DECLARATION

I confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the words of other authors in any form e.g., ideas, equations, figures, text, tables, programs etc. are properly acknowledged. A list of references employed is included.

(Heriot-Watt Computer Science MSc Programme Handbook, Edinburgh campus, 2012)

Signed

Date
ABSTRACT

The aim of this project is to develop an application in Java to generate well-formed Sudoku puzzles, to solve Sudoku puzzles and to aid the player when trying to solve its puzzles. Several algorithms are examined and some are tested to find the fastest, most reliable one that generates puzzles with a unique solution. The final product provides an entertaining interface and useful tools to aid the user while solving puzzles.
**Contents**

DECLARATION ........................................................................................................... 1  
ABSTRACT .................................................................................................................. 2  
Contents ..................................................................................................................... 3  
List of Figures and Tables ......................................................................................... 6  
1.0. Introduction ........................................................................................................ 9  
  1.1. What is SuDoKu ................................................................................................. 9  
  1.2. Dissertation Aim and Objectives ..................................................................... 10  
  1.3. Software Development Methodology .................................................................. 10  
  1.5. Dissertation Outline ......................................................................................... 11  
2.0. Literature Review ................................................................................................. 13  
  2.1. Computer Games and Puzzles .......................................................................... 13  
  2.2. The Mathematics of Sudoku ............................................................................ 14  
  2.3. Sudoku Terminology .......................................................................................... 14  
  2.4. Sudoku Solving and Generating Algorithms ...................................................... 15  
    2.4.1. Pencil and Paper Algorithm .......................................................................... 15  
    2.4.2. Search Based Algorithms ............................................................................. 16  
    2.4.3. Brute-Force and Backtracking Algorithm ................................................... 17  
    2.4.4. Solve Sudoku as a Constraint Problem Algorithm ..................................... 18  
  2.5. Generating Sudoku Puzzle .............................................................................. 19  
  2.6. Sudoku Difficulty Ranking .............................................................................. 19  
  2.7. Conclusion ......................................................................................................... 20  
3.0. Professional, Legal and Ethical Issues .................................................................. 22  
  3.1. Professional and Ethical Issues ........................................................................ 22  
  3.2. Legal Issues ...................................................................................................... 23  
4.0. Project Plan .......................................................................................................... 25  
  4.1. Timeline ............................................................................................................. 25
4.2. Evaluation Plan ................................................................. 26
4.3. Risk Assessment ............................................................... 26
5.0. Requirements Analysis ....................................................... 29
  5.1. Introduction ................................................................. 29
  5.2. Aims ............................................................................. 29
  5.3. Objectives ................................................................. 29
  5.4. User and User Tasks Analysis ........................................... 29
    5.4.1 Getting to Know the Tasks .......................................... 30
    5.4.2 Main Tasks Description ................................................ 31
  5.5. Functional Requirements ................................................ 32
  5.6. Non-functional Requirements .......................................... 34
  5.7. Functional Modelling .................................................... 35
    5.7.1 Use case analysis ....................................................... 35
6.0. Object-Oriented Design ..................................................... 38
  6.2 Structural Modelling ....................................................... 38
    6.2.1 Identify Classes, Attributes and Operations ....................... 38
    6.2.2 Association Between the classes .................................... 39
    6.2.3 Class Diagrams ........................................................... 40
7.0. User Interface Design ....................................................... 43
  7.1. Introduction ................................................................. 43
  7.2. Guidelines for Interface design ........................................ 43
  7.3. Interface Layout .......................................................... 44
  7.4. Graphical User Interface ................................................ 44
    7.4.1. Main Screen Design .................................................. 45
8.0. Implementation ............................................................... 48
  8.1. Introduction ................................................................. 48
  8.2. Technologies Used for Implementing the Application ............... 48
## 8.3 Building Classes

8.4 implementing the functional requirements

<table>
<thead>
<tr>
<th>8.4.1 Solving the puzzle</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.4.2 Generate new puzzle</td>
<td>52</td>
</tr>
<tr>
<td>8.4.3 Puzzle board and gameplay</td>
<td>53</td>
</tr>
<tr>
<td>8.4.4 Saving and Loading puzzle</td>
<td>56</td>
</tr>
</tbody>
</table>

## 9.0 Testing and Evaluation

<table>
<thead>
<tr>
<th>9.1 Testing Levels</th>
<th>58</th>
</tr>
</thead>
<tbody>
<tr>
<td>9.1.1 Unit Test</td>
<td>58</td>
</tr>
<tr>
<td>9.2 Evaluation Plan</td>
<td>60</td>
</tr>
<tr>
<td>9.2.1 Usability Attributes</td>
<td>60</td>
</tr>
<tr>
<td>9.2.2 Empirical Evaluation</td>
<td>60</td>
</tr>
</tbody>
</table>

## 10.0 Conclusions and Future Work

<table>
<thead>
<tr>
<th>10.1 Critical Assessment</th>
<th>67</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.2 Lessons Learnt from this Dissertation</td>
<td>68</td>
</tr>
<tr>
<td>10.3 Future Work</td>
<td>69</td>
</tr>
<tr>
<td>10.4 Conclusions</td>
<td>69</td>
</tr>
</tbody>
</table>

## References

| References | 71 |

## APPENDICES

| APPENDICES | 72 |
List of Figures and Tables

9×9 Sudoku grid .................................................................................................................. 9
20×20 Alphadoku .................................................................................................................. 9
Waterfall Model .................................................................................................................. 10
Sudoku terminology .......................................................................................................... 14
A box with a preemptive set .............................................................................................. 15
Brute-force flowchart diagram ......................................................................................... 17
Row/Column matching ........................................................................................................ 18
Rows/Blocks matching ....................................................................................................... 18
Project Timeline table ....................................................................................................... 25
Project Gantt chart ............................................................................................................ 25
Risk Assessment table ....................................................................................................... 26
The Task Analysis Diagram .............................................................................................. 30
The Main Task Analysis Diagram ...................................................................................... 31
Main Actors ........................................................................................................................ 35
Use case diagram ............................................................................................................... 36
Sudoku Package Classes ................................................................................................... 38
GUI Package Classes ......................................................................................................... 39
Association between Classes ............................................................................................ 39
Sudoku Package Class Diagram ......................................................................................... 40
GUI Package Class Diagram .............................................................................................. 41
The Golden Rules of Interface Design Table ..................................................................... 43
User Interface Layout ............................................................... 44
Gameplay screen description .................................................. 45
Custom puzzle screen description ........................................... 46
Create new puzzle screen description .................................... 46
Puzzle Class Diagram ............................................................. 48
Cell Class Diagram .................................................................. 50
BackTrackSolver Class Diagram ............................................ 50
Search Tree .............................................................................. 51
PuzzleGenerator Class Diagram ............................................ 52
PuzzleJPaned Class Diagram ................................................ 53
CellJTextField Class Diagram ................................................. 55
Test Case 1 Table .................................................................... 59
Test Case 2 Table .................................................................... 60
Test Case 3 Table .................................................................... 60
Test Case 4 Table .................................................................... 60
Questionnaire Analysis Table .................................................. 62
Questionnaire First Page .......................................................... 64
Questionnaire Second Page ...................................................... 65
Questionnaire Summary .......................................................... 66
Critical Assessment Table ........................................................ 68
CHAPTER 1

INTRODUCTION
1.0. Introduction

Although it firstly appeared in the late 70s, Sudoku became popular only within the last decade. In Britain, it became widespread in late 2004, when the Times newspaper introduced the puzzle to its readers for the first time calling it "Su Doku". It soon became a front-page feature in almost all newspapers and was even presented in some Teletext services.

1.1. What is SuDoKu

"Sjii wa dokushini ni kagiru" is the original name and it is of Japanese origin, now known as Sudoku (pronounced Soo Doe Koo) and meaning number game."(Intelm, 2005). It is also sometimes named "Number Place". The game consists of a 9×9 number grid, which is divided into nine major squares; each square contains nine cells (figure 1). The goal of the game is to fill up all the cells with the numbers 1 through 9 under the following rules:

1. Each row has the numbers from 1 to 9 without repeating any number.
2. Each column has the numbers from 1 to 9 without repeating any number.
3. Each 3×3 sub-square has the numbers from 1 to 9 and no number repeats in any square.

Some numbers are given in the puzzle, and the problem is to find the remaining numbers. Each puzzle should have only a unique solution and puzzles can have different ranks of difficulty.

There are several established variations of Sudoku; the grid for example can be altered to the range 4×4 to 16×16. In addition, some types reach 25×25 like Alphadoku and Alphadoku X. Other variations of this puzzle make many changes to the grid like Samurai Sudoku and Killer Sudoku. In this project only the standard (9×9) Sudoku will be considered, though the option of choosing 4×4, 6×6, 8×8, 9×9, 12×12 or 16×16 grids may be supported as well in the final version of the program.
1.2. Dissertation Aim and Objectives

The main aim of this dissertation is to develop a computer based Sudoku program that generates Sudoku puzzles and aids the player to solve them. These puzzles must be well-formed and solving them using the developed program should be entertaining. The core objectives of this dissertation are highlighted below:

- To investigate the different Sudoku generating and solving algorithms and programs out there to gain a full understanding of the problems relevant to developing a Sudoku Generator and Solver application.
- To specify user and system requirements to meet a user’s needs with regard to the application.
- To design the Sudoku Generator and Solver application.
- To implement a Sudoku Generator and Solver application in accordance with the requirements and design.
- To test and evaluate the developed application to ensure that it is usable and efficient.

1.3. Software Development Methodology

A waterfall model of development with iterative cycles was followed in developing this project. The main reason behind using this approach is that it is flexible in the sense that the developer can sequentially move from one stage to the next stage and can return to any previous phase if necessary. In addition, it documents every stage of the development of the system that helps to reduce any confusion in following system requirements as well as helps to complete the system in the expected time. To accomplish this approach successfully, the developer needs to finish the stage which working on before moving the next stage. (Dennis & Haley, 2005).
1.5. Dissertation Outline

The dissertation is in ten chapters, including this introductory chapter.

- **Chapter 2:**
  Presents background research and undertakes a literature review related to the Sudoku Generator and Solver program.

- **Chapter 3:**
  Presents professional, legal and ethical issues.

- **Chapter 4:**
  Describes the project plan and Gantt chart. It also gives the risk assessment of the project and an overview of the evaluation plan.

- **Chapter 5:**
  Carries out the requirements analysis activities for the Sudoku Generator and Solver program.

- **Chapter 6:**
  Presents the Object-Oriented design for the Sudoku Generator and Solver program.

- **Chapter 7:**
  Present different aspects for designing the user interface.

- **Chapter 8:**
  Describes the implementation of the Sudoku Generator and Solver program.

- **Chapter 9:**
  Presents the testing and evaluation for the Sudoku Generator and Solver program.

- **Chapter 10:**
  Gives conclusions on the project and suggests future work that can be done in order to improve the application.
CHAPTER 2

LITERATURE REVIEW
2.0. Literature Review

Although there have been only a few books that discuss generating Sudoku puzzles or programming published since this game became popular, many papers and articles are available that study and inspect the mathematics, algorithms, difficulty ranking and a lot more about Sudoku puzzles. Most of this literature review will focus on these papers and articles due to the lack of published books on this subject.

2.1. Computer Games and Puzzles

Most people play puzzles to relax or sometimes to challenge their mind and logic skills. Computers make playing these puzzles more fun, more varied and much easier to access anytime and anywhere.

Computer Puzzles appeared very early in the history of Computer Science, even before the dawn of personal computers, when programmers tried to develop games and puzzles as computer programs. The first computer game was Tic-Tac-Toe. It was developed as a PhD thesis at the University of Cambridge in 1952 and ran on its mainframe. (Aarseth, 2005). Computer games development spread rapidly and become more popular a decade later.

So what is a Computer Game? According to Salen & Zimmerman (2003), "a game is a system in which players engage in an artificial conflict, defined by rules, that results in a quantifiable outcome". Moreover, what makes Sudoku so special as a computer game? Well, most people agree that "Sudoku puzzles not only are fun and challenging to solve, they offer good training for thinking logically." (Lee, 2006). Several researchers have found that solving Sudoku puzzles on a regular basis is within the top ten alternative and non-traditional ways to improve brainpower.

In addition, anyone can easily play this puzzle from kids to adults. Naturally, to be able to generate or solve Sudoku puzzle programmatically, we must first understand the mathematics behind Sudoku.
2.2. The Mathematics of Sudoku

It is easily noticed that Sudoku a grid can is a kind of Latin square. A Latin square is a \( n \times n \) matrix filled with \( n \) symbols, each occurring once in each row and each column. (Mikheev). And based on that fact, Jarvis & Felgenhauer found that the number of the possible different Sudoku grids is \( 6.671 \times 10^{21} \), 'The exact number is 6670903752021072936960'. (Jarvis & Felgenhauer, 2006). And to understand this puzzle mathematically Simonis (2005) defines it as “A Sudoku square of order \( n \) consists of \( n^4 \) variables formed into a \( n^2 \times n^2 \) grid with values from 1 to such \( n^2 \) that the entries in each row, each column and each of the \( n^2 \) major \( n \times n \) blocks are all different.”

Another essential mathematical issue for Sudoku is “what is the minimal number needed of initial elements in 'the starting set' to generate a puzzle with unique solution?” Although this question has not been mathematically answered yet, there is almost universal agreement that it is 17 as no valid Sudoku puzzle exists with fewer number. (Lee, 2006)

2.3. Sudoku Terminology

Much literature on Sudoku is available on the internet with almost standardized terminology. The following terminology shall be used throughout this project:

- The \( 3 \times 3 \) sub grids will be referred as Blocks, so there are 9 blocks on the Sudoku grid. "Block B is shaded with green in figure 4"
- Each row and column has a number starting from 1 to 9
- A Cell is the smallest unit in the puzzle and it represents a square, which can only have single number in it. It is referred to by (row, column) “C in figure 4 is cell (6,7)".
- The set of cells that are contained within a column, row and block of a certain cell will be referred as the Sector of that cell. "The sector of cell C is shaded with grey in figure 4".
2.4. Sudoku Solving and Generating Algorithms

There are many approaches on how to solve a Sudoku puzzle. Solving the puzzle is important because to generate it, we need to solve the generated puzzle to make sure that there is only a unique solution for it. Most of the following algorithms will be examined to find the fastest and most efficient way to generate a well-formed and entertaining Sudoku puzzle. Our implementation could use more than one algorithm to generate or solve a puzzle.

2.4.1. Pencil and Paper Algorithm

Crook (2009) carried out a study and developed an algorithm that can solve any Sudoku puzzle the same way we solve it using pencil and paper. Moreover, the algorithm is designed to solve diabolical Sudoku puzzles. This algorithm is very useful to rank the puzzle difficulty or to explain the next step or provide a hint to the user. The theory is developed on the basis of matching the number concept. It matches numbers through the cells of the board. Crook (2009) uses the preemptive sets term for the matching number concept and this theorem can be applied to Sudoku boards of various sizes. This tool is used to solve Sudoku puzzles up to the point that either it finds a solution or it requires some more steps, which may be achieved by selecting one of two or more numbers in a random manner from the markup of an empty cell.

A preemptive set is composed of numbers in a set starting from 1 to 9 and is a set of size m where m is equal or bigger than 2 and equal or smaller than 9. These sets of m cells are filled by either the m numbers or a subset of m numbers. The tool has the feature that the distribution of m numbers across the m cells will not be known when the preemptive set is discovered. It will be discovered along with the progress in the solution of the puzzle.
In general, there are some steps, which this algorithm follows in order to solve Sudoku puzzles:

1. Finding all the forced numbers in the puzzle.
2. Marking up the puzzle.
3. “Searching for preemptive sets in all rows and columns continuously, for each new preemptive set boxes-taking appropriate crossout action will applied” (Crook, 2009).
4. When it comes to this step, if the solution is found then it is the end of the algorithm, if not, a random choice is applied and the algorithm returns back and starts again from step 3.

2.4.2. Search Based Algorithms

There are many search-based algorithms that can be applied to the Sudoku solving problem, Jones and Roach (2006) argue that the best results can be obtained using Limited Discrepancy Search. In this search technique, an objective function is employed to ascend the search space, “which make the optimal decisions on which move to make next, always choosing the highest scoring successor in the entire neighbourhood of the current state.” (Jones & Roach, 2006).

The algorithm he used is like the following:

- Start by making the current state into the initial state.
- While a solution is not found or iteration cannot change the current state, do
  - Best-successor becomes a state.
  - Iterate all operators that could be applied to current state and:
    - Determine a successor to current-state by applying the operator.
    - Evaluate successor. If a solution is found return, else if the successor is better than the best-successor let it become the best-successor.
  - If the best successor is better than the current state, change the current state to best-successor.

The key for this algorithm to work is to find a good objective function that measures the difference between the given state and the goal state, which is the solution. Jones & Roach (2006) defined this function as:

\[
f_2 = \sum_{g_{pq} \in G} \left| \left\{ i \mid a_{iq} = g_{pq}, i = 1..9, i \neq p \right\} \cup \left\{ j \mid a_{pj} = g_{pq}, j = 1..9, j \neq q \right\} \right|
\]

Where G is the initial numbers given, \( g_{pq} \) at row p, column q.
2.4.3. Brute-Force and Backtracking Algorithm

Brute-force it is the simplest way to solve a Sudoku puzzle, though it is not the most efficient way. It calculates all possible answers, and sees which one works by trying to put all possible numbers in each unsolved cell until a valid solution is found. When using backtracking with brute-force it takes just few microseconds to solve the puzzle on ordinary hardware today.

One disadvantage of this algorithm that it can’t explain the solution. It is nothing like a human solver. Also it is very inefficient to use this technique to generate a puzzle. The simplest version for this algorithm is as follows (Lee, 2006)

1. Select a first cell.
2. Calculate the possible numbers for the selected cell.
3. Place a number on the selected cell.
4. Check the sector and the block of the cell to see if the number is O.K.
5. If it is O.K select another cell and go to 1.
6. If the number is not allowed, select the second possible number.
7. If all possible numbers do not work leave the cell blank and go back to the previous cell and select the second possible number.

This algorithm solves any **solvable** puzzle rapidly. It is also an efficient way to find if a puzzle has more than one solution by treating the first solution as if it is a failure to find a non-conflicting solution for the last empty cell. If a different solution to the first solution is then found, then the first solution is not unique.

![Figure 6. Brute-force flowchart diagram](image-url)
2.4.4. Solve Sudoku as a Constraint Problem Algorithm

The Constraint programming paradigm is based on relating variables by constraints. Its main difference from imperative programming is that it does not execute in sequences of steps. It states the computation logic to find the solution without any flow control. “In constraint programming, a model is stated is terms of variables that range over their domain of possible values, and as constraints on these variables.” (Hoeve, 2005) There are several constraints used for different problems. According to Simonis (2005), the best way to model a Sudoku as a constraint problem is to use a combination of the alldifferent constraint. This constraint declares that all variables in it are pairwise different. He also define the Sudoku square and the Sudoku problem as "A Sudoku square of order n consists of n^4 variables formed into a n^2 × n^2 grid with values from 1 to such n^2 that the entries in each row, each column and each of the n^2 major n × n blocks are alldifferent. In addition, a Sudoku problem consists of a partial assignment of the variables in a Sudoku square. The objective is to find a completion of the assignment which satisfies the constraints." (Simonis, 2005) He classifies the Sudoku solving model constraints into four interactions; Row/Column interaction, Row/Block interaction, Rows/Blocks interaction and Rows/Columns/Blocks interaction. After defining these constraints, the Shaving technique is used to eliminate unneeded inconsistent values before the searching starts.

The advantage of using constraints programming for solving Sudoku is that a well-written program won’t exceed 100 lines of code, not to mention that it will find the solution in milliseconds. On the other hand, using this paradigm will express the solution directly without iterations to calculate its difficulty or steps to explain the solution for the user.
2.5. Generating Sudoku Puzzle

Many Sudoku players believe that hand-made Sudoku puzzles “crafted puzzles” are more enjoyable than computer-generated ones. They consider it a very annoying process to attempt to solve a Sudoku that is either too difficult or has more than one solution. For a generated puzzle to be valid, it must have only one solution. To do that, we have to solve the generated puzzle and try to find all other possible solutions. There are many ways to generate the puzzle; the easiest way is to fill a few cells in the grid randomly making sure that all rules and constraints are applied and then try to solve it. Constraint satisfaction problems like this are most efficiently solved using constraint solvers. Or for more efficiency, more than one algorithm could be used for generating the puzzle. For example a constraints program could apply the Sudoku constraints to an empty puzzle and then put random digits on the board and use backtracking to solve the puzzle and check if there is more than one solution. Another way is to start with a completed board, and try to reach the smallest possible set of clues that provide a unique solution. All the previously mentioned algorithms can be used to generate a Sudoku puzzle, and one of the aims of the design phase in this project is to test these algorithms to find the most efficient way to generate a well-formed entertaining puzzle.

2.6. Sudoku Difficulty Ranking

Ranking the difficulty of a Sudoku puzzle is a complex topic, because it is a subjective thing. To estimate the real difficulty of a puzzle we must know how many steps and techniques a human being uses to solve that puzzle, and to do that, the algorithm must mimic the methods used by a human being to solve Sudoku puzzles. This makes those “AI” type algorithms noticeably slower than straightforward algorithms. (Brouwer, 2006) There are also some other factors that can be used to determine the difficulty of a puzzle. For example the number of initial empty cells in the puzzle is a guide to difficulty, because the more empty cells, the more effort a user has to expend to solve the puzzle. Another important factor is the placement of the initial numbers. As Lee (2006) emphasizes, “Simple puzzles often have their initial numbers evenly spaced apart, whereas difficult puzzles often have numbers clustered in groups.”
2.7. Conclusion

Due the popularity of the Sudoku game, many programmers who are addicted this game have tried to implement their knowledge of certain algorithms to solve the Sudoku problem. This has resulted in the use of various algorithms to solve and generate Sudoku puzzles and only few of them are discussed in this dissertation. Each algorithm has its advantages and disadvantages in some way.

We could conclude that the best way to achieve efficiency when solving or generating Sudoku puzzle is to use more than one algorithm and combine a set of them that give us the desired speed and the well-formed puzzle we are looking for.

With regard to our project requirements, two types of algorithms could be used for solving; a straight-forward algorithm to find solutions for a puzzle and an AI type algorithm to provide hints and steps in the solution for the user. For generating the puzzle also more than one algorithm can be used. Constraints based algorithms are going to be used definitely because it is provide fast, efficient way to make sure all the game rules are applied. The other issue to be faced is to make sure of the uniqueness of the solution.

The decisions of which algorithms are to be used will be taken in the design phase in this project, where these algorithms are going to be examined to find the fastest and most efficient among them that suits our project.
CHAPTER 3

PROFESSIONAL, LEGAL AND ETHICAL ISSUES
3.0. Professional, Legal and Ethical Issues

3.1. Professional and Ethical Issues

This project will follow “Software Engineering Code of Ethics and Professional Practice” in all of its phases; the last version of this Code is 5.2. According to the Association for Computing Machinery, “It is approved by the IEEE-CE and ACM as the standard for teaching and practicing software engineering.”

The following code of practice is the short version and it is copyrighted © 1999 by the Association for Computing Machinery, Inc. and the Institute for Electrical and Electronics Engineers, Inc.

“Software engineers shall commit themselves to making the analysis, specification, design, development, testing and maintenance of software a beneficial and respected profession. In accordance with their commitment to the health, safety and welfare of the public, software engineers shall adhere to the following Eight Principles:

1. PUBLIC:
   Software engineers shall act consistently with the public interest.

2. CLIENT AND EMPLOYER:
   Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.

3. PRODUCT:
   Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.

4. JUDGMENT:
   Software engineers shall maintain integrity and independence in their professional judgment.

5. MANAGEMENT:
   Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.
6. PROFESSION:
Software engineers shall advance the integrity and reputation of the
profession consistent with the public interest.

7. COLLEAGUES:
Software engineers shall be fair to and supportive of their colleagues.

8. SELF:
Software engineers shall participate in lifelong learning regarding the
practice of their profession and shall promote an ethical approach to
the practice of the profession.”

3.2. Legal Issues
A questionnaire will be used to evaluate the first release of this project’s software. It
will interact with the user and comply with the following legislation: Data Protection
2000.
CHAPTER 4

PROJECT PLAN
4.0. Project Plan

4.1. Timeline

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Duration</th>
<th>Start</th>
<th>Finish</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Research Background</td>
<td>21 days</td>
<td>05/03/2012</td>
<td>02/04/2012</td>
</tr>
<tr>
<td>2</td>
<td>Requirements Analysis</td>
<td>10 days</td>
<td>03/04/2012</td>
<td>16/04/2012</td>
</tr>
<tr>
<td>3</td>
<td>Easter vacation &amp; Exams</td>
<td>14 days</td>
<td>17/04/2012</td>
<td>04/05/2012</td>
</tr>
<tr>
<td>4</td>
<td>Design</td>
<td>14 days</td>
<td>07/05/2012</td>
<td>24/05/2012</td>
</tr>
<tr>
<td>5</td>
<td>Coding and Testing</td>
<td>30 days</td>
<td>25/05/2012</td>
<td>24/06/2012</td>
</tr>
<tr>
<td>6</td>
<td>Coding</td>
<td>30 days</td>
<td>25/05/2012</td>
<td>24/06/2012</td>
</tr>
<tr>
<td>7</td>
<td>Testing</td>
<td>30 days</td>
<td>25/05/2012</td>
<td>24/06/2012</td>
</tr>
<tr>
<td>8</td>
<td>Evaluation</td>
<td>14 days</td>
<td>24/07/2012</td>
<td>07/08/2012</td>
</tr>
<tr>
<td>9</td>
<td>Prototype Final Version</td>
<td>7 days</td>
<td>08/08/2012</td>
<td>15/08/2012</td>
</tr>
<tr>
<td>10</td>
<td>Dissertation Submission</td>
<td>1 day</td>
<td>16/08/2012</td>
<td>16/08/2012</td>
</tr>
<tr>
<td>11</td>
<td>Poster and Presentation</td>
<td>2 days</td>
<td>23/08/2012</td>
<td>24/08/2012</td>
</tr>
</tbody>
</table>

There are five milestones in this project:

- Milestone 24/5: create the final design of the program.
- Milestone 24/6: create the first release of the program.
- Milestone 7/8: evaluate the program and improve it.
- Milestone 15/8: create the final release of the program.
- Milestone 16/8: submit the dissertation.

Three deliverables are required for this project:

- Research report and literature review.
- Dissertation and the final distribution of the program.
- The poster for posters day presentation.
4.2. Evaluation Plan

After the first release of the program being created, it will be distributed as a demo to various users with a questionnaire about the puzzle quality, solving aids tools, difficulty ranking if it is available at that time, game interface design, ease of use, the quality of the puzzles, the enjoyment level of the puzzles and any other related project issues.

A group of at least ten participants will be employed in the evaluation. This group should contain experienced and occasional Sudoku players. Depending on the questionnaire results, the program will be improved to meet users’ expectations and requirements in the final release.

The questionnaire will be online and participants will play two to four different puzzles as they choose, then directed to the questionnaire to fill it in. The questionnaire will collect the minimum amount of personal information; all other data will be related to the program and what the participants think about it. This personal information will include age, genre, Sudoku playing experience and how frequently the participant plays Sudoku puzzles.

4.3. Risk Assessment

This risk assessment attempts to identify all risks that are known at this time for the project and it will be updated continuously during the lifetime of this project.

<table>
<thead>
<tr>
<th>Risk</th>
<th>Likelihood of Event</th>
<th>Risk Mitigation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unrealistic time estimates</td>
<td>Likely</td>
<td>Timeline reviewed weekly to prevent undetected timeline departures.</td>
</tr>
<tr>
<td>Users with negative attitudes toward the program</td>
<td>Likely</td>
<td>A demo version distributed to users with a questionnaire will be used to improve the final version.</td>
</tr>
<tr>
<td>High level of technical complexity</td>
<td>Unlikely</td>
<td>Only proven Sudoku algorithms will be employed by this project.</td>
</tr>
<tr>
<td>Lack of required knowledge/skill in the project personnel</td>
<td>Likely</td>
<td>Assign sufficient time for design and coding</td>
</tr>
<tr>
<td>Risk</td>
<td>Likelihood of Event</td>
<td>Risk Mitigation</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>---------------------</td>
<td>---------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Continually changing system requirements</td>
<td>Unlikely</td>
<td>Requirements capture will be managed at certain stages to prevent this.</td>
</tr>
<tr>
<td>Real-time performance shortfalls</td>
<td>Unlikely</td>
<td>Only proven algorithms will be used which should reduce this risk.</td>
</tr>
<tr>
<td>Lack of cooperation from users</td>
<td>Certainty</td>
<td>Design the questionnaire in a simple though attractive way. Maybe providing rewards or motivations for participants.</td>
</tr>
<tr>
<td>Failure to manage end user expectations</td>
<td>Somewhat unlikely</td>
<td>Involvement of end users throughout development will make this matter unlikely.</td>
</tr>
</tbody>
</table>
CHAPTER 5

REQUIREMENTS ANALYSIS
5.0. Requirements Analysis

5.1. Introduction
Requirements analysis is an important phase in any software development project. It is used to determine every need and requirement to meet the project aims. These needs and requirements may change during the development of this project. This chapter attempts to determine the fundamental requirements for the system, which need to be satisfied when developing, designing and implementing the Sudoku Generator and Solver application. The main objectives of these requirements are to specify the business requirements, which focus on business user needs (or user requirements) as well as on system requirements, which focus on how the system should perform.

5.2. Aims
The aim of this project is to develop a computer program in Java for the end user. The program should be able to generate Sudoku puzzles for the user and be able to solve these puzzles and guarantee that each puzzle has a unique solution. It will also be able to give hints to the player by highlighting mistakes that a player enters in cells and improve the player playing experience through supportive tools.

5.3. Objectives
The program’s main objective is to generate Sudoku puzzles in an effective way that provides the user with a well-formed and entertaining challenge. Another objective is to make solving the puzzle more enjoyable through supportive tools. It also should be easy to use, easy to learn to use and provide the user with an attractive interface that is agreeable to interact with.

5.4. User and User Tasks Analysis
Users are considered to be the backbone for building up any successful system. In this project, we have only one type of potential user who will interact with our proposed application, which is the puzzle player. This user can perform different tasks in the system, as we will see next.
5.4.1 Getting to Know the Tasks

Task analysis can generally be reflected as being part of a family of techniques for representing information where design is concerned. (Dennis & Wixom, 2005).

Task analysis is used to determine how the users interact with our application. What is more, task analysis is used to obtain information about what they do and how they behave. The following diagram illustrates the main user tasks.

![Task Analysis Diagram]

Figure 10. The Task Analysis diagram
5.4.2 Main Tasks Description

The above diagram specifies the main tasks open to potential users in the application.

1. **Create Puzzle Task:**
   It allows the user to create a new random puzzle depending on a difficulty the user chooses.

2. **Open Puzzle Task:**
   Allows the user to open and load a puzzle from a file. This file could be obtained from anywhere; it could be saved previously or copied from another user.

3. **Type in custom puzzle Task:**
   This task allows the user to type in their own custom puzzle by showing a blank puzzle grid and let the user check the entered puzzle for validity. If the entered puzzle is valid, the user will be able to save it.

4. **Save current Puzzle Task:**
   Allows the user to save the current puzzle with its current entries into a file.

5. **Solve Puzzle Task:**
   It allows the user to show the solution of the current puzzle, noting that after this task, playing will stop and the user will have to start new puzzle.

6. **Play Puzzle Task:**
   It allows the user to fill in empty cells of the puzzle and displays a message if the user successfully solves the puzzle.

---

**Figure 11. The Main Task Analysis**
5.5. Functional Requirements

Functional requirements explain the operations that the application should perform. These requirements are listed below for the system analyst and designer of the Sudoku Generator and Solver application.

1. Create new puzzle function
   1.1. The user will select “new puzzle” from the menu or toolbar.
   1.2. The application will display different difficulty choices.
   1.3. The user selects a certain difficulty and presses “create”.
   1.4. The application generates a new puzzle with the selected difficulty.
   1.5. The application displays the generated puzzle for playing.

2. Solve puzzle function
   2.1. The user clicks “solve puzzle” from the menu of the toolbar for the current puzzle or the custom entered puzzle.
   2.2. The application tries to find more than one solution for the puzzle.
   2.3. If there is only one solution it will be displayed.
   2.4. If there is more than one solution, the application will display an error message that the puzzle has more than one solution.
   2.5. If the puzzle cannot be solved, the application will display an error message that the puzzle is unsolvable.

3. Open puzzle function
   3.1. The user selects “open puzzle” from the menu or the toolbar.
   3.2. The application shows an “Open file dialog” to let the user choose a file.
   3.3. The user selects a file and presses “Open”.
   3.4. The application checks the file format for validity.
   3.5. If the file is valid, it will load the puzzle from the file.
   3.6. If the file is invalid, the application will display an error message that the file cannot be opened.

4. Save puzzle function
   4.1. The user presses “save puzzle” from the menu or toolbar.
   4.2. The application shows a “Save file dialog” and let the user choose a file name and location.
4.3. The user writes the file name and presses “save”
4.4. The application converts the current puzzle into an XML format.
4.5. The application tries to write the XML to the selected file.
4.6. If the file could not be written, an error message will be displayed that the file cannot be saved.

5. Type in custom puzzle function
   5.1. The user selects the “enter custom puzzle” option from new puzzle dialog.
   5.2. The application displays a blank puzzle grid and allows the user to type in all entries. It also provides a check button.
   5.3. The user presses the “check” button after finishing entering all the puzzle clues.
   5.4. The application solves the puzzle and checks its validity.
   5.5. If the puzzle is valid, the application allows the user to solve or save it.
   5.6. If the puzzle is invalid, an error message appears that the entered puzzle is invalid and allows the user to continue entering the custom puzzle.

6. Play the puzzle function
   6.1. The user starts to fill in the puzzle’s empty cells.
   6.2. The application checks for each entry if the puzzle is solved.
   6.3. If the puzzle is solved, a message appears that the puzzle has been successfully solved.
   6.4. If the “show mistakes” option is selected, the application changes the font colour to red for each wrong entry.

7. Undo/Redo function
   7.1. The user presses the Undo or Redo buttons from the menu or toolbar.
   7.2. The application checks a list of the last moves for the required action.
   7.3. It undoes or redoes the last move from the stored list.
   7.4. If there are no more moves, a message is displayed that there are no more moves to undo or redo.

8. Timer function
   8.1. The user selects it to start a timer while solving the puzzle.
8.2. The application start a timer with the ability to pause it.
8.3. The application stops the timer once the puzzle is solved by the user or by the application.
8.4. The application resets the timer when a new puzzle is loaded.
8.5. The timer value is saved with the puzzle when it is written to a file.

9. Clear user entries function
9.1. The user presses “clear puzzle” from the menu or toolbar.
9.2. The application removes all user entries from the puzzle.

5.6. Non-functional Requirements
The following information should be considered in the implementation process as these properties have an indirect relation to the system as a whole and can affect the system and interaction between the system and its intended users.

1. Usability Requirements
1.1. The interface of the application should be attractive and enjoyable.
1.2. Navigation through the application should be easy and obvious.
1.3. Dialogue boxes and error messages should be obvious.
1.4. The application must be easy to learn and easy to use for the end user.
1.5. The application must provide an easy to read user manual.
1.6. The application should be evaluated by users to improve the application’s usability.

2. Performance Requirements
2.1. The application should respond to a user’s requests in a reasonable time.
2.2. The user interface have an easy to use and simple design.

3. Operational Requirements
3.1. The application should be implemented in the Java programming language.
3.2. The application should run on any operating system that support the JRE without any additional library being needed.
3.3. The application code must be well documented.
3.4. The application code must follow Java code conventions.
5.7. Functional Modelling

In the previous section, we defined the necessary functional requirements for the Sudoku Generator and Solver application. This section attempts to translate these defined functional requirements into functional modelling which names use cases.

5.7.1 Use case analysis

Use case diagrams are simply descriptions of what the user can perform and how the application should act in response to a user’s action.

5.7.1.1 Identification of Actors

An actor is a person who interacts with the system to accomplish a specific task. However, the actor is not an individual, but is a role that a person can perform while interacting with the system. (Dennis & Wixom, 2005). In our application, we have only one actor, which is the Player. This actor will perform all the tasks and all the use cases depending on it.

5.7.1.2 Identification and Description of use cases

A use case is the task that an actor wants the application to perform. The following list identifies the use cases of the application.

1. Create new puzzle:
   It allows the user to create a new puzzle depending on a difficulty the user chooses or to enter a custom typed in puzzle.

2. Open puzzle:
   It allows the user to open and load a puzzle from a file.

3. Save puzzle:
   It allows the user to save the current puzzle with its given entries into a file.

4. Solve puzzle:
   It allows the user to show the solution of the current puzzle.

5. Undo/Redo:
   It allows the user to undo or redo his moves.

6. Clear puzzle:
   It allows the user to erase all his entries.
7. **Play puzzle:**

It allows the user to fill the empty cells of the puzzle and display a message if the user successfully solved the puzzle.

![Use case diagram for the Sudoku Generator and Solver Application](image-url)
CHAPTER 6

OBJECT-ORIENTED DESIGN
6.0. Object-Oriented Design

After identifying the functional and non-functional requirements and drawing the use case, it is time to focus on designing and building a class diagrams and defining the interactive areas between classes.

6.2 Structural Modelling

Structural modelling (Class diagram) is a formal way to represent functional modelling (Use case) that was created in the previous chapter. It exemplifies different objects such as places, people or things about which information is captured and how they are related to each other. (Dennis & Wixom, 2005). However, in this section, the structural model (Class diagram) will be represented and the relationship between classes noted.

6.2.1 Identify Classes, Attributes and Operations

From the waterfall perspective, classes can be identified depending on the use cases analysis. Consequently, the Sudoku Generator and Solver application has five classes for the game and seven classes for the user interface with each class having its own attributes and operations. The following figures specifies the application’s classes.

Figure 14. Sudoku package Classes
6.2.2 Association Between the classes

Association in UML represents several types of relationship between classes and has subtleties that are not specified in the UML specification. (Pilone, 2005).

This following figure illustrates how the classes in the Sudoku Generator and Solver link to each other.

Figure 15. GUI package Classes

Figure 16. Association between Classes
### 6.2.3 Class Diagrams

**Figure 17. Sudoku package class diagram**
Figure 18. GUI package class diagram
CHAPTER 7

USER INTERFACE DESIGN
7.0. User Interface Design

7.1. Introduction
This section introduces the user interface design aspects of the Sudoku Generator and Solver application. In this section, we will provide information relating to the guidelines followed and the techniques used in the development of the proposed user interface.

7.2. Guidelines for Interface design
There are number of factors that can help us design a successful and user-friendly interface. However, the author decided to follow some standard guidelines to design the user interface of the Sudoku Generator and Solver application. These guidelines are helpful and useful in designing an attractive user interface.

- **Golden Rules of Interface Design**
The following table illustrates the Eight Golden Rules of Interface Design that were recommended by Shneiderman (Shneiderman, 2004).

<table>
<thead>
<tr>
<th>NO</th>
<th>The Golden Rules of Interface Design</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td><em>Strive for consistency.</em> Sequences of action should behave in a consistent way for all screens. Terminology, icons, texts and colours should be consistent between all screens.</td>
</tr>
<tr>
<td>2</td>
<td><em>Cater to universal.</em> The different needs of diverse users must be taken into account. For example, providing short-cuts for experts and explanations for novices.</td>
</tr>
<tr>
<td>3</td>
<td><em>Offer information feedback.</em> System feedback must be offered for all actions. That for frequent and minor actions can be simple and more detailed and substantial for major actions.</td>
</tr>
<tr>
<td>4</td>
<td><em>Design dialogs to yield closure.</em> The design dialogs inform of closures. Actions that relate to one group can be organized to have three phases (beginning, middle and end). Informative feedback is important to tell the user of the completion of actions.</td>
</tr>
</tbody>
</table>
Prevent error. The system must be designed in such a way that it prevents the user from making any errors. This can be accomplished by displaying buttons (making the buttons grey) so the user can only see them but cannot click them. Another example is to prevent the user from entering the wrong characters in the wrong field (e.g. entering numbers in an alphabetic field).

Permit easy reversal of actions. The design should enable the user to easily go back and undo errors. Doing so will relieve any doubts if the user has entered wrong information or data.

Support internal locus of control. Importing a feeling of control is strongly recommended for experienced users.

Reduce short-term memory load. The limitation of human processing in short term memory must be taken into account. The display should be kept simple and window motions must be reduced.

7.3. Interface Layout

This section describes the screen layout that can be adopted for the Sudoku Generator and Solver application. The following figure illustrates the screen layout for the application.

![Figure 19. The layout of the application user interface](image-url)
7.4. Graphical User Interface

This section describes the process of designing the user interface from a paper-based design approach to a computer-based design approach. Different tools were chosen for designing the Sudoku Generator and Solver application user interface. These tools have been carefully selected to enhance the efficiency of the user interface. However, the tools used in designing this application were GIMP pictures editor and NetBeans as the main IDE for the program. The following sections describe how these tools have been used in designing the application.

7.4.1. Main Screen Design

In this section, we will describe the three main screen designs, the gameplay screen, the create new puzzle screen and the custom typed-in puzzle screen. There are other several user interfaces screens but they will not be described as they are common in many applications, like open file screen and save file screen... etc.

Figure 20. Gameplay screen discretion
In the type-in custom puzzle mode, all of the puzzle board becomes blank so the users can enter their custom puzzle. There are two buttons on the bottom pane, “Check Puzzle” that checks the puzzle entered by the user for validity. And “Load Puzzle” button that loads the puzzle for playing. Noting that “Load Puzzle” button is enabled only if the puzzle is valid. Therefore, the user needs to check the puzzle first before loading it for solving and playing.

The “Create Puzzle” screen provides the users with difficulty choices for the new puzzle. It has three difficulty levels: easy, medium and hard. Due time limitation on this project, only the clues number measures the difficulty levels. To accurately measure the difficulty of a puzzle, human-like algorithms needed to find how long it takes a human to solve the puzzle. Such algorithms need longer time to implement, which was not available while developing this project.
CHAPTER 8

IMPLEMENTATION
8.0. Implementation

8.1. Introduction
Many critical aspects of analysis and design activities have been taken into consideration before starting the implementation phase. These aspects assisted the implementing of the application. This chapter is concerned with developing the application and different implementation issues.

8.2. Technologies Used for Implementing the Application
Java was used for creating this application. It is an object-oriented programming language created by Sun Microsystems. The main reason to use this programming language is that it is platform independent. That means, the developer can write the code only once and the compiled application will be able to run in all supported platforms without any code alterations. Moreover, all Java applications can be run online directly through most of the common browsers available nowadays.

The IDE used for writing the Java code was NetBeans. It is a platform framework and an integrated developing environment for Java. It offers good help tools and an easy to use interface design capability.

GIMP was also used to create and design application icons. It is a free open-source image editing tool.

8.3. Building Classes
The first step in the implementation is to create the classes which been designed as specified in the sixth chapter to make the application ready for working as a whole.

The core of all classes, which is essential for all the classes, used by all of them, is the class responsible to represent the Sudoku puzzle itself, and it is called “Puzzle”. This class has a two dimensional array “Cell” to represents the cells of the puzzle, a Boolean variable to flag the puzzle if it is solved or not and a Difficulty enumeration that records the difficulty of the puzzle. There are a lot of methods in this class most of them are

Figure 23. Puzzle Class Diagram
getters and setters, but the essential methods are the following

- **clear( ):** this method clears all the values stored on the cells of the puzzle except the given values which are the clues of the puzzle.

```java
public void clear(){
    for(int row=0;row < 9;row++)
        for(int col=0;col < 9;col++)
            if(!cells[row][col].isGiven())
                cells[row][col].setValue(0);
}
```

- **showSolution( )** this method does not solve the puzzle. It only shows the solution if the puzzle is already solved. It does that by changing the values of cells to the solution values.

```java
public Cell[] getRowCells(Cell cell){
    int cellRow = cell.getRow();
    Cell[] row = new Cell[9];
    for(int i=0;i<9;i++)
        row[i] = this.getCell(cellRow, i);
    return row;
}
```

```java
public Cell[] getColCells(Cell cell){
    int cellCol = cell.getCol();
    Cell[] col = new Cell[9];
    for(int i=0;i<9;i++)
        col[i] = this.getCell(i, cellCol);
    return col;
}
```

```java
public Cell[] getBlockCells(Cell cell){
    int row = (cell.getRow() / 3) * 3;
    int col = (cell.getCol() / 3) * 3;
    Cell[] block = new Cell[9];
    int counter = 0;
    for( int r = 0; r < 3; r++ )
        for( int c = 0; c < 3; c++ )
            block[counter++] = cells[row + r][col + c];
    return block;
}
```

- **getRowCells( ), getColCells( ) and getBlockCells( )** are used to return an array of cells that contain the cells on the same row, column or block of a certain cell. These methods are important to check the entries of a certain cell and the solution of it.

- **checkConflict( )** this method checks an array of cells to make sure all the values on them are not in conflict.

- **equals( )** this method compares the current puzzle with another. It is important to use to make sure if a puzzle has several solutions.
Another essential class is the “Cell” class, which represents an individual puzzle cell that contains a single value. It has five integer values, that represent the value that the user enters into the cell, the solution of the cell, the row, column and block to which the cell belongs. And a Boolean value to check if the cell is given or not. All the functions on this class are getters and setters. This class is used mostly as a type and usually used in an array to represent the puzzle board.

8.4 Implementing the Functional Requirements

After the essential class is implemented, the functional requirements for the application have to be implemented. The first and foremost is solving the Sudoku puzzle.

8.4.1 Solving the puzzle

In the second chapter, several Sudoku solving algorithms were discussed and we found that there are many algorithms used to solve Sudoku. One of the aims of this project was to find a good algorithm to do this job. However, because of time constraints not all of the algorithms were tested. After trying several approaches to find an efficient algorithm during this short time, a combination of a backtracking algorithm and a constraint logic algorithm were used to achieve this goal.

A separate class was created to solve “Puzzle” objects described earlier in this chapter. The class name is “BackTrackSolver” and its main job is to copy the targeted Puzzle object and solve it. It solves the puzzle in two different ways. Each way finds the first solution at the opposite side of the search tree. So if both of the solutions are the same then we can be satisfied that the puzzle has a unique solution. This class has four attributes, a counter used to count the number of iterations to solve the puzzle, the puzzle object which is a copy of the original puzzle so it will not affect it and two indices to track
moving among the puzzle cells. The main method used to solve the puzzle is called “solveCell()”. This method solve one cell at a time then calls itself to solve the next cell. If it reaches the last cell and solves it, then the puzzle is solved. As explained earlier, this class solves the puzzle in two ways. This method solves each cell by trying the numbers from one to nine and backtracking if necessary. One version of the method finds the solution from the other side in the search tree. Another version tries to solve the puzzle in the same way, except using numbers backwards from nine to one to solve the cell. Therefore, if both of the solutions are equal, we can be satisfied that the puzzle has a unique solution. The second method is called reverseSolveCell(). Only the first method will be described as the second method is the same except it’s reversed.

```java
public boolean solveCell(Cell currentCell){
    if(currentCell == null){
        resetPosition();
        currentCell = nextCell();
    }
    int value = 0;
    boolean solved = false;
    while(!solved && value < 9){
        counter++;
        value++;
        currentCell.setValue(value);currentCell.setSol(value);
        if(!(Puzzle.checkConflict(this.puzzle.getRowCells(currentCell)) &&
            Puzzle.checkConflict(this.puzzle.getColCells(currentCell)) &&
            Puzzle.checkConflict(this.puzzle.getBlockCells(currentCell)))))
            continue;
        Cell next = nextCell();
        if(next != null)
            solved = solveCell(next);
        else{
            solved = true;
            this.puzzle.setSolved(true);
        }
    }
    if(!solved){
        currentCell.setValue(0);currentCell.setSol(0);
        previousCell();
    }
    return solved;
}
```

Figure 26. Search Tree
8.4.2 Generate new puzzle

“PuzzleGenerator” is the class responsible for creating new random Sudoku puzzles. It does this task by creating a valid filled puzzle first. Then it chooses a number selected by user to set cells as randomly. These givens are set in a symmetric way to enhance the look of the puzzle and the enjoyment of playing. After setting these givens, it tries to solve the puzzle and checks if there is more than one solution. This process will loop until it obtains a valid puzzle with the selected number of givens. When the valid puzzle is obtained, it is saved to a puzzle object so it can be dealt with by other classes.

```java
public Puzzle getPuzzle(int givens){
    for(int i=0;i < 666;i++){
        puzzle = new Puzzle();
        fillPuzzle();
        clearGivens();
        setGivens(givens);
        clearValues();
        BackTrackSolver solver = new BackTrackSolver(puzzle);
        solver.solveCell(null);
        Puzzle solvedPuzzle = solver.getSolvedPuzzle();
        if(solvedPuzzle == null)
            continue;
        else{
            BackTrackSolver solver2 = new BackTrackSolver(puzzle);
            solver2.reverseSolveCell(null);
            Puzzle solvedPuzzle2 = solver2.getSolvedPuzzle();
            if(solvedPuzzle2.equals(solvedPuzzle)){
                puzzle = solvedPuzzle;
                clearValues();
                puzzle.setDifficulty(gradeDifficulty(givens));
                return puzzle;
            }
        }
    }return new Puzzle();
}
```

```java
private void setGivens(int givensCount){
givenCells.clear();
if(givensCount%2 != 0){
givenCells.add(puzzle.getCell(4,4));
}
for(int i=0;i<givensCount/2;i++){
    Cell cell = getRandomCell();
    if(givenCells.contains(cell)){
        continue;
    }
givenCells.add(cell);
givenCells.add(puzzle.getCell(8-cell.getRow(), 8-cell.getCol()));
}
for(Cell c:givenCells)
c.setGiven(true);
```
8.4.3 Puzzle board and gameplay

The gameplay was implemented on the user interface. Four main classes were created to do this, the first class represents the game board, was named “PuzzleJPanel”, is linked to a puzzle object to represent the puzzle and contains the cells of the puzzle in a class that will be described later. This class interacts directly with the user solving process and aid the user in this process. It manages user movements and changes the colours of the cell's background to aid the user easily recognizing the sector of each cell “the row, column and block of the cell”. It also has a flag named “showMistakes” to determine if it should change the foreground colour of the cell if the user entered a wrong value in it. This class has many methods too. The most important among them are seven methods. update( ) method is used to update the current board with the values exists on the puzzle object. undo( ) and redo( ) are used to move among user movements by defanging an object of type History and storing each move in this object.

```java
public void undo(){
    if(undoHistory.size() > 0){
        History undo = undoHistory.pop();
        redoHistory.push(undo);
        cells[undo.row][undo.col].setValue(undo.oldValue);
    } else
        JOptionPane.showMessageDialog(this, "No more moves to undo");
}
```

```java
public class History {
    public int row, col, oldValue, newValue;

    public History(int row, int col, int oldValue, int newValue){
        this.row = row;
        this.col = col;
        this.oldValue = oldValue;
        this.newValue = newValue;
    }
}
```
setPuzzle( ) method is used to change the current linked puzzle to a different one. The last two methods are used to track user movements to change the cell’s background colour. These methods are called whenever the cursor moves above a cell or exits it.

```java
@Override
public void mouseEntered(MouseEvent e) {
    if(e.getSource() instanceof CellJTextField){
        CellJTextField tfCell = ((CellJTextField)e.getSource());
        int row = tfCell.getRow();
        int col = tfCell.getCol();
        int block = row/3 * 3 + col/3;

        Component[] blockCells = panels[block].getComponents();
        for (Component temp : blockCells){
            tfCell = ((CellJTextField)temp);
            if(tfCell.isGiven())
                tfCell.setBackground(blockHoverGivenColor);
            else
                tfCell.setBackground(blockHoverDefaultColor);
        }

        for(int i=0;i < 9;i++) {
            if(cells[row][i].isGiven())
                cells[row][i].setBackground(hoverGivenColor);
            else
                cells[row][i].setBackground(hoverDefaultColor);
            if(cells[i][col].isGiven())
                cells[i][col].setBackground(hoverGivenColor);
            else
                cells[i][col].setBackground(hoverDefaultColor);
        }
    }
}
```

The eighth method is responsible for entering values into the cells to solve the puzzle. This method captures the user input. It checks if the key is one of the number keys, or it does nothing. If it is a number key, then it sees if there already a value in the cell to replace it or if it is a new value. And if the key is a backspace or delete key it deletes the value in the cell. It also checks with each entry if the puzzle is solved, and if it is solved then it displays a message that the puzzle is solved and stops the timer.
The second class created while implementing the gameplay is the “CellJTextField” class. This class represents a single cell on the puzzle board and is linked to a Cell object that exists on the puzzle object linked on the PuzzleJPanel. The main purpose of this class is to display the cells of the puzzle in an organized way. The third class is the “TimerJPanel” and its purpose is to display a timer while playing the puzzle. The timer can be paused or hidden if the user wishes it. It is created as a thread and can be controlled by methods defined on it. These methods are reset( ) which resets the timer counting, stop( ) which stops the timer counting and run( ) which starts the timer.

The fourth class is “MainJFrame”. This class contains all of the above objects and manages them. It also provides the user with tools and menus that aid the user while solving the puzzle.
8.4.4 Saving and Loading puzzle

“PuzzleFileManager” is the class responsible for writing and reading Sudoku puzzles. It converts the Puzzle object into an XML file with all the values and puzzle statuses saved as attributes in this file and saves it with the extension .puzzle.

The following is an example of a puzzle saved as XML

```
<puzzle solved="true" difficulty="EASY">
  <cell row="0" col="0" sol="6" value="6" given="true"/>
  <cell row="0" col="1" sol="1" value="0" given="true"/>
  <cell row="0" col="2" sol="3" value="3" given="true"/>
  <cell row="0" col="3" sol="2" value="0" given="true"/>
  ...
  <cell row="8" col="8" sol="2" value="2" given="true"/>
</puzzle>
```

```java
public static boolean writePuzzle(Puzzle puzzle, File file){
    StringBuilder str = new StringBuilder();
    FileWriter fw;
    if(!file.getName().endsWith(".puzzle")){
        String newFile = file.getAbsolutePath().trim() + ".puzzle";
        file = new File(newFile);
    }
    str.append("<puzzle solved="true" difficulty="EASY">
    str.append(puzzle.isSolved()).append(""");
    str.append(" difficulty="").append(puzzle.getDifficulty()).append("">
    for(int row=0;row<9;row++){
        for(int col=0;col<9;col++){
            Cell cell = puzzle.getCell(row, col);
            str.append("<cell ");
            str.append("row=").append(cell.getRow()).append(" ");
            str.append("col=").append(cell.getCol()).append(" ");
            str.append("sol=").append(cell.getSol()).append(" ");
            str.append("value=").append(cell.getValue()).append(" ");
            str.append("given=").append(cell.isGiven()).append("/>");
        }
        str.append("</puzzle>");
    }
    try{
        if(!file.exists())
            file.createNewFile();
        fw = new FileWriter(file);
        fw.write(str.toString());
        fw.close();
    }catch(IOException e){
        return false;
    }
    return true;
}
```

[All Code Listing are Located in Appendix C]
CHAPTER 9

TESTING AND EVALUATION
9.0. Testing and Evaluation

This chapter focuses on the testing and evaluation of the Sudoku Generator and Solver application. The aim of this part of the project is to ensure that the application functions are properly developed and meet all user’s requirements which were specified in the analysis phase.

9.1. Testing Levels

Once the application is built it needs to be tested to ensure that it is usable and can perform all tasks that were specified in the analysis phase for the users. Testing has different scopes such as Unit Test, Integration Testing, System Testing and Acceptance Testing. (Copeland, 2004).

9.1.1 Unit Test

A Unit test has two strategies for testing – a black box test and a white box test. The black box test is passed on the requirements and specifications and does not require any knowledge about the structure or implementation of the system. On the other hand, the white box test is based on the structure and implementation of the system and requires good knowledge of, and skills in, programming. (Copeland, 2004). In this project, the black box strategy was selected to test the functionalities of the system.

9.1.1.1 Applying Black Box Testing

<table>
<thead>
<tr>
<th>Test Number</th>
<th>Function</th>
<th>Result</th>
<th>Description</th>
<th>Expected results notes</th>
<th>Actual results notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Type number inside a cell</td>
<td>☑ Passed</td>
<td>Select a cell in the puzzle board</td>
<td>Test case is valid.</td>
<td>Test case accepted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Type any number from 1 to 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Number</td>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------------</td>
<td>----</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Function</td>
<td>Type a character inside the cell</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Result</td>
<td>☑ Passed  ☐ Failed</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
| Description | Select a cell in the puzzle board  
Type character that is not a number |
| Expected results notes | Test case is invalid. And the application should not display the character. |
| Actual results notes | Test case was rejected with the correct action. |

<table>
<thead>
<tr>
<th>Test Number</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Create new puzzle</td>
</tr>
<tr>
<td>Result</td>
<td>☑ Passed  ☐ Failed</td>
</tr>
</tbody>
</table>
| Description | Select File from the menu  
Select New Puzzle  
Choose 30 for clues number  
Select Create |
| Expected results notes | Test case is valid |
| Actual results notes | Test case was accepted |

<table>
<thead>
<tr>
<th>Test Number</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
<td>Solve a puzzle</td>
</tr>
<tr>
<td>Result</td>
<td>☑ Passed  ☐ Failed</td>
</tr>
</tbody>
</table>
| Description | After creating new Puzzle in the previous test  
Select Puzzle from menu  
Select Solve |
| Expected results notes | Test case is valid |
| Actual results notes | Test case was accepted |

[The Rest of Test Cases are Located in Appendix A]
9.2. Evaluation Plan

The evaluation plan was modelled after the different stages of the usability evaluation process of the application. The usability evaluation process was broken down into two stages:

- Usability attributes
- Empirical Evaluation.

9.2.1 Usability Attributes

Usability attributes are the characteristics of software that need to be measured. Faulkner, in his book “Usability Engineering”, recommends a number of usability attributes such as effectiveness, efficiency, user satisfaction, learnability and flexibility to be used in evaluating the characteristics of any software. (Faulkner, 200). In this section we will apply these usability attributes to the Sudoku Generator and Solver application to determine its strong and weak points.

**Efficiency**

The application is efficient where the user can normally reach their goal in maximum of three clicks. On the other hand, only one task needed more than three clicks.

**Effectiveness**

The application has achieved all functions and tasks with a high quality of information coordination. On the other hand, some functions such as sound are not included in the application.

**User Satisfaction**

In this stage we cannot judge whether the users are satisfied or not. However, it would be clear in an empirical evaluation.

**Learnability**

To interact with the application, users do not need to learn special skills. However the application is designed for all types of users, the speed up factor depends on how well users are familiar with Sudoku puzzles and the application.

9.2.2 Empirical Evaluation

This section introduces the second method for evaluating the Sudoku Generator and Solver application. In this method, the user will evaluate the application. The reason
behind using this method is to discover some problems that may not have been encountered or envisaged by the developer. This method has different techniques such as Experiment, Observation and Questioner.(Dix & others, 1997). In this section, the Questioner part will be applied to inspect the empirical method.

### 9.2.2.1 Questionnaires

A questionnaires is a popular way to evaluate a system by asking the user about it. This technique is used because it is helpful in obtaining suggestions and improvements for the application.

### 9.2.2.2 Questionnaire Analysis

This questionnaire can be accessed from:

[www2.macs.hw.ac.uk/~aj153/sudoku](http://www2.macs.hw.ac.uk/~aj153/sudoku)

<table>
<thead>
<tr>
<th>Questions</th>
<th>Result</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. How often do you play Sudoku?</td>
<td>Daily</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Few times a week</td>
<td>13%</td>
</tr>
<tr>
<td></td>
<td>Once a week</td>
<td>38%</td>
</tr>
<tr>
<td></td>
<td>Less often</td>
<td>50%</td>
</tr>
<tr>
<td>2. In which category do you put yourself as a Sudoku player?</td>
<td>beginner</td>
<td>63%</td>
</tr>
<tr>
<td></td>
<td>intermediate</td>
<td>25%</td>
</tr>
<tr>
<td></td>
<td>advanced</td>
<td>13%</td>
</tr>
<tr>
<td>3. Does information appear clearly in the application’s screens?</td>
<td>Excellent</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td>Good</td>
<td>50%</td>
</tr>
<tr>
<td></td>
<td>Fair</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Poor</td>
<td>0%</td>
</tr>
<tr>
<td>4. Are the contents of the application well organized?</td>
<td>Excellent</td>
<td>88%</td>
</tr>
<tr>
<td></td>
<td>Good</td>
<td>13%</td>
</tr>
<tr>
<td></td>
<td>Fair</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Poor</td>
<td>0%</td>
</tr>
<tr>
<td>5. How would you evaluate the design of the application’s screens?</td>
<td>Excellent</td>
<td>63%</td>
</tr>
<tr>
<td></td>
<td>Good</td>
<td>38%</td>
</tr>
<tr>
<td></td>
<td>Fair</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Poor</td>
<td>0%</td>
</tr>
<tr>
<td>6. How would you evaluate the use of layout, colour and graphics in the application screen?</td>
<td>Excellent</td>
<td>75%</td>
</tr>
<tr>
<td></td>
<td>Good</td>
<td>25%</td>
</tr>
<tr>
<td></td>
<td>Fair</td>
<td>0%</td>
</tr>
<tr>
<td></td>
<td>Poor</td>
<td>0%</td>
</tr>
<tr>
<td>7. How easy was it to accomplish your goals using the application?</td>
<td>Excellent</td>
<td>38%</td>
</tr>
<tr>
<td></td>
<td>Good</td>
<td>63%</td>
</tr>
<tr>
<td></td>
<td>Excellent</td>
<td>Good</td>
</tr>
<tr>
<td>---------------------</td>
<td>-----------</td>
<td>------</td>
</tr>
<tr>
<td>8. How entertaining were the puzzles?</td>
<td>25%</td>
<td>63%</td>
</tr>
<tr>
<td>9. How good was the difficulty grading of the puzzles?</td>
<td>25%</td>
<td>38%</td>
</tr>
<tr>
<td>10. How good was the puzzle design?</td>
<td>50%</td>
<td>50%</td>
</tr>
</tbody>
</table>

[The Individual Results are Located in Appendix B]

The above table shows the results of the questionnaire. As can be seen from the above table, there were 10 questions and each question had four answers (Excellent, Good, Fair, Poor) except the first two questions. There were also five comments boxes that ask the participant for comment about interface, game playing, puzzles difficulty, puzzles design, and any other general comments. According to the answers to these questions, most of the participants evaluated the application by giving it an “excellent” grade. That means the over all of evaluation the application is acceptable. However, a number of responses suggested a “fare” grade for the “how entertaining were the puzzles?” question and the “How good was the difficulty grading of the puzzles?” question. These questions will be considered in a future work. However, there were a small number of comments and suggestions, which will be studied carefully for future work. The following are the evaluation screens and a summary of the questionnaire.
EVALUATION

Please before filling the evaluation, Play the game first. You can run the game directly from the browser. You can also download an executable copy to run it locally from your computer. Please note that when running the game from the browser, you will not be able to save the puzzle or open previously saved puzzles.

How often do you play Sudoku?

☐ daily
☐ few times a week
☐ once a week
☐ less often

In which category do you put yourself as a Sudoku player?

☐ beginner
☐ intermediate
☐ advanced

Read the following statements and check the box that best reflects your opinion of the statement.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Excellent</th>
<th>Good</th>
<th>Fair</th>
<th>Poor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does information appear clearly in the application's screens?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Are the contents of the application well organized?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How would you evaluate the design of the application's screens?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How would you evaluate the use of layout, colour and graphics in the application?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How easy was it to accomplish your goals using the application?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How entertaining were the puzzles?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How good was the difficulty grading of the puzzles?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>How good was the puzzle design?</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Sudoku Generator and Solver

EVALUATION

Please help to improve the content of the application: Add your comments about each section.

User interface

Game playing

Puzzles difficulty

Puzzles design

-----

Are there areas of this application that could be improved?

-----

Figure 30. Questionnaire second page
In which category do you put yourself as a Sudoku player?

- **Beginner**: 63% (5)
- **Intermediate**: 25% (2)
- **Advanced**: 13% (1)

*Total Responses: 8, 100% of submissions

How often do you play Sudoku?

- **Daily**: 0% (0)
- **Few times a week**: 13% (1)
- **Once a week**: 38% (3)
- **Less often**: 50% (4)

*Total Responses: 8, 100% of submissions

---

Read the following statements and check the box that best reflects your opinion of the statement.

<table>
<thead>
<tr>
<th>Statement</th>
<th>Excellent</th>
<th>Good</th>
<th>Fair</th>
<th>Poor</th>
<th>Total Responses</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does information appear clearly in the application’s screens?</td>
<td>50% (4)</td>
<td>50% (4)</td>
<td></td>
<td></td>
<td>100% (8)</td>
</tr>
<tr>
<td>Are the contents of the application well organized?</td>
<td>88% (7)</td>
<td></td>
<td></td>
<td></td>
<td>100% (8)</td>
</tr>
<tr>
<td>How would you evaluate the design of the application’s screens?</td>
<td>63% (5)</td>
<td>38% (3)</td>
<td></td>
<td></td>
<td>100% (8)</td>
</tr>
<tr>
<td>How would you evaluate the use of layout, colour and graphics in the application?</td>
<td>75% (6)</td>
<td>25% (2)</td>
<td></td>
<td></td>
<td>100% (8)</td>
</tr>
<tr>
<td>How easy was it to accomplish your goals using the application?</td>
<td>38% (3)</td>
<td>63% (5)</td>
<td></td>
<td></td>
<td>100% (8)</td>
</tr>
<tr>
<td>How entertaining were the puzzles?</td>
<td>25% (2)</td>
<td>63% (5)</td>
<td></td>
<td></td>
<td>100% (8)</td>
</tr>
<tr>
<td>How good was the difficulty grading of the puzzles?</td>
<td>25% (2)</td>
<td>38% (3)</td>
<td>38% (3)</td>
<td></td>
<td>100% (8)</td>
</tr>
<tr>
<td>How good was the puzzle design?</td>
<td>50% (4)</td>
<td>50% (4)</td>
<td></td>
<td></td>
<td>100% (8)</td>
</tr>
</tbody>
</table>

*Total Responses: 8
CHAPTER 10

CONCLUSIONS AND FUTURE WORK
10.0. Conclusions and Future Work

10.1. Critical Assessment

In this section, we critically assess overall the Sudoku Generator and Solver. The procedure of having the application criticised is to highlight strengths and weaknesses that were faced during the development of this project. The following table shows the main assessment of the Sudoku Generator and Solver application.

<table>
<thead>
<tr>
<th>Critical Assessment</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Planning</strong></td>
</tr>
<tr>
<td><strong>Strengths</strong></td>
</tr>
<tr>
<td><strong>Weaknesses</strong></td>
</tr>
<tr>
<td><strong>Analysis</strong></td>
</tr>
<tr>
<td><strong>Strengths</strong></td>
</tr>
<tr>
<td><strong>Weaknesses</strong></td>
</tr>
<tr>
<td><strong>Design</strong></td>
</tr>
<tr>
<td><strong>Strengths</strong></td>
</tr>
<tr>
<td><strong>Weaknesses</strong></td>
</tr>
<tr>
<td><strong>Implementation</strong></td>
</tr>
<tr>
<td><strong>Strengths</strong></td>
</tr>
<tr>
<td><strong>Weaknesses</strong></td>
</tr>
</tbody>
</table>
10.2 Lessons Learnt from this Dissertation

After completing this dissertation, it is useful to specify what was learnt in various areas in the development of this dissertation. The following list highlights the lessons that the author has learned.

Lesson 1: planning is a crucial phase. Successful project planning can contribute towards the success of a project.

Lesson 2: Studying the different levels of user tasks aid system development and provides the developer with a good background to designing a new system in a form that meets user needs.

Lesson 3: I learnt how to use various UML techniques in developing this project. Structural modelling was applied that assisted me in firming up my knowledge with regard to using UML.

Lesson 4: I learnt new tools such as GIMP and UModel that helped me in designing the user interface and creating the UML diagrams.

Lesson 5: Before starting this dissertation, my knowledge of Java programing language was imprecise. However, once I started developing this project, my knowledge has been increased and my implementation skills have improved.

Lesson 6: I learnt different techniques in evaluation and testing this project such as Analytical Evaluation and Empirical Evaluation for the evaluation and Black box for the testing.

### Testing and Evaluation

<table>
<thead>
<tr>
<th>Strengths</th>
<th>Black box testing was used to test functionalities of the project. Moreover, Empirical Evaluation were used too to gain accurate evaluation result.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weaknesses</td>
<td>It took time to learn and apply these methods.</td>
</tr>
</tbody>
</table>
10.3 Future Work

In this section, some points are highlighted which may need further research and analysis to improve the proposed application. Additional features that may need to be added for further improvements of usability are as follows:

- Solve the puzzle using human-like solving algorithms so the difficulty of the puzzle can be graded more acutely.
- Other variations of Sudoku like 6×6 or 12×12 puzzles can be added easily in the application because the Puzzle class can already manage them. Just a little code alteration is needed on the interface.
- If human-like solving algorithms are implemented, hints and a step-by-step guide can be added to aid the player.
- The ability to choose the colours of the cells when the mouse hovers over them would be more convenient for the user.
- Showing the solution for the selected cell only was planned to be done but it was not because of lack of time.

10.4 Conclusions

The objectives of this study were to understand software engineering better by carrying out the development of this system. The understanding of the different issues faced in Sudoku Computer Games development were explained throughout this study and used to ensure that the final product would satisfy the user needs.

The management aspects of this study have been a useful aid in building my knowledge and understanding of how to work through similar projects in the future.

In addition, to the main problems encountered during this study, time limitations and the short time scale were the most limiting issue. The adoption of new developing technologies was also difficult. In addition, many features were envisaged to be done in the planning phase, but due to lack of time could not be implemented.

However, time limitations were overcome and a fully operational Sudoku Generator and Solver program was developed. The different sources available to the developer
did compensate for some of the time limitations as it can be said that due to the vast resources on software engineering provided by the university library, the developer was able to come up with the required technologies.

This study has allowed me to develop an understanding of how this type of project is handled in the fast-growing IT industry, as well as how to build on the different aspects of puzzle solving algorithms.
References

   http://www.andrew.cmu.edu/user/vanhoeve/papers/alldiff.pdf

   http://www.afjarvis.staff.shef.ac.uk/sudoku/felgenhauer_jarvis_spec1.pdf

   http://www.afjarvis.staff.shef.ac.uk/sudoku/russell_jarvis_spec2.pdf

   http://www.springerlink.com/content/g2278u057576xv1g/


   http://4c.ucc.ie/~hsimonis/sudoku.pdf
APPENDICES
## Appendix A

### Test Cases

<table>
<thead>
<tr>
<th>Test Number</th>
<th>Function</th>
<th>Result</th>
<th>Description</th>
<th>Expected results notes</th>
<th>Actual results notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Type number inside a cell</td>
<td>✔ Passed</td>
<td>Select a cell in the puzzle board</td>
<td>Test case is valid.</td>
<td>Test case accepted.</td>
</tr>
<tr>
<td>2</td>
<td>Type a character inside the cell</td>
<td>✔ Passed</td>
<td>Select a cell in the puzzle board</td>
<td>Test case is invalid.</td>
<td>Test case was rejected with the correct action.</td>
</tr>
<tr>
<td>3</td>
<td>Create new puzzle</td>
<td>✔ Passed</td>
<td>Select File from the menu</td>
<td>Test case is valid.</td>
<td>Test case was accepted</td>
</tr>
<tr>
<td>4</td>
<td>Solve a puzzle</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test Number</td>
<td>Function</td>
<td>Result</td>
<td>Description</td>
<td>Expected results notes</td>
<td>Actual results notes</td>
</tr>
<tr>
<td>-------------</td>
<td>----------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------</td>
<td>------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>5</td>
<td>Type number inside a cell</td>
<td>☑ Passed</td>
<td>Select a cell in the puzzle board</td>
<td>Test case is valid</td>
<td>Test case accepted.</td>
</tr>
<tr>
<td>6</td>
<td>Type a character inside the cell</td>
<td>☑ Passed</td>
<td>Select a cell in the puzzle board</td>
<td>Test case is invalid. And the application should not display the character.</td>
<td>Test case was rejected with the correct action.</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Name</th>
<th>Age</th>
<th>Gender</th>
<th>Experience</th>
<th>Difficulty</th>
<th>Overall Performance</th>
<th>Usability</th>
<th>Learnability</th>
<th>Enjoyment</th>
<th>Problem Solving</th>
<th>Puzzle Design</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>25</td>
<td>Male</td>
<td>Beginner</td>
<td>Easy</td>
<td>Excellent</td>
<td>Good</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Mary</td>
<td>30</td>
<td>Female</td>
<td>Intermediate</td>
<td>Medium</td>
<td>Excellent</td>
<td>Good</td>
<td>Excellent</td>
<td>Excellent</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
<tr>
<td>Sam</td>
<td>35</td>
<td>Male</td>
<td>Advanced</td>
<td>Hard</td>
<td>Excellent</td>
<td>Good</td>
<td>Excellent</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
<td>Good</td>
</tr>
</tbody>
</table>

Comments: John found the application easy to use and the puzzles challenging, while Mary enjoyed the problem-solving aspect and found the design elegant. Sam appreciated the advanced difficulty level.

Great job!
Appendix C

Code Listing

Puzzle.java

```java
package sudoku;
/**
 * @author Ammar Jouwanah
 */
public class Puzzle {
    private Cell cells[][];
    private boolean solved;
    private Difficulty difficulty;

    public Puzzle(Puzzle puzzle){
        this.solved = puzzle == null?false:puzzle.isSolved();
        this.difficulty = puzzle == null?Difficulty.NONE:puzzle.getDifficulty();
        cells = new Cell[9][9];
        for(int row=0;row < 9;row++){
            for(int col=0;col < 9;col++){
                if(puzzle == null)
                    cells[row][col] = new Cell(row,col,0,false);
                else
                    cells[row][col] = new Cell(row,col,
                                                puzzle.getCell(row, col).getSol(),
                                                puzzle.getCell(row, col).getValue(),
                                                puzzle.getCell(row, col).isGiven());
            }
        }
    }

    public Puzzle(){
        this(null);
    }

    public void clear(){
        for(int row=0;row < 9;row++)
            for(int col=0;col < 9;col++)
                if(!cells[row][col].isGiven())
                    cells[row][col].setValue(0);
    }

    public void setDifficulty(Difficulty difficulty){
        this.difficulty = difficulty;
    }

    public Difficulty getDifficulty(){
        return this.difficulty;
    }

    public boolean isSolved() {
        return solved;
    }

    public void setSolved(boolean solved) {
```

public Cell[][] getCells() {
    return cells;
}

public Cell getCell(int row, int col) {
    return cells[row][col];
}

public Cell[] getRowCells(Cell cell) {
    int cellRow = cell.getRow();
    Cell[] row = new Cell[9];
    for (int i=0;i<9;i++)
        row[i] = this.getCell(cellRow, i);
    return row;
}

public Cell[] getColCells(Cell cell) {
    int cellCol = cell.getCol();
    Cell[] col = new Cell[9];
    for (int i=0;i<9;i++)
        col[i] = this.getCell(i, cellCol);
    return col;
}

public Cell[] getBlockCells(Cell cell) {
    int row = (cell.getRow() / 3) * 3;
    int col = (cell.getCol() / 3) * 3;
    Cell[] block = new Cell[9];
    int counter = 0;
    for (int r = 0;r < 3;r++)
        for (int c = 0;c < 3;c++)
            block[counter++] = cells[row + r][col + c];
    return block;
}

public void showSolution() {
    if (this.solved)
        for (int row = 0;row < 9;row++)
            for (int col=0;col < 9;col++)
                cells[row][col].setValue(cells[row][col].getSol());
}

public boolean checkSolution() {
    if (this.solved)
        for (int row = 0;row < 9;row++)
            for (int col=0;col < 9;col++)
                if (cells[row][col].getValue() != cells[row][col].getSol())
                    return false;
    return true;
}

public static boolean checkConflict(Cell[] cells) {
    for (int i=0;i<cells.length -1;i++)
        for (int j=i+1;j<cells.length;j++)
            if (cells[i].getValue() != 0 && cells[i].getValue() == cells[j].getValue())
                return false;
    return true;
}

@Override
public boolean equals(Object object){
    this.solved = solved;
}
if(object instanceof Puzzle){
    Puzzle puzzle = (Puzzle)object;
    for(int row=0;row < 9;row++){
        for(int col=0;col < 9;col++){
            if(cells[row][col].getValue() != puzzle.getCell(row, col).getValue())
                return false;
        }
    }
    return true;
}
return false;
}

Cell.java

package sudoku;

/**<*
 * @author Ammar Jouwanah
 */

public class Cell {
    private int value;
    private int sol;
    private int row;
    private int col;
    private int block;
    private boolean given;

    public Cell(int row, int col, int sol, int value, boolean given){
        this.row = row;
        this.col = col;
        this.block = row/3*3 + col/3;
        this.given = given;
        this.value = value;
        this.sol = sol;
    }

    public Cell(int row, int col, int value, boolean given){
        this(row, col, (given? value:0) , value, given);
    }

    public Cell(int row, int col){
        this(row, col, 0, false);
    }

    public int getBlock() {
        return block;
    }

    public int getCol() {
        return col;
    }

    public boolean isGiven() {
        return given;
    }

    public void setGiven(boolean given){
        this.given = given;
    }

    public int getRow() {
        return row;
    }
}
public int getValue() {
    return value;
}

public void setValue(int value) {
    this.value = value;
}

public int getSol() {
    return sol;
}

public void setSol(int sol) {
    this.sol = sol;
}
}

package sudoku;

/**
 * @author Ammar Jouwanah
 */

public class BackTrackSolver {
    private int rowIndex;
    private int colIndex;
    private Puzzle puzzle;
    private int counter = 0;

    public BackTrackSolver(Puzzle puzzle) {
        this.puzzle = new Puzzle(puzzle);
        rowIndex = 0;
        colIndex = 0;
    }

    public boolean solveCell(Cell currentCell) {
        if (currentCell == null) {
            resetPosition();
            currentCell = nextCell();
        }
        int value = 0;
        boolean solved = false;
        while (!solved && value < 9) {
            counter++;
            value++;
            currentCell.setValue(value);
            currentCell.setSol(value);
            if (!((Puzzle.checkConflict(this.puzzle.getRowCells(currentCell))
                    && Puzzle.checkConflict(this.puzzle.getColCells(currentCell))
                    && Puzzle.checkConflict(this.puzzle.getBlockCells(currentCell)))))
                continue;
            Cell next = nextCell();
            if (next != null)
                solved = solveCell(next);
            else{
                solved = true;
                this.puzzle.setSolved(true);
            }
        }
        if (!solved) {
            currentCell.setValue(0);
            currentCell.setSol(0);
            previousCell();
        }
    }
}

BackTrackSolver.java
public boolean reverseSolveCell(Cell currentCell){
    if(currentCell == null){
        resetPosition();
        currentCell = nextCell();
    }
    int value = 10;
    boolean solved = false;
    while(!solved && value > 1){
        counter++;
        value--;
        currentCell.setValue(value);currentCell.setSol(value);
        if(!((Puzzle.checkConflict(this.puzzle.getRowCells(currentCell)) &&
                Puzzle.checkConflict(this.puzzle.getColCells(currentCell)) &&
                Puzzle.checkConflict(this.puzzle.getBlockCells(currentCell))))
            continue;
        Cell next = nextCell();
        if(next != null)
            solved = reverseSolveCell(next);
        else{
            solved = true;
            this.puzzle.setSolved(true);
        }
    }
    if(!solved){
        currentCell.setValue(0);currentCell.setSol(0);
        previousCell();
    }
    return solved;
}

private Cell nextCell(){
    colIndex++;
    if(colIndex > 8){
        colIndex = 0;
        rowIndex++;
    }
    if(rowIndex > 8){
        rowIndex = 0;
        colIndex = -1;
        return null;
    }
    if(puzzle.getCell(rowIndex, colIndex).isGiven())
        return nextCell();
    return puzzle.getCell(rowIndex, colIndex);
}

private Cell previousCell(){
    colIndex--;
    if(colIndex < 0){
        colIndex = 8;
        rowIndex--;
    }
    if(rowIndex < 0){
        rowIndex = 0;
        colIndex = -1;
        return null;
    }
    if(puzzle.getCell(rowIndex, colIndex).isGiven())
        return previousCell();
    return puzzle.getCell(rowIndex, colIndex);
```java
private void resetPosition()
{
    rowIndex = 0;
    colIndex = -1;
    counter = 0;
}

public Puzzle getSolvedPuzzle()
{
    ///////////////DEBUG/////////////
    //System.out.println(counter);
    ///////////////DEBUG/////////////
    if(this.puzzle.isSolved())
        return this.puzzle;
    return null;
}
}
```

PuzzleGenerator.java

```java
package sudoku;

/**
 * @author Ammar Jouwanah
 */
import java.util.ArrayList;
import java.util.Random;

public class PuzzleGenerator {
    private Puzzle puzzle;
    private ArrayList<Cell> givenCells;
    private Random randomGen;

    public PuzzleGenerator()
    {
        puzzle = new Puzzle();
        givenCells = new ArrayList<Cell>();
        randomGen = new Random();
    }

    public Puzzle getPuzzle(Difficulty diff)
    {
        if(diff == Difficulty.EASY) //35-45 Easy
            return getPuzzle(randomGen.nextInt(45 - 35 + 1) + 35);
        if(diff == Difficulty.MEDIUM) //30-35 Medium
            return getPuzzle(randomGen.nextInt(35 - 30 + 1) + 30);
        if(diff == Difficulty.HARD) //27-30 Hard
            return getPuzzle(randomGen.nextInt(30 - 28 + 1) + 27);

        return getPuzzle(40);
    }

    public Puzzle getPuzzle(int givens)
    {
        for(int i=0;i < 666;i++)
        {
            puzzle = new Puzzle();
            fillPuzzle();
            clearGivens();
            setGivens(givens);
            clearValues();

            BackTrackSolver solver = new BackTrackSolver(puzzle);
            solver.solveCell(null);
            Puzzle solvedPuzzle = solver.getSolvedPuzzle();
            if(solvedPuzzle == null)
                continue;
            else
                BackTrackSolver solver2 = new BackTrackSolver(puzzle);
        }
    }
```
solver2.reverseSolveCell(null);
Puzzle solvedPuzzle2 = solver2.getSolvedPuzzle();
if(solvedPuzzle2.equals(solvedPuzzle)){
    puzzle = solvedPuzzle;
    clearValues();
    puzzle.setDifficulty(gradeDifficulty(givens));
    return puzzle;
}
return new Puzzle();

private Difficulty gradeDifficulty(int givens){
    if(givens < 27)
        return Difficulty.EVIL;
    if(givens < 30) // 27-30 Hard
        return Difficulty.HARD;
    if(givens < 35) // 30-35 Medium
        return Difficulty.MEDIUM;
    if(givens < 45) // 35-45 Easy
        return Difficulty.EASY;
    return Difficulty.NONE;
}

private Cell getRandomCell(){
    int row = randomGen.nextInt(5);
    int col;
    if(row==4)
        col = randomGen.nextInt(4);
    else
        col = randomGen.nextInt(9);
    return puzzle.getCell(row,col);
}

private void setCell(Cell cell){
    int counter = 0;
    do{
        cell.setValue(randomGen.nextInt(9)+1);
        counter++;
    }while(counter<300 && !Puzzle.checkConflict(this.puzzle.getRowCells(cell)) &&
        Puzzle.checkConflict(this.puzzle.getColCells(cell)) &&
        Puzzle.checkConflict(this.puzzle.getBlockCells(cell)));
    cell.setSol(cell.getValue());
    cell.setGiven(true);
}

private void clearGivens(){
    for(int row=0;row<9;row++)
        for(int col=0;col<9;col++)
            puzzle.getCell(row, col).setGiven(false);
}

private void resetValues(){
    for(int row=0;row<9;row++)
        for(int col=0;col<9;col++)
            puzzle.getCell(row, col).setValue(puzzle.getCell(row, col).getSol());
}

private void clearValues(){
    for(int row=0;row<9;row++)
        for(int col=0;col<9;col++)
            if(!puzzle.getCell(row, col).isGiven())
                puzzle.getCell(row, col).setValue(0);
private void setGivens(int givensCount){
    givenCells.clear();
    if(givensCount%2 != 0){
        givenCells.add(puzzle.getCell(4,4));
    }
    for(int i=0;i<givensCount/2;i++){  
        Cell cell = getRandomCell();
        if(givenCells.contains(cell)){
            i--;
            continue;
        }
        givenCells.add(cell);
        givenCells.add(puzzle.getCell(8-cell.getRow(), 8-cell.getCol()));
    }
    for(Cell c:givenCells)
        c.setGiven(true);
}
private void fillPuzzle(){
    Puzzle solvedPuzzle;
    while(true){
        puzzle = new Puzzle();
        startValues();
        BackTrackSolver solver = new BackTrackSolver(puzzle);
        solver.solveCell(null);
        solvedPuzzle = solver.getSolvedPuzzle();
        if(solvedPuzzle != null){
            break;
        }
    }
    this.puzzle = solvedPuzzle;
}
private void startValues(){
    setCell(puzzle.getCell(0, 0));
    setCell(puzzle.getCell(0, 3));
    setCell(puzzle.getCell(0, 6));
    setCell(puzzle.getCell(1, 1));
    setCell(puzzle.getCell(1, 4));
    setCell(puzzle.getCell(1, 7));
    setCell(puzzle.getCell(2, 2));
    setCell(puzzle.getCell(2, 5));
    setCell(puzzle.getCell(2, 8));
    setCell(puzzle.getCell(3, 0));
    setCell(puzzle.getCell(3, 3));
    setCell(puzzle.getCell(3, 6));
    setCell(puzzle.getCell(5, 2));
    setCell(puzzle.getCell(5, 5));
    setCell(puzzle.getCell(5, 8));
    setCell(puzzle.getCell(6, 0));
    setCell(puzzle.getCell(6, 3));
    setCell(puzzle.getCell(6, 6));
    setCell(puzzle.getCell(8, 2));
    setCell(puzzle.getCell(8, 5));
    setCell(puzzle.getCell(8, 8));
}

PuzzleFileManager.java

```java
package sudoku;
```
/**
 * @author Ammar Jouwanah
 */

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import org.w3c.dom.*;
import org.xml.sax.SAXException;

public class PuzzleFileManager {

    public static Puzzle readPuzzle(File file) {
        Puzzle puzzle = new Puzzle();
        try {
            DocumentBuilderFactory docBuilderFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
            Document doc = docBuilder.parse(file);

            // normalize text representation
            doc.getDocumentElement().normalize();
            if (!doc.getDocumentElement().getNodeName().equals("puzzle")) {
                return null;
            }

            puzzle.setSolved(Boolean.parseBoolean(doc.getDocumentElement().getAttribute("solved")));
            puzzle.setDifficulty(Difficulty.valueOf(doc.getDocumentElement().getAttribute("difficulty")));

            NodeList listOfCells = doc.getElementsByTagName("cell");
            for (int i = 0; i < listOfCells.getLength(); i++) {
                Node cell = listOfCells.item(i);
                NamedNodeMap attributes = cell.getAttributes();
                for (int j = 0; j < attributes.getLength(); j++) {
                    int row = Integer.parseInt(attributes.getNamedItem("row").getNodeValue());
                    int col = Integer.parseInt(attributes.getNamedItem("col").getNodeValue());
                    puzzle.getCell(row, col).setSol(Integer.parseInt(attributes.getNamedItem("sol").getNodeValue()));
                    puzzle.getCell(row, col).setValue(Integer.parseInt(attributes.getNamedItem("value").getNodeValue()));
                    puzzle.getCell(row, col).setGiven(Boolean.parseBoolean(attributes.getNamedItem("given").getNodeValue()));
                }
            }
        } catch (ParserConfigurationException | SAXException | IOException | DOMException e) {
            return null;
        }
        return puzzle;
    }

    public static boolean writePuzzle(Puzzle puzzle, File file) {
        StringBuilder str = new StringBuilder();
        FileWriter fw;
        if (!file.getName().endsWith(".puzzle")) {
            String newFile = file.getAbsolutePath().trim() + ".puzzle";
            file = new File(newFile);
        }

        str.append("<puzzle solved="").append(puzzle.isSolved()).append("">");

        return null;
    }
}

str.append(" difficulty=").append(puzzle.getDifficulty()).append("/>
");
for(int row=0;row<9;row++)
    for(int col=0;col<9;col++){
        Cell cell = puzzle.getCell(row, col);
        str.append("cell ");
        str.append("row=").append(cell.getRow()).append(" ");
        str.append("col=").append(cell.getCol()).append(" ");
        str.append("sol=").append(cell.getSol()).append(" ");
        str.append("value=").append(cell.getValue()).append(" ");
        str.append("given=").append(cell.isGiven()).append("/>
");
    }
    str.append("</puzzle>
");
try{
    if(!file.exists())
        file.createNewFile();
    fw = new FileWriter(file);
    fw.write(str.toString());
    fw.close();
} catch(IOException e){
    return false;
}
return true;
}

CellJTextField.java

package sudoku.gui;
/**
 * @author Ammar Jouwanah
 */
import java.awt.Font;
import javax.swing.JTextField;
import sudoku.Cell;
public class CellJTextField extends JTextField {
    private Cell cell;
    private boolean notationMood;
    public CellJTextField(Cell cell){
        super();
        this.cell = cell;
        this.setValue(cell.getValue());
        this.setEditable(!cell.isGiven());
        this.setHorizontalAlignment(JTextField.CENTER);
        this.setFont(new Font("Monospaced", Font.BOLD, 24));
        notationMood = false;
    }
    public void setCell(Cell cell){
        this.cell = cell;
        this.setValue(cell.getValue());
        this.setEditable(!cell.isGiven());
        notationMood = false;
    }
    private void setNotationMood(boolean status){
        if(status){
            notationMood = true;
            this.setFont(new Font("Monospaced", Font.BOLD, 14));
        } else {
            notationMood = false;
            this.setFont(new Font("Monospaced", Font.BOLD, 24));
        }
    }
}
else {
    notationMood = false;
    this.setFont(new Font("Monospaced", Font.BOLD, 24));
}

@Override
public void setText(String str){
    //do nothing
}

public void setValue(int value) {
    if(value > 0 && value < 10){
        setNotationMood(false);
        super.setText(value + "");
        cell.setValue(value);
    } else{
        super.setText(" ");
        cell.setValue(0);
    }
}

public int getValue(){
    return cell.getValue();
}

public int getSol(){
    return cell.getSol();
}

public int getCol() {
    return cell.getCol();
}

public int getRow() {
    return cell.getRow();
}

public boolean isGiven() {
    return cell.isGiven();
}

public void setGiven(boolean given){
    cell.setGiven(given);
}

public void update(){
    notationMood = false;
    if(cell.getValue() == 0)
        super.setText(" ");
    else
        super.setText(cell.getValue() + "");
}

History.java

package sudoku.gui;

/**
 */
* @author Ammar Jouwanah
public class History {
    public int row, col, oldValue, newValue;

    public History(int row, int col, int oldValue, int newValue) {
        this.row = row;
        this.col = col;
        this.oldValue = oldValue;
        this.newValue = newValue;
    }
}

package sudoku.gui;

/**
 * @author Ammar Jouwanah
 */
import java.awt.BorderLayout;
import java.awt.FlowLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;
import java.io.File;
import javax.swing.*
import sudoku.*;
public class MainJFrame extends JFrame {

private TimerJPanel timerPanel;
private PuzzleJPanel puzzlePanel;
private JMenuBar menuBar;
private JMenu fileMenu, viewMenu, puzzleMenu, helpMenu;
private JMenuItem newMenuItem, openMenuItem, saveMenuItem, exitMenuItem;
private JCheckBoxMenuItem timerChkBox, toolBarChkBox, showMistakesChkBox;
private JMenuItem undoMenuItem, redoMenuItem;
private JMenuItem clearMenuItem, solveMenuItem;
private JMenuItem userManualMenuItem, aboutMenuItem;
private JToolBar toolBar;
private JToggleButton timerButton;
private Puzzle puzzle;

/** Constructor to setup the game and the GUI */
public MainJFrame() {
    super("Sudoku");
PuzzleGenerator puzzleGen = new PuzzleGenerator();
Puzzle startPuzzle = puzzleGen.getPuzzle(Difficulty.NONE);
initComponents(startPuzzle);
this.setVisible(true);
}

private void solvePuzzle() {
    if (timerPanel.isRunning())
timerPanel.stop();
    if (this.puzzle.isSolved()){
this.puzzle.showSolution();
this.puzzlePanel.update();
}
else{
BackTrackSolver solver = new BackTrackSolver(this.puzzle);
solver.solveCell(null);
Puzzle solvedPuzzle = solver.getSolvedPuzzle();
if(solvedPuzzle == null)
JOptionPane.showMessageDialog(this,"Buzzle can't be solved");
else{
    this.puzzle = solvedPuzzle;
puzzlePanel.setPuzzle(this.puzzle);
}
}

private void clearPuzzle() {
    this.puzzle.clear();
    this.puzzlePanel.update();
}

private void savePuzzle() {
    JFileChooser fc = new JFileChooser();
    fc.setFileFilter(new PuzzleFilter());
    fc.setSelectedFile(new File("Untitled.puzzle"));
    int returnVal = fc.showSaveDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION){
        if(PuzzleFileManager.writePuzzle(puzzle, fc.getSelectedFile()))
            JOptionPane.showMessageDialog(this,"File created.");
        else
            JOptionPane.showMessageDialog(this,"Error writing file.");
    }
}

private void openPuzzle() {
    timerPanel.reset();
    Puzzle openedPuzzle;
    JFileChooser fc = new JFileChooser();
    fc.setFileFilter(new PuzzleFilter());
    int returnVal = fc.showOpenDialog(this);
    if (returnVal == JFileChooser.APPROVE_OPTION){
        openedPuzzle = PuzzleFileManager.readPuzzle(fc.getSelectedFile());
        if(openedPuzzle == null)
            JOptionPane.showMessageDialog(this,"Error reading file.");
        else{
            this.puzzle = openedPuzzle;
puzzlePanel.setPuzzle(this.puzzle);
        }
    }
}

private void newPuzzle(){
    timerPanel.reset();
    NewPuzzleJDialog test = new NewPuzzleJDialog(this,true);
test.setVisible(true);
}

public void loadPuzzle(Puzzle puzzle){
    this.puzzle = puzzle;
puzzlePanel.setPuzzle(this.puzzle);
}

private void helpWindow(){
}

private void quit(){
    System.exit(0);
}

private void showTimer();
private void showTimerTB(){
    timerPanel.setVisible(timerButton.isSelected());
    timerChkBox.setSelected(timerButton.isSelected());
    this.pack();
}

private void showToolBar(){
    toolBar.setVisible(toolBarChkBox.isSelected());
    this.pack();
}

private void initComponents(Puzzle puzzle){
    this.puzzle = puzzle;
    puzzlePanel = new PuzzleJPanel(this.puzzle);
    timerPanel = new TimerJPanel();
    timerPanel.setVisible(false);

    //////////////////create and set menus///////////////////////
    menuBar = new JMenuBar();
    fileMenu = new JMenu("File");
    fileMenu.setMnemonic(KeyEvent.VK_F);
    viewMenu = new JMenu("View");
    viewMenu.setMnemonic(KeyEvent.VK_V);
    puzzleMenu = new JMenu("Puzzle");
    puzzleMenu.setMnemonic(KeyEvent.VK_P);
    helpMenu = new JMenu("Help");
    helpMenu.setMnemonic(KeyEvent.VK_H);

    newMenuItem = new JMenuItem("New Puzzle");
    newMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_N, KeyEvent.CTRL_DOWN_MASK));
    newMenuItem.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent event) {
            newPuzzle();
        }
    });

    openMenuItem = new JMenuItem("Open Puzzle");
    openMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, KeyEvent.CTRL_DOWN_MASK));
    openMenuItem.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent event) {
            openPuzzle();
        }
    });

    saveMenuItem = new JMenuItem("Save Puzzle");
    saveMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_O, KeyEvent.CTRL_DOWN_MASK));
    saveMenuItem.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent event) {
            savePuzzle();
        }
    });

    exitMenuItem = new JMenuItem("Exit");
    exitMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_X, KeyEvent.CTRL_DOWN_MASK));
    exitMenuItem.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent event) {
        }
    });
}
public void actionPerformed(ActionEvent event) {
    quit();
}
};
toolBarChkBox = new JCheckBoxMenuItem("Show ToolBar");
toolBarChkBox.setSelected(true);
toolBarChkBox.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        showToolBar();
    }
});
timerChkBox = new JCheckBoxMenuItem("Show Timer");
timerChkBox.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        showTimer();
    }
});
showMistakesChkBox = new JCheckBoxMenuItem("Show Mistakes");
showMistakesChkBox.setSelected(true);
showMistakesChkBox.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        puzzlePanel.setShowMistakes(showMistakesChkBox.isSelected());
    }
});
undoMenuItem = new JMenuItem("Undo");
undoMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Z,
    KeyEvent.CTRL_DOWN_MASK));
undoMenuItem.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent event) {
        puzzlePanel.undo();
    }
});
redoMenuItem = new JMenuItem("Redo");
redoMenuItem.setAccelerator(KeyStroke.getKeyStroke(KeyEvent.VK_Y,
    KeyEvent.CTRL_DOWN_MASK));
redoMenuItem.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent event) {
        puzzlePanel.redo();
    }
});
clearMenuItem = new JMenuItem("Clear Puzzle");
clearMenuItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        clearPuzzle();
    }
});
solveMenuItem = new JMenuItem("Solve Puzzle");
solveMenuItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        solvePuzzle();
    }
});
userManualMenuItem = new JMenuItem("User Manual");
userManualMenuItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        helpWindow();
    }
});
aboutMenuItem = new JMenuItem("About");
aboutMenuItem.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent event) {
        //show about;
    }
});

menuBar.add(fileMenu);
menuBar.add(viewMenu);
menuBar.add(puzzleMenu);
menuBar.add(helpMenu);
fileMenu.add(newMenuItem);
fileMenu.add(openMenuItem);
fileMenu.addSeparator();
fileMenu.add(saveMenuItem);
fileMenu.addSeparator();
fileMenu.add(exitMenuItem);
viewMenu.add(toolBarChkBox);
viewMenu.add(timerChkBox);
viewMenu.add(showMistakesChkBox);
puzzleMenu.add(undoMenuItem);
puzzleMenu.add(redoMenuItem);
puzzleMenu.addSeparator();
puzzleMenu.add(clearMenuItem);
puzzleMenu.add(solveMenuItem);
helpMenu.add(userManualMenuItem);
helpMenu.add(aboutMenuItem);
this.setJMenuBar(menuBar);

/////////////////////////////////////////////////////
//////////////create and set tool bar////////////////
toolBar = new JToolBar("Toolbar", JToolBar.HORIZONTAL);
toolBar.setFloatable(false);
JButton newButton = new JButton(new ImageIcon(this.getClass().getResource("/resources/images/new.png")));
nobutton.setToolBarText("New Puzzle");
nobutton.setBorderPainted(false);
nobutton.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent event) {
        newPuzzle();
    }
});

JButton openButton = new JButton(new ImageIcon(this.getClass().getResource("/resources/images/open.png")));
openButton.setToolBarText("Open Puzzle");
openButton.setBorderPainted(false);
openButton.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent event) {
        openPuzzle();
    }
});

JButton saveButton = new JButton(new ImageIcon(this.getClass().getResource("/resources/images/save.png")));
saveButton.setToolBarText("Save current puzzle");
saveButton.setBorderPainted(false);
saveButton.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent event) {
        savePuzzle();
    }
});
JButton clearButton = new JButton(new
ImageIcon(this.getClass().getResource("/resources/images/clear.png")));
clearButton.setToolTipText("Clear");
clearButton.setBorderPainted(false);
clearButton.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent event) {
        clearPuzzle();
    }
});

JButton solveButton = new JButton(new
ImageIcon(this.getClass().getResource("/resources/images/solve.png")));
solveButton.setToolTipText("Solve Puzzle");
solveButton.setBorderPainted(false);
solveButton.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent event) {
        solvePuzzle();
    }
});

JButton undoButton = new JButton(new
ImageIcon(this.getClass().getResource("/resources/images/undo.png")));
undoButton.setToolTipText("Undo");
undoButton.setBorderPainted(false);
undoButton.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent event) {
        puzzlePanel.undo();
    }
});

JButton redoButton = new JButton(new
ImageIcon(this.getClass().getResource("/resources/images/redo.png")));
redoButton.setToolTipText("Redo");
redoButton.setBorderPainted(false);
redoButton.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent event) {
        puzzlePanel.redo();
    }
});

toggleButton = new JToggleButton(new
ImageIcon(this.getClass().getResource("/resources/images/timer-icon.png")));
toggleButton.setToolTipText("Show/Hide Timer");
//toggleButton.setBorderPainted(false);
toggleButton.addActionListener(new java.awt.event.ActionListener() {
    @Override
    public void actionPerformed(java.awt.event.ActionEvent event) {
        showTimerTB();
    }
});
NewPuzzleDialog.java

```java
package sudoku.gui;

import java.awt.Cursor;
import sudoku.Difficulty;
import sudoku.Puzzle;
import sudoku.PuzzleGenerator;

/**
 * @author Ammar
 */
public class NewPuzzleJDialog extends javax.swing.JDialog {

    /**
     * Creates new form NewPuzzleJDialog
     */
    public NewPuzzleJDialog(MainJFrame parent, boolean modal) {
        super(parent, modal);
        this.parent = parent;
        initComponents();
        givensLabel.setText("new puzzle with "+givensSlider.getValue()+" clues.");
        this.setLocationRelativeTo(parent);
    }

    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    private void initComponents() {
        difficultyRBGroup = new javax.swing.ButtonGroup();
        jLabel1 = new javax.swing.JLabel();
        easyRadioButton = new javax.swing.JRadioButton();
        medRadioButton = new javax.swing.JRadioButton();
        hardRadioButton = new javax.swing.JRadioButton();
        givensLabel = new javax.swing.JLabel();
        givensSlider = new javax.swing.JSlider();
    }
}
```
jSeparator1 = new javax.swing.JSeparator();
createButton = new javax.swing.JButton();
cancelButton = new javax.swing.JButton();
jLabel3 = new javax.swing.JLabel();

setDefaultCloseOperation(javax.swing.WindowConstants.DISPOSE_ON_CLOSE);
setTitle("Create Puzzle");

jLabel1.setText("Difficulty");
difficultyRBGroup.add(easyRadioButton);
easyRadioButton.setSelected(true);
easyRadioButton.setText("Easy");
easyRadioButton.setEnabled(false);
difficultyRBGroup.add(medRadioButton);
medRadioButton.setText("Medium");
medRadioButton.setEnabled(false);
difficultyRBGroup.add(hardRadioButton);
hardRadioButton.setText("Hard");
hardRadioButton.setEnabled(false);

givensLabel.setText("gLabel2");
givensSlider.setMajorTickSpacing(5);
givensSlider.setMaximum(45);
givensSlider.setMinimum(28);
givensSlider.setMinorTickSpacing(2);
givensSlider.setToolTipText(""yor a.
);
givensSlider.addChangeListener(new javax.swing.event.ChangeListener() {
    public void stateChanged(javax.swing.event.ChangeEvent evt) {
        sliderStateChanged(evt);
    }
});

createButton.setText("Create Puzzle");
createButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        createButtonActionPerformed(evt);
    }
});

cancelButton.setText("Cancel");
cancelButton.addActionListener(new java.awt.event.ActionListener() {
    public void actionPerformed(java.awt.event.ActionEvent evt) {
        cancelButtonActionPerformed(evt);
    }
});

jLabel3.setFont(new java.awt.Font("Tahoma", 1, 18)); // NOI18N
jLabel3.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
jLabel3.setText("Create new puzzle");

javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
    layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
    .addGap(javax.swing.GroupLayout.DEFAULT_SIZE, 130, Short.MAX_VALUE)
);

javax.swing.GroupLayout等内容
private void cancelButtonActionPerformed(java.awt.event.ActionEvent evt) {
    this.setVisible(false);
}

private void createButtonActionPerformed(java.awt.event.ActionEvent evt) {
    PuzzleGenerator puzzleGen = new PuzzleGenerator();
    this.setCursor(Cursor.getDefaultCursor());
    Puzzle puzzle = puzzleGen.getPuzzle(givensSlider.getValue());
    parent.loadPuzzle(puzzle);
    this.setCursor(Cursor.getDefaultCursor());
}
PuzzleFilter.java

```java
package sudoku.gui;

import java.io.File;
import javax.swing.filechooser.FileFilter;

/**
 * @author Ammar Jouwanah
 */
public class PuzzleFilter extends FileFilter{
    @Override
    public boolean accept(File f) {
        if (f.isDirectory())
            return true;
        return f.getName().endsWith(".puzzle");
    }
    @Override
    public String getDescription() {
        return "Sudoku puzzle files";
    }
}
```

PuzzleJPanel.java

```java
package sudoku.gui;
```
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.GridLayout;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.util.Stack;
import javax.swing.BorderFactory;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import sudoku.Puzzle;

/**
 * @author Ammar Jouwanah
 */
public class PuzzleJPanel extends JPanel implements MouseListener, KeyListener {
    private JPanel[] panels;
    private CellJTextField[][] cells;
    private Color defaultColor, blockHoverDefaultColor, hoverDefaultColor,
        blockHoverGivenColor, hoverGivenColor, givenColor;
    private Stack<History> undoHistory, redoHistory;
    private boolean showMistakes;
    private Puzzle puzzle;

    public PuzzleJPanel(Puzzle puzzle) {
        super();
        this.puzzle = puzzle;
        showMistakes = true;
        blockHoverDefaultColor = new Color(255, 255, 150);
        hoverDefaultColor = new Color(255, 255, 200);
        blockHoverGivenColor = new Color(200, 200, 150);
        hoverGivenColor = new Color(225, 225, 175);
        defaultColor = Color.WHITE;
        givenColor = new Color(225, 225, 225);

        undoHistory = new Stack<History>();
        redoHistory = new Stack<History>();

        panels = new JPanel[9];
        cells = new CellJTextField[9][9];

        for (int i = 0; i < 9; i++) {
            panels[i] = new JPanel();
            panels[i].setLayout(new GridLayout(3, 3));
            panels[i].setBorder(BorderFactory.createLineBorder(Color.BLACK));
        }

        for (int row = 0; row < 9; row++) {
            for (int col = 0; col < 9; col++) {
                int block = row / 3 * 3 + col / 3;

                cells[row][col] = new CellJTextField(puzzle.getCell(row, col));
                cells[row][col].addMouseListener(this);
                cells[row][col].addKeyListener(this);
                if (cells[row][col].isGiven())
                    cells[row][col].setBackground(givenColor);
                else
                    cells[row][col].setBackground(defaultColor);
                cells[row][col].setBorder(javax.swing.BorderFactory.createMatteBorder(1, 1, 0, 0, Color.DARK_GRAY));
                panels[block].add(cells[row][col]);
            }
        }
    }
```java
this.setLayout(new GridLayout(3, 3));
for (int i = 0; i < 9; i++) {
    this.add(panels[i]);
}
this.setPreferredSize(new Dimension(50 * 9, 50 * 9));

public void update(){
    for (int row = 0; row < 9; row++)
        for (int col=0; col < 9; col++)
            cells[row][col].update();
}

public void undo(){
    if (undoHistory.size() > 0){
        History undo = undoHistory.pop();
        redoHistory.push(undo);
        cells[undo.row][undo.col].setValue(undo.oldValue);
    } else
        JOptionPane.showMessageDialog(this, "No more moves to undo");
}

public void redo(){
    if (redoHistory.size() > 0){
        History redo = redoHistory.pop();
        undoHistory.push(redo);
        cells[redo.row][redo.col].setValue(redo.newValue);
    } else
        JOptionPane.showMessageDialog(this, "No more moves to redo");
}

public void setShowMistakes(boolean showMstk){
    showMistakes = showMstk;
    if (puzzle.isSolved())
        for (int row = 0; row < 9; row++)
            for (int col=0; col < 9; col++)
                if (showMistakes)
                    if (cells[row][col].getValue() != 0 && cells[row][col].getValue() != cells[row][col].getSol())
                        cells[row][col].setForeground(Color.RED);
}

public boolean isShowMistakes(){
    return showMistakes;
}

public void setPuzzle(Puzzle puzzle){
    this.puzzle = puzzle;
    for (int row = 0; row < 9; row++)
        for (int col=0; col < 9; col++)
            if (cells[row][col].isGiven())
                cells[row][col].setBackground(givenColor);
            else
                cells[row][col].setBackground(defaultColor);
}
@Override
public void mouseClicked(MouseEvent e) {
    //do nothing
}

@Override
public void mousePressed(MouseEvent e) {
    //do nothing
}

@Override
public void mouseReleased(MouseEvent e) {
    //do nothing
}

@Override
public void mouseEntered(MouseEvent e) {
    if(e.getSource() instanceof CellJTextField){
        CellJTextField tfCell = ((CellJTextField)e.getSource());
        int row = tfCell.getRow();
        int col = tfCell.getCol();
        int block = row/3 * 3 + col/3;
        Component[] blockCells = panels[block].getComponents();
        for (Component temp : blockCells){
            tfCell = ((CellJTextField)temp);
            if(tfCell.isGiven())
                tfCell.setBackground(blockHoverGivenColor);
            else
                tfCell.setBackground(blockHoverDefaultColor);
        }
        for(int i =0;i < 9;i++) {
            if(cells[row][i].isGiven())
                cells[row][i].setBackground(hoverGivenColor);
            else
                cells[row][i].setBackground(hoverDefaultColor);
            if(cells[i][col].isGiven())
                cells[i][col].setBackground(hoverGivenColor);
            else
                cells[i][col].setBackground(hoverDefaultColor);
        }
    }
}

@Override
public void mouseExited(MouseEvent e) {
    if(e.getSource() instanceof CellJTextField){
        CellJTextField tfCell = ((CellJTextField)e.getSource());
        int row = tfCell.getRow();
        int col = tfCell.getCol();
        int block = row/3 * 3 + col/3;
        Component[] blockCells = panels[block].getComponents();
        for (Component temp : blockCells){
            tfCell = ((CellJTextField)temp);
            if(tfCell.isGiven())
                tfCell.setBackground(givenColor);
            else
                tfCell.setBackground(defaultColor);
        }
        for(int i =0;i < 9;i++) {
            if(cells[row][i].isGiven())
                cells[row][i].setBackground(givenColor);
            else
                cells[row][i].setBackground(givenColor);
        }
    }
}
cells[row][i].setBackground(defaultColor);
if (cells[i][col].isGiven())
cells[i][col].setBackground(givenColor);
else
    cells[i][col].setBackground(defaultColor);
}

@Override
public void keyTyped(KeyEvent e) {
    if (e.getSource() instanceof JTextField) {
        JTextField tfCell = ((JTextField)e.getSource());
        if (!tfCell.isGiven()) {
            char c = e.getKeyChar();
            e.consume();
            if (((c >= '1') && (c <= '9')) {
                undoHistory.push(new History(tfCell.getRow(),
                                           tfCell.getCol(), tfCell.getValue(),
                                           Character.getNumericValue(c)));
                tfCell.setValue(Character.getNumericValue(c));
                if (showMistakes)
                    if (puzzle.isSolved()) {
                        if (tfCell.getValue() != 0 && tfCell.getValue() !=
                            tfCell.getSol())
                            tfCell.setForeground(Color.RED);
                        else
                            tfCell.setForeground(Color.BLACK);
                    } else
                        JOptionPane.showMessageDialog(this, "Congratulations, the puzzle is solved.");
                else if ((c == KeyEvent.VK_BACK_SPACE) || (c == KeyEvent.VK_DELETE)) {
                    undoHistory.push(new History(tfCell.getRow(),
                                                   tfCell.getCol(), tfCell.getValue(), 0));
                    tfCell.setValue(0);
                }
            }
        }
    }
}

@Override
public void keyPressed(KeyEvent e) {
    //do nothing
}

package sudoku.gui;
import java.awt.Dimension;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JLabel;
import javax.swing.JPanel;
/**
 * @author Ammar Jouwanah
 */
public class TimerJPanel extends JPanel implements Runnable{
    private long time;
    private boolean running;
    private JLabel timeLabel;
    private JButton runButton, resetButton;
    private Thread timer;

    public TimerJPanel(){
        time = 0;
        running = false;
        this.setLayout(new FlowLayout());
        timeLabel = new JLabel("00:00:00");
        timeLabel.setFont(new Font("Monospaced", Font.BOLD, 24));
        runButton = new JButton("Start");
        runButton.setPreferredSize(new Dimension(100,25));
        runButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent event) {
                runButton();
            }
        });
        resetButton = new JButton("Reset");
        resetButton.setPreferredSize(new Dimension(100,25));
        resetButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent event) {
                time = 0;
                timeLabel.setText(getTime(time));
            }
        });
        this.add(runButton);
        this.add(timeLabel);
        this.add(resetButton);
    }

    public String getTime(){
        return getTime(time);
    }

    private String getTime(long time){
        int hours,minutes,seconds;
        hours = (int) (time / 3600);
        minutes = (int) (time / 60 - hours * 60);
        seconds = (int) time - (hours * 3600 + minutes * 60);
        return (hours < 10? "0" + hours:hours) + ":" +
                (minutes < 10? "0" + minutes:minutes) + ":" +
                (seconds < 10? "0" + seconds:seconds);
    }

    private void runButton(){
        if(running){
            stop();
        } else{
            running = true;
            timer = new Thread(this);
            timer.start();
            runButton.setText("Pause");
        }
    }

    public boolean isRunning(){
        return running;
    }
}
public void stop()
{
    running = false;
    timer.stop();
    runButton.setText("Start");
}

public void reset()
{
    if(running){
        running = false;
        timer.stop();
    }
    time = 0;
    runButton.setText("Start");
    timeLabel.setText("00:00:00");
}

@Override
public void run()
{
    while(true){
        time++;
        timeLabel.setText(getTime(time));
        try {
            Thread.sleep(1000);
        } catch (InterruptedException ex) {
            //do nothing.
        }
    }
}