but I was not sure whether to use a 1D or 2D array. After researching and talking to my supervisor, I decided to use a 1D array that has a size of 81 as I have more experience using this type of array and felt that because of its flexibility, all of the tasks could be completed with this type of data structure. The array structure for the Sudoku puzzle is shown below:



0	1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16	17
18	19	20	21	22	23	24	25	26
27	28	29	30	31	32	33	34	35
36	37	38	39	40	41	42	43	44
45	46	47	48	49	50	51	52	53
54	55	56	57	58	59	60	61	62
63	64	65	66	67	68	69	70	71
72	73	74	75	76	77	78	79	80

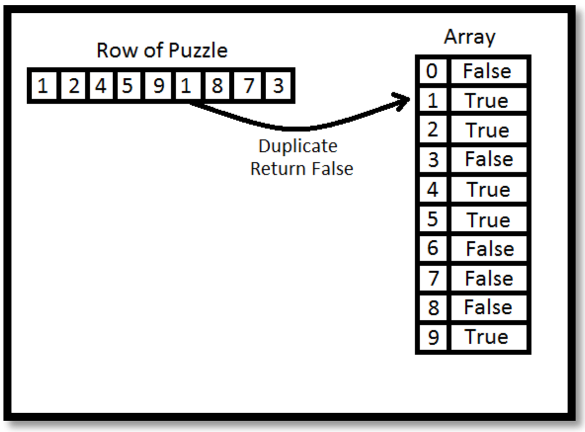
After I had decided to use a 1D array I chose to use 0's as the empty spaces and the numbers 1-9 to represent the puzzle. I created two constructors, the first created an array full of empty spaces (0's) and a second constructor that allowed the user to enter 81 parameters in order to input a puzzle. This was required in order to test my methods with JUnit, but will also be required in future chapters when a user can insert an unfinished Sudoku puzzle that can be played in the program.

The next step was to determine what methods would be required in order to check if a Sudoku puzzle had been completed. I decided to implement methods to check each row, each column, and each square along with a final method that would call all of these methods which would determine if the puzzle had been completed successfully. Instead of embedding two ‘for loops’, I decided to create two methods for each check, one that would read a single row/column/square of the puzzle, and the second would call the single row/column/square check for each of the 9 rows/columns/squares in the puzzle. By splitting the methods into two different methods, it made the code easier to read and would be easier to test each method and to debug them. Below is an example of the Check Square method.

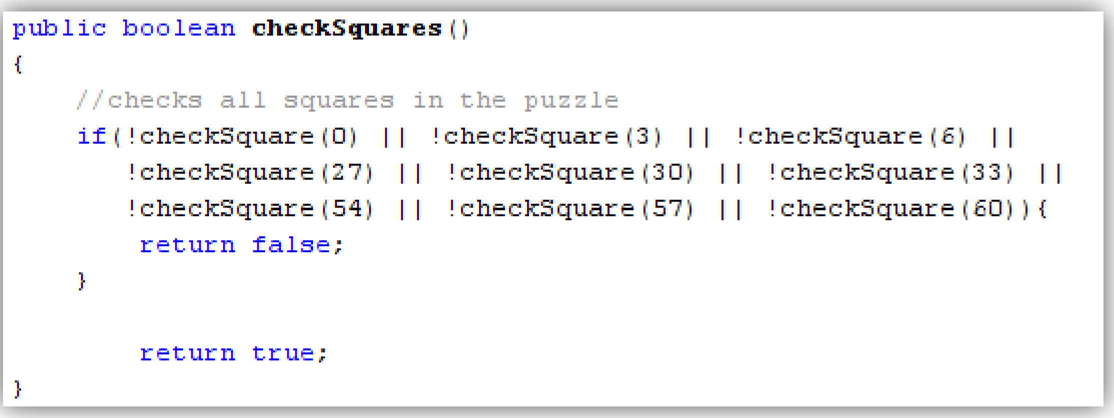


```
public boolean checkSquare(int s)
{
    boolean [] items = new boolean []{false, false, false, false,false,
        false, false, false, false, false};

    for(int i = 0; i < 3; i++){

        for(int x = 0; x < 3; x++){
            if(puzzle[s+x+9*i] == 0){
                System.out.println("wrong, 0 value"); //tester
                return false;
            }

            int value = puzzle[s+x+9*i];

            if(items[value]){
                System.out.println("wrong, value exists"); //tester
                return false;
            }
            items [value] = true;
        }

    }
    System.out.println("right - square"); //tester
    return true;
}
```

As you can see from the image above, the Check Square method contains an array called ‘items’ with 10 positions. This array ‘items’ contains Booleans that were by default set to ‘False’ in each of the check methods. When the check method was called, it would determine which number 1-9 was in the cell and set that position in the second array to ‘True’. If this position in the array was already set to true, it would stop the check method and return ‘False’, stating that the puzzle is invalid as there was a duplication of numbers. An example of this can be seen in the diagram below.



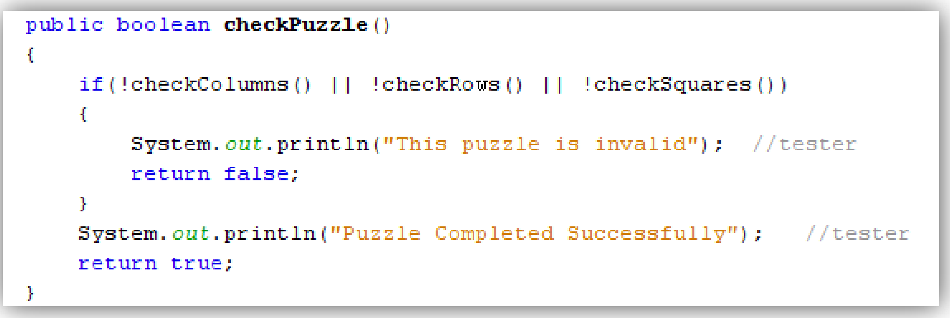
This array holds the numbers that have been seen during the check as it goes through the ‘for loops’. Next it will go through a loop for the rows, and then a second loop for the columns. This will iterate through the square starting at the most upper left position, going 3 cells over, and then going down to the next row repeating these steps. When it enters the second ‘for loop’, it will determine what square it is in by taking the user input of the most upper left position of the square and call the mathmatical equation “s + x + 9 * i” so for example if I wanted to check the first square starting at array position 0, the equation would be 0 + 0 + 9 * 0 = 0, for the next loop around it would be 0 + 1 + 9 * i = 1. As you can see this mathmatical equation will loop through one square determined by the ‘s’ value that is inputted. Once this mathmatical equation is completed, it will check if the cell position in the main array contains a 0, if it does it will return ‘False’. Otherwise it will store the variable that is inside that position and check the boolean array ‘items’ to determine if it is already ‘True’, if it is it will return ‘False’, else it will make that position of the array as ‘True’, and loop around once again. The next method to check all squares in the puzzle is shown below.


```
public boolean checkSquares()
{
    //checks all squares in the puzzle
    if(!checkSquare(0) || !checkSquare(3) || !checkSquare(6) ||
       !checkSquare(27) || !checkSquare(30) || !checkSquare(33) ||
       !checkSquare(54) || !checkSquare(57) || !checkSquare(60)){
        return false;
    }

        return true;
}
```

As you can see from the image above, the checkSquares method checks all of the squares in the puzzle. It calls the checkSquare method for each square calling the most upper left position of the square. The check squares method hardcoded the starting positions of the squares. This was required as there is no way to iterate through the squares because there is no pattern that connects them. If any of these checks returns a ‘False’ value, the method will return ‘False’, else if all methods are successful it will return ‘True’.

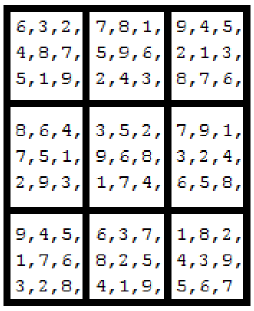
The methods to check rows and columns are very similar to the code that was shown above for checking the squares although they slightly differ. The check column and row methods contain 1 ‘for loop’ in each of the methods, and contain different mathematical equations. The mathematical equation for the column is “x + 9 * i” and the equation for the row is “9 * x + i“, these different equations are located in the same place as the check square method and do the same job but for a different section of the puzzle. The check rows and columns methods contain one ‘for loop’ that loops for 9 rounds, then contains an if statement determining whether the check single column or row is ‘False’, if it is then the method will return a ‘False’, otherwise it will return ‘True’.

After these check methods had been created, I then created a check puzzle method that would call all of these methods in order to determine if the whole puzzle was completed successfully or not. The result of this test would print “Puzzle Completed Successfully” or “This Puzzle is Invalid” depending on the result of the check.


```java
public boolean checkPuzzle()
{
    if(!checkColumns() || !checkRows() || !checkSquares())
    {
        System.out.println("This puzzle is invalid");  //tester
        return false;
    }
    System.out.println("Puzzle Completed Successfully");   //tester
    return true;
}
```

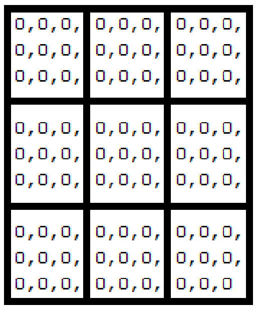
2.2 Testing

After implementing all of these methods, I then created a test by using JUnit. A full test plan for each of the methods that have been implemented can be seen in further detail by looking at Appendix 1 on page 28. To test this task I created 4 tests, the first checks a successfully completed puzzle that calls random checks on each row, column, and square to make sure it returns ‘True’. I then created 3 tests for incorrectly completed puzzles, checking each row, column, and square that the error lies on to make sure it returns ‘False’, as well as checking correct sections of the puzzle to ensure it returns ‘True’. Each of these tests contained a different puzzle with random empty positions and duplicates. The puzzles that have been tested are shown below.



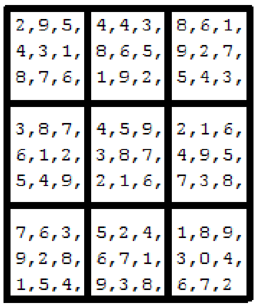
6,3,2, 4,8,7, 5,1,9,	7,8,1, 5,9,6, 2,4,3,	9,4,5, 2,1,3, 8,7,6,
8,6,4, 7,5,1, 2,9,3,	3,5,2, 9,6,8, 1,7,4,	7,9,1, 3,2,4, 6,5,8,
9,4,5, 1,7,6, 3,2,8,	6,3,7, 8,2,5, 4,1,9,	1,8,2, 4,3,9, 5,6,7

testGoodPuzzle1



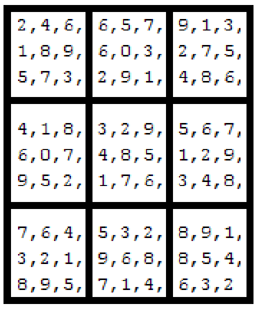
0,0,0, 0,0,0, 0,0,0,	0,0,0, 0,0,0, 0,0,0,	0,0,0, 0,0,0, 0,0,0,
0,0,0, 0,0,0, 0,0,0,	0,0,0, 0,0,0, 0,0,0,	0,0,0, 0,0,0, 0,0,0,
0,0,0, 0,0,0, 0,0,0,	0,0,0, 0,0,0, 0,0,0,	0,0,0, 0,0,0, 0,0,0

testBadPuzzle1



2,9,5, 4,3,1, 8,7,6,	4,4,3, 8,6,5, 1,9,2,	8,6,1, 9,2,7, 5,4,3,
3,8,7, 6,1,2, 5,4,9,	4,5,9, 3,8,7, 2,1,6,	2,1,6, 4,9,5, 7,3,8,
7,6,3, 9,2,8, 1,5,4,	5,2,4, 6,7,1, 9,3,8,	1,8,9, 3,0,4, 6,7,2

testBadPuzzle2



2,4,6, 1,8,9, 5,7,3,	6,5,7, 6,0,3, 2,9,1,	9,1,3, 2,7,5, 4,8,6,
4,1,8, 6,0,7, 9,5,2,	3,2,9, 4,8,5, 1,7,6,	5,6,7, 1,2,9, 3,4,8,
7,6,4, 3,2,1, 8,9,5,	5,3,2, 9,6,8, 7,1,4,	8,9,1, 8,5,4, 6,3,2

testBadPuzzle3

First I tested a new puzzle ‘testGoodPuzzle1’ that checked different rows, columns, and squares to determine whether the test was completed successfully, without the puzzle containing any errors. I checked the first and last row/column/square and a random one in the middle. For this test I checked rows 0, 3 and 8, columns 0, 4 and 8, and squares 1, 4 and 8. The checkRows, checkColumns, and checkSquares methods were also called. The Check Puzzle method was finally called to determine whether or not the puzzle had been completed successfully. All of these methods returned ‘True’ as the puzzle did not contain duplicates or empty spaces.

Next I tested a new puzzle ‘testBadPuzzle1’ that contained a puzzle filled with empty spaces. To test for these empty spaces, I checked rows 0, 3 and 8, columns 0, 4 and 8 and squares 1, 5 and 8. The checkRows, checkColumns, and checkSquares methods were also called as well as checkPuzzle. All of these methods returned ‘False’ as the puzzle contained empty spaces.

Next I tested a new puzzle ‘testBadPuzzle2’ that contained a duplicate number and an empty space. To test for the duplicate number, I checked row 0, column 3, and square 2. All of these checks returned ‘False’ as they all contain a duplicate number 4. The empty space was also checked by testing the row 7, column 7, and square 9. All of these methods returned ‘False’ as they all contained an empty space. I then checked rows 4 and 6, columns 1 and 5 and squares 4 and 7 to ensure that they returned ‘True’ as these rows, columns, and squares contained no duplicates or empty spaces. A final check was created to ensure that the Check Puzzle method would return ‘False’ as the puzzle was invalid and contained duplicates or errors.

Finally I tested a new puzzle ‘testBadPuzzle3’ that contained two sets of duplicates, and one empty space. To test for the first duplicate ‘6’ I checked row 0, column 3, and square 2. The second duplicate ‘8’ was checked in the same manner by checking row 7, column 6 and the square 9 starting on position 60. To check for the empty space I checked row 1, column 4 and square 2. All of these methods should return ‘False’ as they contain a duplicate or empty space. I then tested correct segments of the puzzle to ensure that they would return ‘True’. To test this I checked rows 2 and 5, columns 2 and 7 and squares 1 and 6. These all returned ‘True’ as they contained no empty spaces or duplicates. The Check Puzzle method was also called and returned ‘False’ as the puzzle contained errors. A final check was done to ensure that the puzzle was completed successfully just too finally ensure that the puzzle had flipped correctly and that the puzzle was indeed still met the requirements of a Sudoku puzzle and contained no duplicates or empty spaces. All of these tests exercise all parts of the code to ensure no errors would occur in any situation, and all of the tests passed when I ran them.

2.3 User Interaction

Once the implementation and testing of the Sudoku Completion Checker task, I then decided to put in a method that would allow user interaction. This would allow the user to input a number 1-9 into the puzzle at a certain position. This was done by storing the number in the puzzle array at the position inserted by the user. The new puzzle would then be printed to the terminal window. If the number is not a valid number 0-9, or the position number was not between 0-80, an error message would be printed to the terminal window stating that the move is invalid.

Once this method was implemented I then created a test using JUnit. To test this method I created a fully empty puzzle and randomly inputted numbers 1-9 at random positions within the array. I then used a Boolean to determine if the position was filled with the number I inserted using the method. I also checked that random positions that had not had the method called on them have not changed from 0. On these same positions I then used the method to change the number using a number that was not between 0-9. I then finally checked that the puzzle positions had not changed from 0. This test exercised all parts of the code and passed the test.

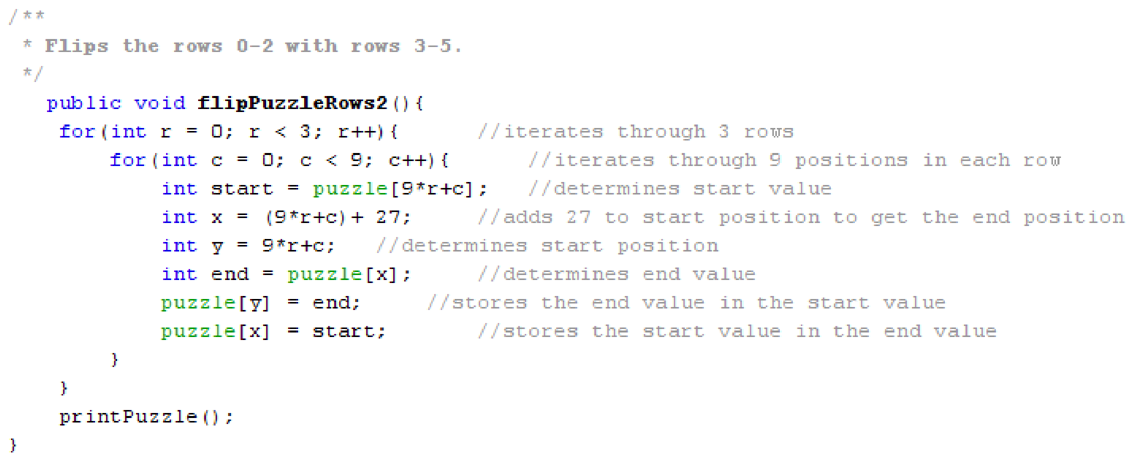
Chapter 3 – Generate Random Completed Puzzle

3.1 Implementation

The next task was to find a way to create a Sudoku puzzle that satisfied all of the Sudoku constraints. The puzzle that was created would then be used in a later task to generate an unsolved puzzle for the user to solve. I first researched different approaches and algorithms that were commonly used for creating Sudoku puzzles. I found two articles by Peter Forret and Dry Icons that mentioned manipulating a puzzle to create a new puzzle, Dry Icons (2009), Forret (2006), Zhan([n.d.]). This could be done in several ways such as flipping the puzzle horizontally, vertically, swapping rows and columns. Not only can these flips be done by themselves, but will also create a solvable puzzle if multiple flips are used, Dry Icons (2009), Forret (2006), Zhan([n.d.]).

Although this method is not creating a genuine random puzzle but instead manipulating a puzzle, the effect would most likely not be noticed by a user playing the game. I decided to create ten different methods that would manipulate the puzzle in a random way. The flips that I decided to include were flipping the puzzle horizontally then vertically, horizontally, vertically, swapping rows 0-2 with rows -5 and 3-5 with 6-8, swapping rows 0-2 with 3-5, swapping rows 4-5 with 6-8, swapping rows 0-2 with 6-8, swapping columns 0-2 with 3-5, swapping columns 0-2 with 6-8, and swapping columns 3-5 with 6-8. With only one inputted puzzle the user could then play up to 10 different puzzles by just calling one of these methods above, if multiple methods were called above many more puzzles could be created by using this manipulation technique.

Each method contained different mathematical equations to determine the start position of the first variable that needed to be switched. “For loops” were used to loop through the Sudoku puzzle until it reached the last position that needed switching. Every time it went through the loop it would store the starting position inside a temporary variable and store the variable that it should be switched with in another temporary variable. The temporary variables were then used to store in the opposite position of the array: start state = end state temp and end state = start state temp. An example of one of these flip methods is shown below:



```java
/**
 * Flips the rows 0-2 with rows 3-5.
 */
  public void flipPuzzleRows2(){
   for(int r = 0; r < 3; r++){       //iterates through 3 rows
       for(int c = 0; c < 9; c++){       //iterates through 9 positions in each row
           int start = puzzle[9*r+c];    //determines start value
           int x = (9*r+c)+ 27;      //adds 27 to start position to get the end position
           int y = 9*r+c;    //determines start position
           int end = puzzle[x];      //determines end value
           puzzle[y] = end;      //stores the end value in the start value
           puzzle[x] = start;       //stores the start value in the end value
       }
   }
   printPuzzle();
}
```

Each of the 10 flip methods on the same completed Sudoku puzzle can be seen in Appendix 4 on page 32.

3.2 Testing

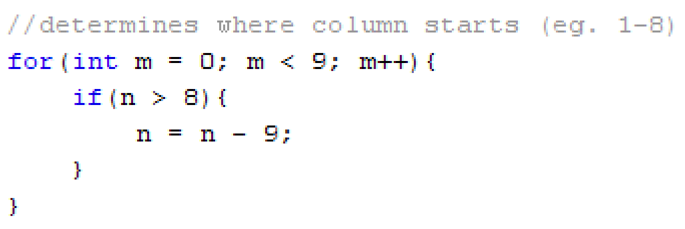
After implementing all of the puzzle flip methods, I then created a test by using JUnit. A full test plan for each of the methods that have been implemented can be seen in further detail by looking at Appendix 1 on page 28. To test this task I created 10 tests, one for each of the methods that was created to ensure they were working as they were intended. Each test contained a finished puzzle and a second puzzle that were flipped in the correct way. The finished puzzle would then call the flip method and the two puzzles were compared by selecting random cells that were changed and see if they both contain the same value. Many checks were made on each puzzle containing the positions that had been flipped as well as positions that stayed the same to ensure that no errors had occurred. All of these tests exercised all parts of the code to ensure no errors would occur in any situation, and all of the tests passed when I ran them.

Chapter 4 – One Possible Solution Method

4.1 Implementation

The object of this task was to create a method that given a location on the board, would determine whether or not only one possible number 1-9 could be inserted into the specified position, while still following Sudoku constraints such as 1-9 in each row, column and square.

To start implementing this method I first created an array of Booleans called items. This technique has been shown in the chapter talking about the Sudoku completion checker. This Boolean is used in the same way by checking the position ‘True’ if the number has been seen before. Next I used the mathematical equation a/9 where ‘a’ is the position being checked. This determined what row the position lied on. To check which column the position lied on, I used a ‘for loop’ with an ‘if statement’ inside. The ‘if statement’ determined whether the position was greater than 8, if it was then it would subtract 9 from the position number. This would loop around 9 times ensuring that the final number would be a number 1-8 which is the starting points of each column. The code for checking which column the position lies on is shown below.



```
//determines where column starts (eg. 1-8)
for(int m = 0; m < 9; m++){
    if(n > 8){
        n = n - 9;
    }
}
```

The squares were hardcoded determining whether the position was between certain positions inside the array. Once the row, column, and square the position lied on were found, the method would then set the Boolean array to ‘True’ for each number that was found in the row, column, and square the position lied on. After this had been completed a loop would count how many ‘False’ values were still in the array. Finally an ‘if statement’ was created to determine if only 1 number was ‘False’, if so then it would return that number, otherwise it would return ‘False’.

This method would be used later on in future chapters to generate a random Sudoku puzzle with empty spaces as well as solving the puzzle.

4.2 Testing

After implementing this method, I then created a test by using JUnit. A full test plan for each of the methods that have been implemented can be seen in further detail by looking at Appendix 1 on page 28. To test this task I created a test that would test empty spaces as well as spaces that are filled with a number 1-9 to see the result that was returned by the method. I first started by inputting a completed puzzle and manually removed numbers from the puzzle and changed them to 0. The numbers that had been selected should only have one possible solution but also contain multiple empty spaces on a row, column or square to ensure the method would work properly. Eight of the cell numbers were removed and have been set to empty. I then checked that each of these cell positions returned a number 1-9 which was correct for that cell. The correct number had to be checked manually. I then checked that filled positions of the puzzle did indeed return a 0 result. This is important as it will be used in later chapters when generating a new puzzle, as the puzzle will be completely filled which would result in a 0 return value from the method. As you can see this test exercised all parts of the code to ensure no errors would occur in any situation, and all of the tests passed when I ran them.

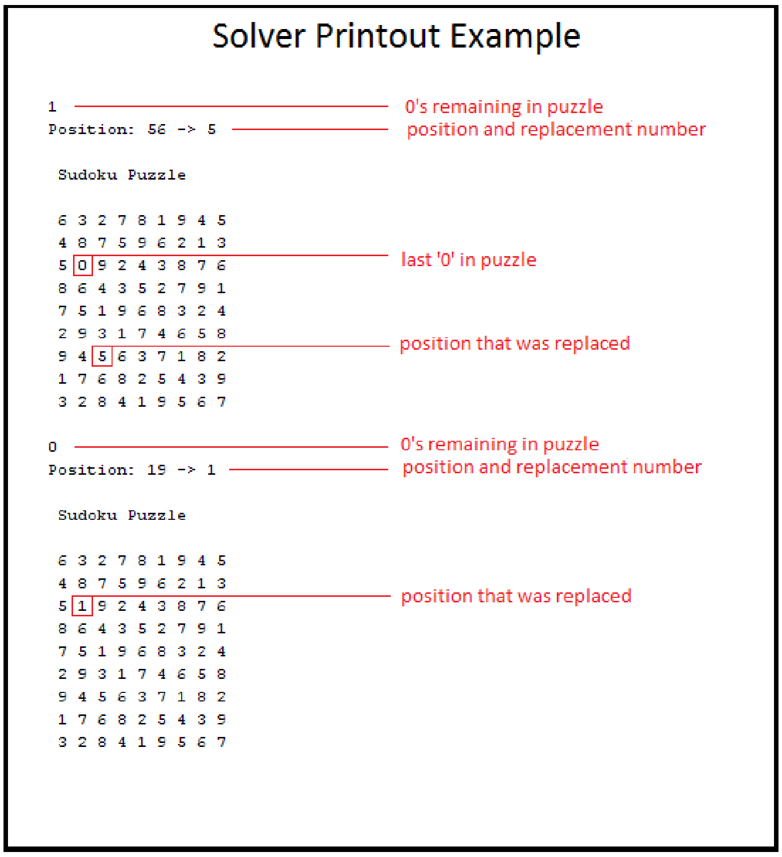
Chapter 5 – Sudoku Puzzle Solver

5.1 Implementation

The objective of this task was to create a method that would solve an easy Sudoku puzzle. This task will use the method that was mentioned in the Check One Solution chapter in order to solve the puzzle.

The first thing I did was to create a counter to determine how many empty spaces were in the puzzle. This was done by creating a ‘for loop’ that iterated through each of the 81 positions and determined if the position value was 0, if it was it would increment an integer variable. Once this count had been completed, it would enter a ‘while loop’. The ‘while loop’ will run as long as the counter variable is greater than 0. Inside the while loop contained a “for loop” that would iterate through each of the 81 positions in the array. It would then check each position to determine if the value was 0, if it was it would call the Check One Choice method using the position as the parameter. The result from the Check One Choice method would either be a number 1-9 if only one choice was available, or 0 if there were multiple choices the position could be. It would then store this number inside the position of the array it is in.

For every check if the Check One Choice method returns a number 1-9 it will first subtract 1 from the counter method and print out how many 0’s are still in the puzzle and the position number followed by the number that was inserted into the puzzle. The new puzzle was then printed to the terminal window. An example of this printout is shown below.

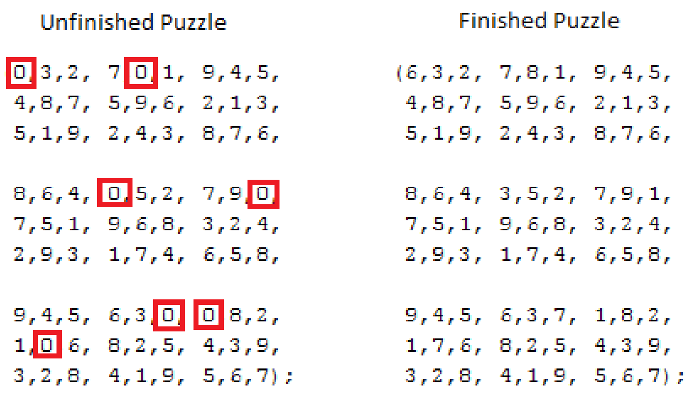

Solver Printout Example
0's remaining in puzzle
position and replacement number
last '0' in puzzle
position that was replaced
0's remaining in puzzle
position and replacement number
position that was replaced

```
1
Position: 56 -> 5

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 0 9 2 4 3 8 7 6
 8 6 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 4 5 6 3 7 1 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

0
Position: 19 -> 1

 Sudoku Puzzle

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

An example of a full printout of the Solver method solving an uncompleted generated puzzle can be seen in Appendix 2 on page 29.

5.2 Testing

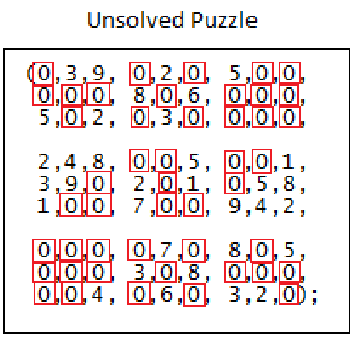
After implementing the Sudoku solver method, I then created a test by using JUnit. A full test plan for each of the methods that have been implemented can be seen in further detail by looking at Appendix 1 on page 28. To test this task I created a test that creates a Sudoku puzzle that contains empty spaces randomly throughout the puzzle, and another that was the same Sudoku puzzle but was fully completed. The puzzle that I tested is on the left, and the same puzzle that had been correctly completed on the right. The two puzzles are shown below:



```
Unfinished Puzzle                      Finished Puzzle

0,3,2, 7,0,1, 9,4,5,                   (6,3,2, 7,8,1, 9,4,5,
4,8,7, 5,9,6, 2,1,3,                    4,8,7, 5,9,6, 2,1,3,
5,1,9, 2,4,3, 8,7,6,                    5,1,9, 2,4,3, 8,7,6,

8,6,4, 0,5,2, 7,9,0                     8,6,4, 3,5,2, 7,9,1,
7,5,1, 9,6,8, 3,2,4,                    7,5,1, 9,6,8, 3,2,4,
2,9,3, 1,7,4, 6,5,8,                    2,9,3, 1,7,4, 6,5,8,

9,4,5, 6,3,0, 0,8,2,                    9,4,5, 6,3,7, 1,8,2,
1,0,6, 8,2,5, 4,3,9,                    1,7,6, 8,2,5, 4,3,9,
3,2,8, 4,1,9, 5,6,7);                   3,2,8, 4,1,9, 5,6,7);
```

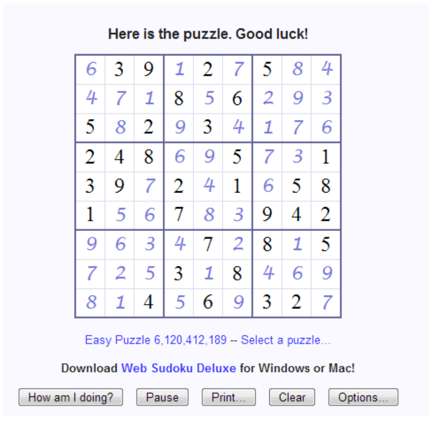
As you can see from the diagram above, the first puzzle contains 7 empty spaces within the board. To test that the solver was working correctly I called the solver method on the unfinished puzzle. After this I then checked that each position that contained an empty space was checked against the finish puzzle to ensure they contained the same value. For example the first empty space on the boards above should have contained a 6. Once the solver was completed I then compared the value in this position to the value of the finished puzzle in the same position. This was done for each of the empty spaces on the board, as well as testing random positions that had not changed. This test exercised all parts of the code to ensure no errors would occur in any situation, and the test passed when I ran it.

I also decided to test whether other websites "Easy" Sudoku puzzles could be solved using this method. To test this I went on a website called websudoku (http://www.websudoku.com/?level=1&set_id=6120412189) and played an "Easy" puzzle until I had successfully completed it by hand. The unfinished puzzle and the completed version are shown below.



Unsolved Puzzle

```
{0,3,9, 0,2,0, 5,0,0,
 0,0,0, 8,0,6, 0,0,0,
 5,0,2, 0,3,0, 0,0,0,

 2,4,8, 0,0,5, 0,0,1,
 3,9,0, 2,0,1, 0,5,8,
 1,0,0, 7,0,0, 9,4,2,

 0,0,0, 0,7,0, 8,0,5,
 0,0,0, 3,0,8, 0,0,0,
 0,0,4, 0,6,0, 3,2,0};
```



Here is the puzzle. Good luck!

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

Easy Puzzle 6,120,412,189 -- Select a puzzle...

Download Web Sudoku Deluxe for Windows or Mac!

As you can see from the image above, the unsolved puzzle that was taken from the website contained 46 empty spaces. I created a new test in JUnit and created a puzzle that was the same as the unfinished puzzle seen above. Then I called the solver method and compared the results with the actual results of the puzzle. To test this I tested all of the empty positions in the unfinished game and then tested it against the values that were found in the completed version. I also tested a few that were provided in the unfinished puzzle to ensure that they had not changed. I conducted a final check to see whether the puzzle had been completed successfully or not. All of these tests exercised all parts of the code to ensure no errors would occur in any situation, and all of the tests passed when I ran them.

I also checked a medium difficulty puzzle from the same website in my program. It solved 5 of the empty positions and then it remained stuck, so this solver is only useable on Easy difficulty puzzles.

Chapter 6 – Sudoku Puzzle Generator

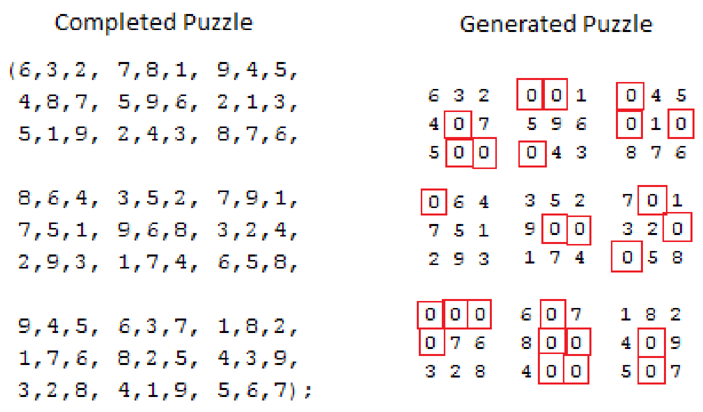
6.1 Implementation

The objective of this task was to generate a Sudoku puzzle by removing a number at random and using the method Check One Choice from a previous chapter. I first started researching for a way to generate a number at random between two given numbers. I found a class called Random which allows an input of a high and a low number that determines the lowest and highest number that you want your random number to be between.

I first started by creating a Random class, and creating two integer variables, one which held 0 and another which held 80. I then created a 'for loop' that iterates through 50 times. A method is called on the random class variable to create a random number, the method is shown below.

```
int R = r.nextInt(high-low)+low;      //generates random number
```

In this method the R variable is the Random variable that had been created, and the variables low and high were also declared at the top of the method where low is 0, and high is 80. The number that is created is stored within an integer variable. The variable is then used to check whether Check One Choice returns 0, which it should as there are no 0's on the board, so all of the items in the Boolean array are 'True'. If the result is 0 then it will store 0 in the array at the random number position. This is called the 'digging method' by removing random positions within the grid, Zhan([n.d.]). Once the 'for loop' has completed it will then print out the new generated puzzle with empty spaces to the terminal. An example of a finished puzzle, and the same puzzle once the generate method has been called can be seen below.



Completed Puzzle

```
{6,3,2,  7,8,1,  9,4,5,
 4,8,7,  5,9,6,  2,1,3,
 5,1,9,  2,4,3,  8,7,6,

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

 9,4,5,  6,3,7,  1,8,2,
 1,7,6,  8,2,5,  4,3,9,
 3,2,8,  4,1,9,  5,6,7};
```

Generated Puzzle

```
6 3 2   0 0 1   0 4 5
4 0 7   5 9 6   0 1 0
5 0 0   0 4 3   8 7 6

0 6 4   3 5 2   7 0 1
7 5 1   9 0 0   3 2 0
2 9 3   1 7 4   0 5 8

0 0 0   6 0 7   1 8 2
0 7 6   8 0 0   4 0 9
3 2 8   4 0 0   5 0 7
```

A full printout of the generate method removing numbers from the puzzle and changing them to empty spaces can be seen in Appendix 3 on page 30. As you can see from the example above, there are 25 empty spaces in the newly generated puzzle.

6.2 Testing

Although there is no way to actually create a test method in JUnit to ensure the empty spaces are being created, I manually tested this by using a printout method. The number of empty spaces varies as a random position is selected each time. I have tested the generate method on the same puzzle above 10 times and the number of empty spaces in each is as follows:

33, 29, 29, 31, 35, 26, 30, 28, 31, 28

From this test the average amount of empty spaces that are generated in each puzzle is 30. When the loop iterated through 60 times instead of the previous 50, the numbers that are removed will increase, but the solver method very rarely will not complete the puzzle. This is why I have chosen to stick with 50 as my loop number. Even though my generate method only removes around 30 positions of the puzzle, the solver method can still solve puzzles with higher amounts of empty spaces. This can be seen in the previous chapter where I tested an easy Sudoku puzzle from a website to see if my solver could solve the puzzle. A study was conducted by Zhan Research Group ([n.d.]) that determined how many cells should be removed for each difficulty. For an extremely difficult puzzle the puzzle should contain more than 50 cells that are filled, for an easy puzzle 36-49 cells must be filled, Zhan ([n.d.]). As you can see from my average above, my puzzle can generate between an Extremely Easy puzzle and an Easy Puzzle when being compared to the graph by Zhan ([n.d.]).

Although I could not use JUnit to test that the empty spaces were being created, I did test if the solver method would indeed solve a puzzle that had been created using the generate method. A full test plan for each of the methods that have been implemented can be seen in further detail by looking at Appendix 1 on page 28. To test this I created two puzzles that were exactly the same. I then called the generate method on one of the puzzles and straight after called the solve method. After the methods had been called I then compared random positions within the puzzle to ensure that the puzzles were the same after the solver method had solved the puzzle. Due to the fact that each generator selects random positions to change, I decided to test all of the positions 1-40 which is the first half of the puzzle. I then selected random numbers between the numbers 41-80, to see that the positions were also the same as the second puzzle. This test exercised all parts of the code to ensure no errors would occur in any situation, and all of the tests passed when I ran them.

Chapter 7 - Evaluation

In this chapter of my report I will be explaining my strengths and weaknesses throughout each task of my project followed with an overall project evaluation. This chapter will also include future work that is planned including extensions as well as improvements to the provided program.

7.1 Sudoku Completion Checker

For this task I feel that I did a great job at determining what methods would be needed in order to fully check that a puzzle had been completed successfully. Although many methods were needed for this task, they all worked together perfectly and gave an accurate result. I feel that there is probably a more accurate way of determining where each square region starts rather than hardcoding in the starting position, but with the time constraint I felt that it was better to just get methods working correctly even if the coding was not perfect or the most accurate way of doing so. This method also helped me to wrap my head around the different ways to loop around my project, for example iterating through a row, column, or square. This method changed my mind set throughout the rest of the puzzle and helped a lot with the implementation of mathematical equations required for other methods.

7.2 Generate Random Completed Puzzle

For this method I honestly feel that I am cheating and am not genuinely creating a "random" Sudoku puzzle. Due to time constraints I felt that this would be the easiest way of implementing a somewhat random puzzle state for the user to solve, and would not be highly noticeable as multiple flips can be made on the puzzle. Even though only one puzzle is inputted into the program, with the 10 different flip methods, if only one flip was made each time the puzzle was played 10 different puzzles could be created. If multiple flips were made on the inputted puzzle the possibilities are much greater.

Although the flip methods do indeed create new different puzzles by manipulating a finished puzzle, it is not totally random but instead only making small changes to the original puzzle. Even though the puzzle is not completely random, I feel that this would not be very noticeable to a user. The generator method also helps with hiding the fact that the puzzle is not random, as it removes random cell positions, so each game would have empty spaces in different positions at the beginning of each game.

If I had more time to do this project, I would have much rather implemented a proper algorithm that would create a Sudoku Puzzle that was indeed random. Even if this was not possible, I feel that a more sophisticated way of generating the flip methods instead of calling them yourself would have been more suitable for a user playing the game. More puzzles could have also been input and chosen at random with random flips called on the puzzle, but time did not allow for this.

7.3 One Possible Solution

This method I feel did its purpose but it contained a lot of code and could have possibly been separated into separate methods. I completed this task toward the end of the 12 weeks and ran out of time to enhance the current program. The method works correctly but could have also allowed for a separate return statement that would be used for the generate method. This method contained a lot of duplication for example the square checker parts, but this code was needed as the start of the squares cannot be found by a mathematical equation so it must be hardcoded in. I feel that I did a good job of determining what row and column the position lied on by using mathematical equations.

7.4 Sudoku Puzzle Solver

For this task I feel that I have done a great job at displaying the solve methods results. Instead of simply returning a solved puzzle with all of the empty spaces filled in, I instead decided to print out each step of the puzzle so the process of the solver can be seen step by step. I feel that this solver was coded perfectly for an Easy puzzle as the solver can even solve Easy puzzles from Sudoku Puzzle websites. I don’t really feel that I did anything wrong with this method, apart from the fact that it will only work on Easy puzzles. Due to time constraints I was only able to create a solve method for an Easy puzzle but not for a Medium or Difficult puzzle.

7.5 Sudoku Puzzle Generator

For this task I feel that I did a good job of generating a “random” puzzle. As I have stated in section 7.2, although the puzzle is not exactly a random puzzle, it is not very obvious that the same puzzle is being manipulated for each game. I feel that by using the random number to remove cell numbers it creates a more random effect since the same cell positions will not be removed with every game. I feel that this task was done very well and does indeed create an Easy puzzle for a user to solve. I feel that maybe the number of cell positions that are removed is somewhat low compared to other online websites that generate Sudoku puzzles. Many of those websites contain around 40 cells that have been removed where my program only removes around 30-35, Zhan ([n.d.]). I have noticed that if I increase the “for loop” iteration number to 60, the solver method will take a long time to solve the puzzle, and 50 seems to be the best iteration number for this method in order to generate a puzzle, that can be solved in an efficient time.

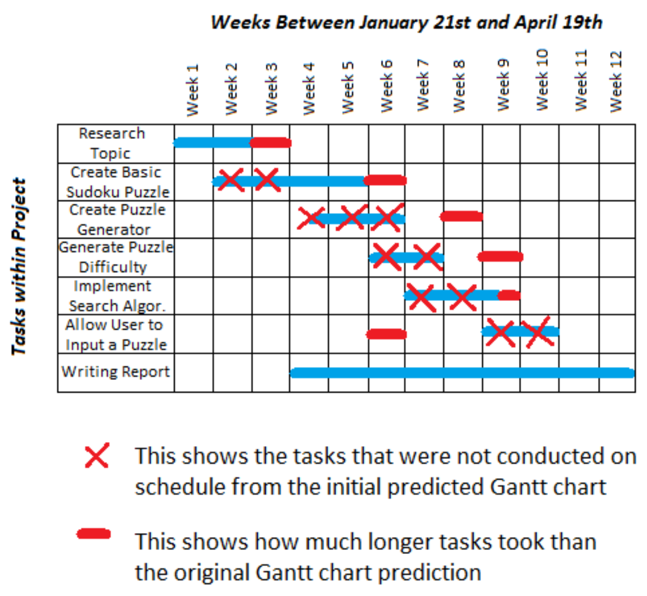
7.6 Overall Evaluation

I feel that the biggest problem I had was the time constraint of the project. My original project objectives were obviously too high for the time that I was given and I was only able to create and solve Easy puzzles. For the most part I feel that my program was coded well and provided little excess duplication throughout, although some tasks could have been implemented in a better way as you can see from the sections above. Overall I feel that the project was a success, as I feel that I achieved many of the initial objectives that I had decided, the only thing that is missing is the ability to change the difficulty of the puzzle.

7.6.1 Project Schedule

Another problem I noticed was that my original objectives were thought out how you would expect a game to be programmed, but the actual implementation of tasks turned out to be completely different from the original Gantt chart. Many of the tasks either took a longer or shorter time frame to be completed that I first anticipated. The diagram below shows the actual schedule of the project and how long each task took to implement and complete. Some of these methods have not yet been fully completed as they only work for an Easy puzzle. This would include the Check One Choice method, as well as the Solver and the Generator. This should be taken into account when looking at the following schedule document.

Sudoku Puzzle - Actual Gantt Chart



As you can see from the diagram above the research took longer than I had originally expected. It is unfortunate as a lot of time during this researching stage was spent looking into the different search algorithms that could be used for the Solve method, that did not end up being implemented. A different approach was used in order to solve the puzzle so no search algorithm was actually needed. You can also see that many of the tasks actually took less time than I originally first anticipated. Although the graph above shows nothing for the 7th week, I took most of this week to practice using JUnit and fully understand how my methods could be tested by using this feature. This took some getting used to but in the end it was worth spending the time learning the feature properly. Although my objectives and schedule changed quite a lot, I feel that I did indeed keep to schedule and in the end resulted in a working program that can be played on an Easy difficulty which is all that time allowed for.

7.7.2 Future Work

Improvements

- A generator method should be added that would generate more difficult puzzles for the user to solve. This would need to generate Medium and Hard difficulty puzzles.
- A solver method should be added that could solve harder puzzles. This would need to solve Medium and Hard difficulty puzzles. This method could possibly extend the already existing solver method, or possibly even replace it.
- A method should be created that would randomly call one of the flip methods by random selection. This method could possibly call multiple flip methods to generate a random puzzle.
- In addition to the above improvement, a method should also be created that would randomly select a completed puzzle which would result in a different completed puzzle that is being “manipulated” by the improvement above.

Extensions

- A user friendly GUI could be added to make the gameplay more enjoyable and make the game easier to play.

References

Bach, A. (2007). 'Advantage of Python'. [ONLINE] Available at: http://www.webdotdev.com/nvd/content/view/1073/. [Accessed 2nd February 2013]

Dawson, M. (2003) Python Programming for the absolute beginner Premier Press p.308

Dry Icons. (2009). 'A simple algorithm for generating Sudoku puzzles'. [ONLINE] Available at: http://dryicons.com/blog/2009/08/14/a-simple-algorithm-for-generating-sudoku-puzzles/. [Accessed 16th March 2013]

Forret, P. (2006). 'A Sudoku challenge generator'. [ONLINE] Available at: http://blog.forret.com/2006/08/a-sudoku-challenge-generator/. [Accessed 16th March 2013]

Matuszek, D. (2002). 'Backtracking'. [ONLINE] Available at: http://www.cis.upenn.edu/~matuszek/cit594-2012/Pages/backtracking.html. [Accessed 2nd February 2013]

MS. (2007). 'Java Advantages and Disadvantages'. [ONLINE] Available at: http://www.webdotdev.com/nvd/articles-reviews/java/java-advantages-and-disadvantages-1042.html. [Accessed 1st February 2013]

Patel, K. ([n.d.]). 'Most Significant Advantages of Java Language'. [ONLINE] Available at: http://www.streetdirectory.com/travel_guide/114362/programming/most_significant_advantages_of_java_language.html. [Accessed 1st February 2013]

Programmer World. (2012). 'Top Java IDE's'. [ONLINE] Available at: http://faq.programmerworld.net/programming/best-java-ide-review.html. [Accessed 2nd February 2013]

Rossum, G. (1997). 'Comparing Python to Other Languages'. [ONLINE] Available at: http://www.python.org/doc/essays/comparisons.html. [Last accessed 2nd February 2013]

Smith, D. (2005). 'So you thought Sudoku came from the Land of the Rising Sun'. [ONLINE] Available at: http://www.guardian.co.uk/media/2005/may/15/pressandpublishing.usnews. [Accessed 1st February 2013]

Spall, J. (2004). 'Introduction to Stochastic Search and Optimization'. [ONLINE] Available at: http://www.jhuapl.edu/spsa/PDF-SPSA/Handbook04_StochasticOptimization.pdf. [Accessed 3rd February 2013]

Webopedia. (2013). 'Brute Force'. [ONLINE] Available at: http://www.webopedia.com/TERM/B/brute_force.html. [Accessed 3rd February 2013]

Zhan Research Group. ([n.d.]). 'Sudoku Puzzles Generating: from Easy to Evil'. [ONLINE] Available at: http://zhangroup.aporc.org/images/files/Paper_3485.pdf. [Accessed 16th March 2013]

Bibliography

Bach, A. (2007). 'Advantage of Python'. [ONLINE] Available at: http://www.webdotdev.com/nvd/content/view/1073/. [Accessed 2nd February 2013]

Chen, Xi. ([n.d.]). 'Dancing Links'. [ONLINE] Available at: http://cgi.cse.unsw.edu.au/~xche635/dlx_sodoku/. [Accessed 17th March 2013]

Conceptis Puzzles. (2013). 'Sudoku History'. [ONLINE] Available at: http://www.conceptispuzzles.com/index.aspx?uri=puzzle/sudoku/history. [Accessed 1st February 2013]

Dawson, M. (2003) Python Programming for the absolute beginner Premier Press p.308. [Accessed 1st February 2013]

Dry Icons. (2009). 'A simple algorithm for generating Sudoku puzzles'. [ONLINE] Available at: http://dryicons.com/blog/2009/08/14/a-simple-algorithm-for-generating-sudoku-puzzles/. [Accessed 16th March 2013]

Eclipse. (2013). 'Eclipse IDE for Java Developers'. [ONLINE] Available at: http://www.eclipse.org/downloads/moreinfo/java.php. [Accessed 2nd February 2013]

Forret, P. (2006). 'A Sudoku challenge generator'. [ONLINE] Available at: http://blog.forret.com/2006/08/a-sudoku-challenge-generator/. [Accessed 16th March 2013]

Matuszek, D. (2002). 'Backtracking'. [ONLINE] Available at: http://www.cis.upenn.edu/~matuszek/cit594-2012/Pages/backtracking.html. [Accessed 2nd February 2013]

MS. (2007). ‘Java Advantages and Disadvantages’. [ONLINE] Available at: http://www.webdotdev.com/nvd/articles-reviews/java/java-advantages-and-disadvantages-1042.html. [Accessed 1st February 2013]

NetBeans. (2013). 'NetBeans IDE - The Smarter and Faster Way to Code'. [ONLINE] Available at: https://netbeans.org/features/index.html. [Accessed 2nd February 2013]

Patel, K. ([n.d.]). 'Most Significant Advantages of Java Language'. [ONLINE] Available at: http://www.streetdirectory.com/travel_guide/114362/programming/most_significant_advantages_of_java_language.html. [Accessed 1st February 2013]

Pbl. (2010). 'Sudoku I - Basic Tools'. [ONLINE] Available at: http://www.dreamincode.net/forums/topic/195833-sudoku-i-basic-tools/. [Accessed 17th March 2013]

Programmer World. (2012). 'Top Java IDE's'. [ONLINE] Available at: http://faq.programmerworld.net/programming/best-java-ide-review.html. [Accessed 2nd February 2013]

Rees, G. (2007). 'Zendoku Puzzle Generation'. [ONLINE] Available at: http://garethrees.org/2007/06/10/zendoku-generation/. [Accessed 17th March 2013]

Rossum, G. (1997). 'Comparing Python to Other Languages'. [ONLINE] Available at: http://www.python.org/doc/essays/comparisons.html. [Last accessed 2nd February 2013]

Smith, D. (2005). 'So you thought Sudoku came from the Land of the Rising Sun'. [ONLINE] Available at: http://www.guardian.co.uk/media/2005/may/15/pressandpublishing.usnews. [Accessed 1st February 2013]

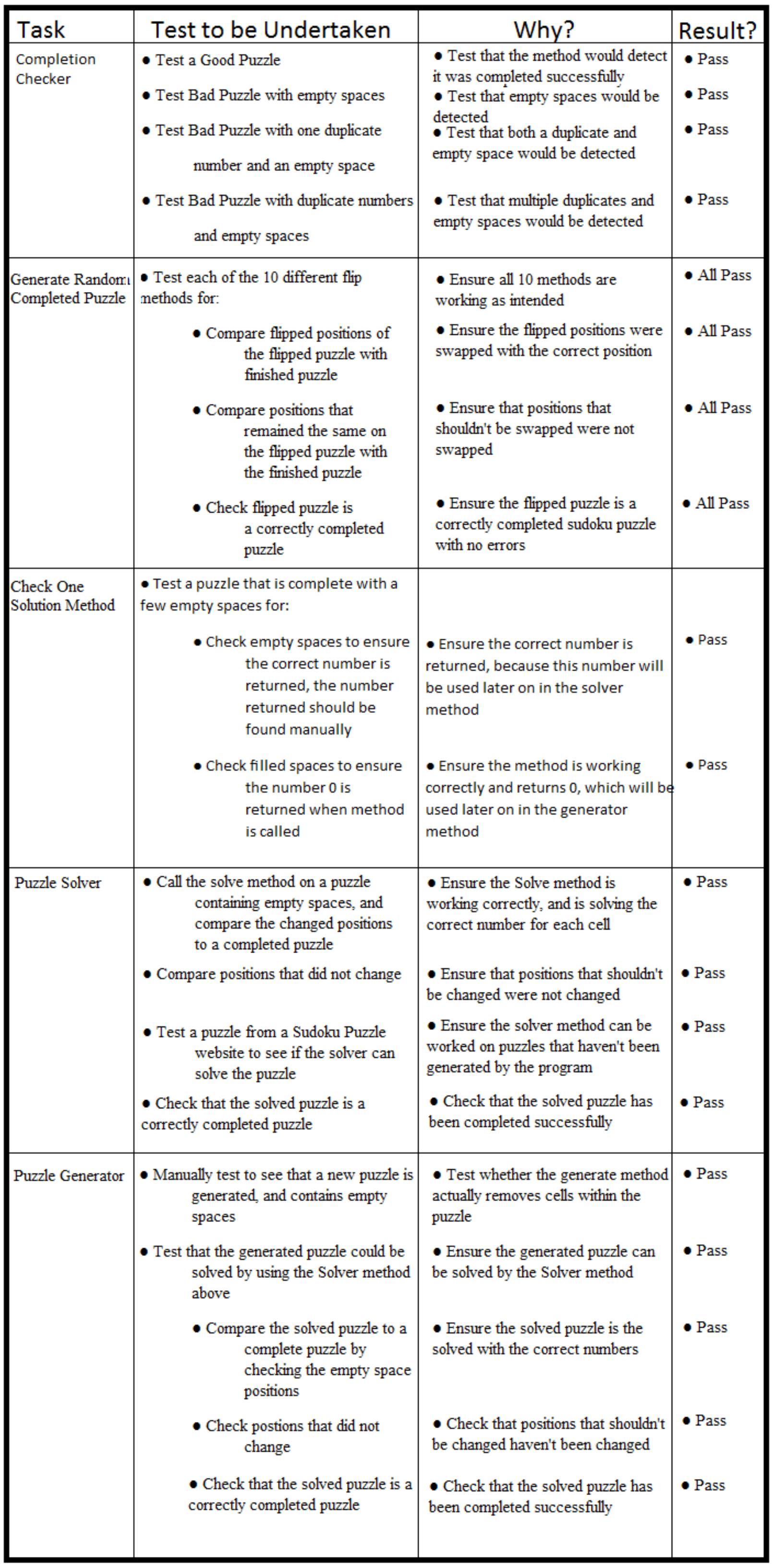
Spall, J. (2004). 'Introduction to Stochastic Search and Optimization'. [ONLINE] Available at: http://www.jhuapl.edu/spsa/PDF-SPSA/Handbook04_StochasticOptimization.pdf. [Accessed 3rd February 2013]

Webopedia. (2013). 'Brute Force'. [ONLINE] Available at: http://www.webopedia.com/TERM/B/brute_force.html. [Accessed 3rd February 2013]

Zhan Research Group. ([n.d.]). 'Sudoku Puzzles Generating: from Easy to Evil'. [ONLINE] Available at: http://zhangroup.aporc.org/images/files/Paper_3485.pdf. [Accessed 16th March 2013]

Appendices

Appendix 1 - Test Plan Document: This shows the test plan that was used in order to thoroughly test each method within the program



Task	Test to be Undertaken	Why?	Result?
Completion Checker	• Test a Good Puzzle	• Test that the method would detect it was completed successfully	• Pass
	• Test Bad Puzzle with empty spaces	• Test that empty spaces would be detected	• Pass
	• Test Bad Puzzle with one duplicate number and an empty space	• Test that both a duplicate and empty space would be detected	• Pass
	• Test Bad Puzzle with duplicate numbers and empty spaces	• Test that multiple duplicates and empty spaces would be detected	• Pass
Generate Random Completed Puzzle	• Test each of the 10 different flip methods for:	• Ensure all 10 methods are working as intended	• All Pass
	• Compare flipped positions of the flipped puzzle with finished puzzle	• Ensure the flipped positions were swapped with the correct position	• All Pass
	• Compare positions that remained the same on the flipped puzzle with the finished puzzle	• Ensure that positions that shouldn't be swapped were not swapped	• All Pass
	• Check flipped puzzle is a correctly completed puzzle	• Ensure the flipped puzzle is a correctly completed sudoku puzzle with no errors	• All Pass
Check One Solution Method	• Test a puzzle that is complete with a few empty spaces for:		
	• Check empty spaces to ensure the correct number is returned, the number returned should be found manually	• Ensure the correct number is returned, because this number will be used later on in the solver method	• Pass
	• Check filled spaces to ensure the number 0 is returned when method is called	• Ensure the method is working correctly and returns 0, which will be used later on in the generator method	• Pass
Puzzle Solver	• Call the solve method on a puzzle containing empty spaces, and compare the changed positions to a completed puzzle	• Ensure the Solve method is working correctly, and is solving the correct number for each cell	• Pass
	• Compare positions that did not change	• Ensure that positions that shouldn't be changed were not changed	• Pass
	• Test a puzzle from a Sudoku Puzzle website to see if the solver can solve the puzzle	• Ensure the solver method can be worked on puzzles that haven't been generated by the program	• Pass
	• Check that the solved puzzle is a correctly completed puzzle	• Check that the solved puzzle has been completed successfully	• Pass
Puzzle Generator	• Manually test to see that a new puzzle is generated, and contains empty spaces	• Test whether the generate method actually removes cells within the puzzle	• Pass
	• Test that the generated puzzle could be solved by using the Solver method above	• Ensure the generated puzzle can be solved by the Solver method	• Pass
	• Compare the solved puzzle to a complete puzzle by checking the empty space positions	• Ensure the solved puzzle is the solved with the correct numbers	• Pass
	• Check postions that did not change	• Check that positions that shouldn't be changed haven't been changed	• Pass
	• Check that the solved puzzle is a correctly completed puzzle	• Check that the solved puzzle has been completed successfully	• Pass

Appendix 2 - Solve Method Printout

```
 Sudoku Puzzle

 6 0 2 7 8 1 9 0 5
 4 8 0 5 9 6 2 0 0
 0 0 9 2 4 3 8 7 6
 0 0 0 3 5 2 7 0 1
 0 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

26
Position: 1 -> 3

 Sudoku Puzzle

 6 3 2 7 8 1 9 0 5
 4 8 0 5 9 6 2 0 0
 0 0 9 2 4 3 8 7 6
 0 0 0 3 5 2 7 0 1
 0 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

25
Position: 7 -> 4

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 0 5 9 6 2 0 0
 0 0 9 2 4 3 8 7 6
 0 0 0 3 5 2 7 0 1
 0 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

24
Position: 11 -> 7

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 0 0
 0 0 9 2 4 3 8 7 6
 0 0 0 3 5 2 7 0 1
 0 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

23
Position: 16 -> 1

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 0
 0 0 9 2 4 3 8 7 6
 0 0 0 3 5 2 7 0 1
 0 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

22
Position: 17 -> 3

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 0 0 9 2 4 3 8 7 6
 0 0 0 3 5 2 7 0 1
 0 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7
```

```
21
Position: 18 -> 5

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 0 9 2 4 3 8 7 6
 0 0 0 3 5 2 7 0 1
 0 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

20
Position: 19 -> 1

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 0 0 0 3 5 2 7 0 1
 0 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

19
Position: 27 -> 8

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 0 3 5 2 7 0 1
 0 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

18
Position: 29 -> 4

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 0 1
 0 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

17
Position: 34 -> 9

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 0 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

16
Position: 36 -> 7

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 0 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7
```

```
15
Position: 40 -> 6

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 0 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

14
Position: 43 -> 2

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 0 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

13
Position: 46 -> 9

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 0 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

12
Position: 47 -> 3

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 0 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

11
Position: 49 -> 7

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 0 0 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

10
Position: 56 -> 5

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 0 5 0 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7
```

```
9
Position: 57 -> 6

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 0 5 6 3 7 0 0 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

8
Position: 61 -> 8

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 0 5 6 3 7 0 8 0
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

7
Position: 62 -> 2

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 0 5 6 3 7 0 8 2
 1 7 6 8 2 5 0 3 9
 0 0 8 4 1 9 0 6 7

6
Position: 69 -> 4

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 0 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 0 0 8 4 1 9 0 6 7

5
Position: 72 -> 3

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 0 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 0 8 4 1 9 0 6 7

4
Position: 73 -> 2

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 0 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 0 6 7
```

```
3
Position: 78 -> 5

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 0 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 0 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

2
Position: 28 -> 6

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 6 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 0 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

1
Position: 55 -> 4

 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 6 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

0
Position: 60 -> 1

 Sudoku Puzzle

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

Solved Puzzle
 Sudoku Puzzle

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

Appendix 3 - Generator Method Printout

```
0
60 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 6 4 3 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

1
30 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 8 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 6 4 0 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

2
10 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 0 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 6 4 0 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 9 3 1 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

3
46 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 0 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 6 4 0 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 0 3 1 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

4
48 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 0 7 5 9 6 2 1 3
 5 1 9 2 4 3 8 7 6
 8 6 4 0 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

5
22 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 0 7 5 9 6 2 1 3
 5 1 9 2 0 3 8 7 6
 8 6 4 0 5 2 7 9 1
 7 5 1 9 6 8 3 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7
```

```
6
42 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 0 7 5 9 6 2 1 3
 5 1 9 2 0 3 8 7 6
 8 6 4 0 5 2 7 9 1
 7 5 1 9 6 8 0 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

7
19 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 2 7 9 1
 7 5 1 9 6 8 0 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

8
32 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 1
 7 5 1 9 6 8 0 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 3 2 8 4 1 9 5 6 7

9
72 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 1
 7 5 1 9 6 8 0 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

10
33 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 9 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 1
 7 5 1 9 6 8 0 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

11
6 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 1
 7 5 1 9 6 8 0 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7
```

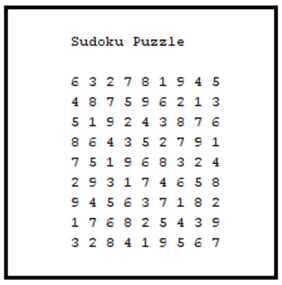
```
12
36 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 1
 0 5 1 9 6 8 0 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

13
41 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 1
 0 5 1 9 6 0 0 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

14
38 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 1
 0 5 0 9 6 0 0 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 7 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

15
64 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 1
 0 5 0 9 6 0 0 2 4
 2 0 3 0 7 4 6 5 8
 9 4 5 6 3 7 0 8 2
 1 0 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

16
54 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 1
 0 5 0 9 6 0 0 2 4
 2 0 3 0 7 4 6 5 8
 0 4 5 6 3 7 0 8 2
 1 0 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

17
44 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 1
 0 5 0 9 6 0 0 2 0
 2 0 3 0 7 4 6 5 8
 0 4 5 6 3 7 0 8 2
 1 0 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7
```

```
18
58 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 1
 0 5 0 9 6 0 0 2 0
 2 0 3 0 7 4 6 5 8
 0 4 5 6 0 7 0 8 2
 1 0 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

19
35 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 0
 0 5 0 9 6 0 0 2 0
 2 0 3 0 7 4 6 5 8
 0 4 5 6 0 7 0 8 2
 1 0 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

20
53 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 0
 0 5 0 9 6 0 0 2 0
 2 0 3 0 7 4 6 5 8
 0 4 5 6 0 7 0 8 2
 1 0 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

21
20 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 0
 0 5 0 9 6 0 0 2 0
 2 0 3 0 7 4 6 5 8
 0 4 5 6 0 7 0 8 2
 1 0 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

22
52 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 6
 8 6 4 0 5 0 7 9 0
 0 5 0 9 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 5 6 0 7 0 8 2
 1 0 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

23
26 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 9 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 5 6 0 7 0 8 2
 1 0 6 8 2 5 4 3 9
 0 2 8 4 1 9 5 6 7
```

```
24
66 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 9 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 5 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

25
56 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 9 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

26
39 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

27
62 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

28
40 -> 0
 Sudoku Puzzle

 6 3 2 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

29
2 -> 0
 Sudoku Puzzle

 6 3 0 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

30
47 -> 0
 Sudoku Puzzle

 6 3 0 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

31
1 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 4 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

32
9 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

33
35 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

34
58 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 7 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

35
11 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

36
35 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 1 3
 5 0 9 2 0 3 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

37
23 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 1 3
 5 0 9 2 0 0 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

38
68 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 1 3
 5 0 9 2 0 0 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

39
47 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 1 3
 5 0 9 2 0 0 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

40
17 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 1 0
 5 0 9 2 0 0 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

41
16 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 0 0
 5 0 9 2 0 0 8 7 0
 8 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

42
27 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 0 0
 5 0 9 2 0 0 8 7 0
 0 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

43
58 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 0 0
 5 0 9 2 0 0 8 7 0
 0 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

44
77 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 0 0
 5 0 9 2 0 0 8 7 0
 0 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

45
2 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 0 0
 5 0 9 2 0 0 8 7 0
 0 6 4 0 5 0 7 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

46
33 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 0 0
 5 0 9 2 0 0 8 7 0
 0 6 4 0 5 0 0 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 9
 0 2 8 4 1 9 5 6 7

47
71 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 0 0
 5 0 9 2 0 0 8 7 0
 0 6 4 0 5 0 0 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 0
 0 2 8 4 1 9 5 6 7

48
68 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 0 0
 5 0 9 2 0 0 8 7 0
 0 6 4 0 5 0 0 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 0
 0 2 8 4 1 9 5 6 7

49
10 -> 0
 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 0 0
 5 0 9 2 0 0 8 7 0
 0 6 4 0 5 0 0 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 0
 0 2 8 4 1 9 5 6 7

 Sudoku Puzzle

 6 0 0 7 8 1 0 4 5
 0 0 0 5 9 6 2 0 0
 5 0 9 2 0 0 8 7 0
 0 6 4 0 5 0 0 9 0
 0 5 0 0 6 0 0 2 0
 2 0 3 0 7 4 6 0 8
 0 4 0 6 0 7 0 8 2
 1 0 6 0 2 5 4 3 0
 0 2 8 4 1 9 5 6 7
```

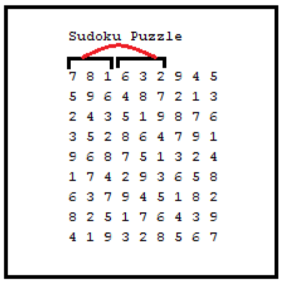
Appendix 4: Each of the 10 Flip Method Printouts

Origional



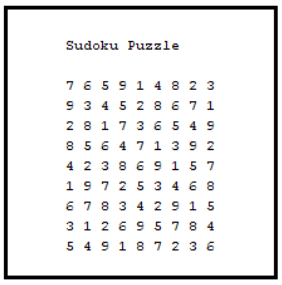
```
Sudoku Puzzle

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

FlipPuzzleColumn1



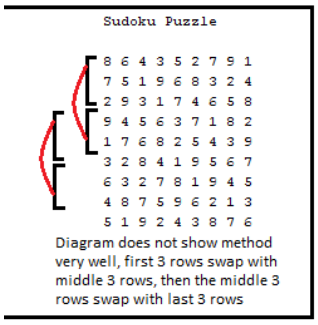
```
Sudoku Puzzle

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

FlipPuzzleHorizontal Vertical



```
Sudoku Puzzle

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

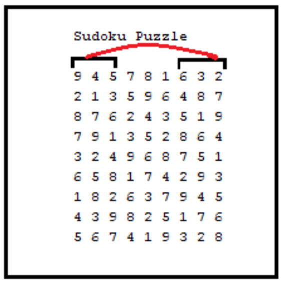
FlipPuzzleRows1



```
Sudoku Puzzle

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

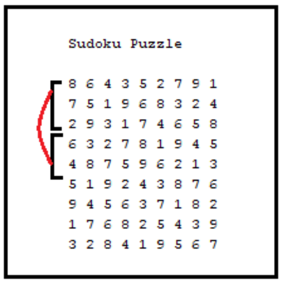
Diagram does not show method very well, first 3 rows swap with middle 3 rows, then the middle 3 rows swap with last 3 rows

FlipPuzzleColumn2



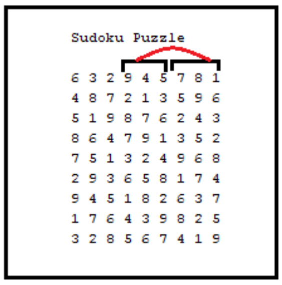
```
Sudoku Puzzle

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

FlipPuzzleRows2



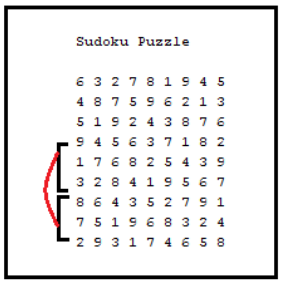
```
Sudoku Puzzle

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

FlipPuzzleColumn3



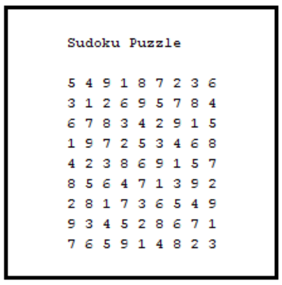
```
Sudoku Puzzle

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

FlipPuzzleRows3



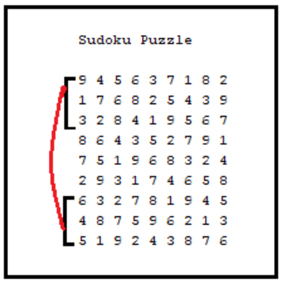
```
Sudoku Puzzle

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

FlipPuzzle Horizontal



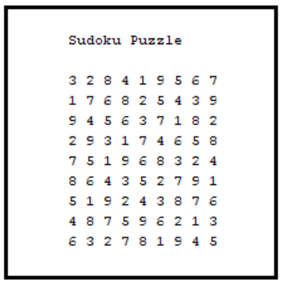
```
Sudoku Puzzle

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

FlipPuzzleRows4



```
Sudoku Puzzle

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

FlipPuzzle Vertical



```
Sudoku Puzzle

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

Appendix 5

The full source code for this project can be seen in the folder ‘Sudoku Puzzle‘.