Sudoku Solver

Nikolas Pilavakis

MInf Project (Part 2) Report

Master of Informatics
School of Informatics
University of Edinburgh

2021

Abstract

Sudoku[15], in its simplest form, is defined as a $n^2 \times n^2$ grid that is initially filled in with a certain number of cells. The objective is to fill every cell with numbers 1 to n^2, without using any number more than once in the same row, column, or $n \times n$ block. Due to the puzzle's popularity, numerous variants have emerged, changing the size of the puzzle, or adding additional rules. In this report, the design, implementation, and evaluation of an android Sudoku solving app is discussed. The developed app allows users to solve a puzzle using pencil marks, while allowing them to request hints which are generated based on an extensive set on human solving techniques. The app was also evaluated by multiple users of variable Sudoku knowledge and familiarity with similar apps.

During the COVID-19 pandemic, the popularity of online apps has grown significantly[44, 36]. This observation motivated the idea of multiplayer Sudoku. An idea that is mostly unexplored, despite the popularity of the puzzle spanning multiple decades. Both competitive and cooperative Sudoku are examined, with an initial version of the latter being designed and implemented.

Acknowledgements

I would like to express my sincere gratitude towards my supervisor, professor Nigel Topham for his continuous guidance and support throughout the duration of the project.

I would also like to thank my family and friends for their constant support in these trying times.

Finally, I would like to thank each and every one of the people that took the time to participate in the project's evaluation.

Table of Contents

Chapter 1

Introduction

The first year of the project involved the development of an android app that can recognise a Sudoku puzzle using the phone's camera, and solve it using a backtracking algorithm. This included a basic version User Interface (UI) which was mainly used for display purposes. The goals for the second year of the project were initially concerned with fine-tuning the aforementioned implementation, evaluating the effect of different vision techniques to the performance of the recognition process, benchmarking and user evaluation. Shortly after the beginning of the academic year, discussion with my supervisor resulted in the project taking a completely different direction.

The focus on the project was shifted towards human solving of the puzzle. The main objective of the project was the design, implementation and evaluation of an android app that allows users to solve Sudoku puzzles and potentially request hints. As a result, this year's implementation is mostly unrelated to last year's work. Even the developed UI and the internal representation of the puzzle had to be adapted to include use of notes, also known as pencil marks. Later, the idea of multiplayer Sudoku emerged. A concept that was previously not even mentioned in literature and online Sudoku communities, let alone implemented. This was then added to the project's goals.

The resulting app can be used for solving of Sudoku puzzles. The app allows users to solve a puzzle using pencil marks, highlighting cells, or request a detailed hint generated from the current state of the puzzle. Furthermore, undo functionality was implemented, and incorrectly filled cells are highlighted. A simple version of cooperative multiplayer Sudoku was implemented as a proof of concept, using Firebase Firestore. A questionnaire was designed, which was used for the evaluation of the app.

In chapter 2, previous work is discussed, followed by a summary of commonly used human solving techniques and popular Sudoku variants. Similar apps are then analysed, and important features are identified. In chapter 3, the design of the user interface and the algorithms required for hints is discussed, followed by a conceptual design that can be used to adapt the app for variants and competitive multiplayer. Next, the implementation details are presented. Chapter 5 outlines the evaluation of the app, the results of which are analysed extensively. Finally, the achievements of the project are summarised, followed by its limitations and possible future work.

Chapter 2

Background

2.1 Previous work

As this project is a continuation of last year's work, a brief summary of the key features developed last year is necessary to put this year's work into perspective. The goal for the first year of the project was to develop an android app that uses the phone's camera to detect a puzzle and then displays the solution of the puzzle. The app was developed in Kotlin.

2.1.1 User Interface

The user interface developed included functionality necessary for the scanning of the puzzle, which is irrelevant for this part of the project. A graphical representation of the scan puzzle was implemented, where the user could choose to edit the puzzle or display a solution. Changes to the puzzle can then be discarded with the click of a button. The Editing feature can be adapted for the implementation of puzzle solving by the user.

2.1.2 Solving algorithm

For reasons described in section 2.2.2, the scanned puzzles were solved using a backtracking algorithm which solves the cell with the smallest number of options, limiting the width of the search tree. Pseudocode of the solving algorithm is shown in algorithm 1:

The algorithm maintains a set of possible values for each cell. This array of sets can be adapted to suit the needs of the hint feature.

Functionality that checks whether a puzzle satisfies the required constraints by checking every row, column and block was also developed.

Algorithm 1 SolveSudokuRevised(grid,setsArray)

Require: a set of illegal values for each empty cell. Sets are ordered by descending size and stored in an array.

1: $largestSet \in setsArray\ s.t.size(largestSet) == max\{size(A)|A \in setsArray\}$
2: **if** largestSet == NULL **then**
3: **return** true {Solving successful}
4: **else**
5: **for** $digit \in \{1..9\} - largestSet$ **do**
6: Assign digit to cell corresponding to largestSet.
7: UpdatedSets= UpdateSets(grid,sets,cell)
8: **if** UpdatedSets == NULL **then**
9: **return** false {Branch cannot lead to solution, backtrack}
10: **end if**
11: **if** SolveSudokuRevised(grid,UpdatedSets) **then**
12: **return** true {further explore branch}
13: **else**
14: Undo Assignment
15: **return** false {backtrack}
16: **end if**
17: **end for**
18: **end if**

Algorithm 2 UpdateSets(grid,sets,cell)

1: **for** each set in row,column, grid **do**
2: add content of updated cell to set
3: **if** set.size == 9 **then**
4: **return** NULL {Empty cell has no more legal values}
5: **end if**
6: **end for**
7: sort sets
8: **return** sets

2.2 Part one background summary

This chapter is a continuation of the background section written during the first year of the project. The majority of available literature was discussed in the report for part one. As many of the points previously covered are built upon in this report, a summary of the key points touched last year is provided in this section.

2.2.1 Sudoku Background

There are 6,670,903,752,021,072,936,960 valid Sudoku grids [1, 11], of which 5,472,730,538 are essentially different [12]. while fairly uncommon in printed format, valid puzzles with multiple correct solutions exist.

The minimum number of clues that must be present in a puzzle for the puzzle to have a unique solution is 17 [35, 29].

Multiple ways to determine the difficulty of a puzzle are present in literature. One method uses the total number of clues, or the lower bound of number of clues in each row or column as an inverse measure of difficulty. Puzzle difficulty can range from extremely easy to evil. [27]

The aforementioned method ignores the position of the clues in the puzzle. A more sophisticated method of determining the difficulty of a puzzle is based on the complexity of the techniques that must be used for the puzzle to be solved by a human.[20]. Use of this method requires a human solving the puzzle before the difficulty can be determined. The various solving techniques are of prime importance for this project and are discussed in section 2.3. A continuous scale of difficulty is also available, ranking puzzles with a score from 1 to 4 [10].

2.2.2 Solving the Sudoku

Solving Sudoku puzzles programmatically is proven to be ASP-complete and hence NP complete [62]. Therefore, the complexity of any known approach increases exponentially with the size of the grid. Backtracking[18] is the most popular algorithm used for Sudoku solving. Due to the practical benefits of backtracking, namely speed and guaranteed finding of solutions, a version of backtracking was implemented for the first year of the project. Examples in literature that solved Sudoku using this algorithm include: [19, 20, 27, 25, 51, 30, 2, 19]

A variety of soft-computing algorithms were also proposed. Such as Artificial Bee Colony (ABC) [24, 45, 43, 63, 32] and Genetic Annealing (GA) [33, 34, 48, 50, 60, 5].

Solutions on FPGAs were also proposed [31, 59, 14, 9, 23, 61, 21].

Comparison between backtracking, simulated annealing and genetic algorithms verified backtracking to be the most effective approach. [22]

Other approaches used are:

- Boolean algebra [6]

- Boolean satisfiability and Constrained Programming (CP) [3, 54, 46, 7]
- Metaheuristics [39, 28, 55, 56, 57, 17, 4]
- Rewriting rules [49]
- Sinkhorn balancing [38]
- Entropy minimization [16]
- Hall's marriage theorem [58]
- Integer programming [4],
- Harmony search [13]
- membrane computing [8]
- Hybrid Ant Colony Optimization (ACO) [47]

2.3 Solving techniques

Unless otherwise noted, all images used in this section were created using Hodoku[1]. Since most publications are concerned with solving the puzzles programmatically, human solving techniques are usually omitted. The most detailed description of solving techniques is given by Jana et.al. [20]. On the contrary, multiple websites explaining the different solving techniques exist. The most commonly mentioned techniques are outlined in this section. Use of these techniques allows for solving of most (if not all) printed puzzles without requiring any guesswork. When the same technique can be found under different names, alternative names are mentioned in parentheses. Techniques are presented in order of increasing difficulty.

2.3.1 Houses in Sudoku

It is common in solving literature to refer to individual rows, columns or blocks as "houses". A standard Sudoku has 9 of each type of house, resulting in a total of 27 houses.

2.3.2 Pencil marks

Although use of pencil marks is not a technique per se, it is necessary for most of the techniques outlined in this section. Pencil marks can be summarised as follows: For every empty cell, small numbers are noted indicating all the legal number that can be placed in the cell, without violating the puzzle constraints. Pencil marks are gradually removed as the puzzle is solved. An example of pencil marks is shown in figure 2.1.

For consistency and easier pattern recognition, pencil marks are always placed in the same position in the cell, resembling a dial pad, as shown in figure 2.2

[1] http://hodoku.sourceforge.net/en/index.php

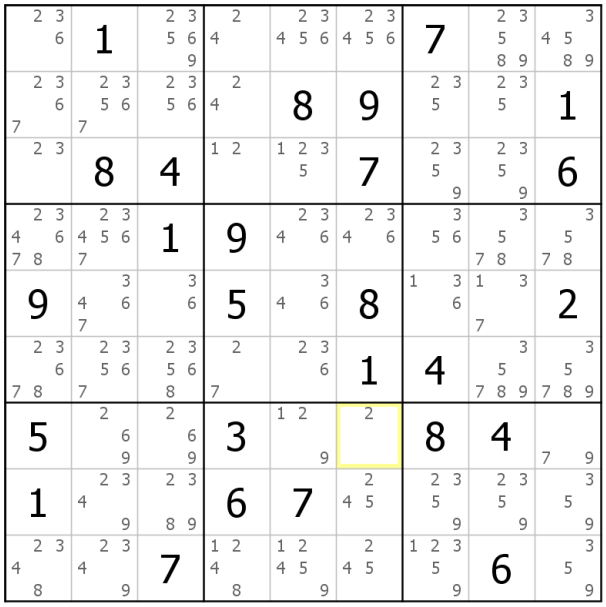


Figure 2.1: Use of pencil marks

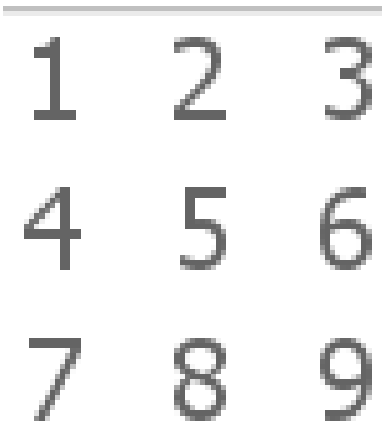


Figure 2.2: Positioning of pencil marks

2.3.3 Open singles

This technique is the most basic form of Sudoku solving and does not require pencil mark use. Simply, when there is a house with only one missing cell, there is only one legal candidate for that cell.

2.3.4 Naked singles (lone singles)

Naked singles encapsulate the essence of pencil marks and are a generalisation of Open singles. When there is only one pencil mark left for a cell, there is only one valid number for that cell. An example is the highlighted cell in figure 2.1

2.3.5 Hidden singles

Hidden singles are a more advanced technique as they are harder to spot. When a number appears as a pencil mark only once for a given row, column, or block, it is the solution to that cell. Highlighting all squares where a number appears as a pencil mark, helps spot hidden singles, and is widely used by Sudoku applications. Consistent positioning of pencil marks is crucial for the detection of hidden singles, as shown in figure 2.3. In the row example, the highlighted cell is the only cell of that row that for which 2 is a valid solution. Similarly, number 3 is the only a candidate in the highlighted squares for both column and block examples. To keep things concise, only one type of house will be used for demonstration hereafter, as the 3 types of houses (rows, columns and blocks) are symmetric.

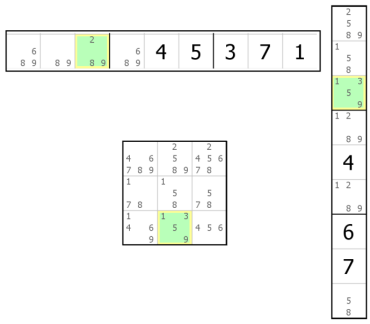


Figure 2.3: Hidden singles for a single row,column or block

2.3.6 Naked pairs

Naked pairs are not a direct solving technique, but can be a powerful tool for elimination of pencil marks. When two cells of the same house have the exact same two pencil marks, these two pencil marks can be eliminated from every other cell in the house. An example is shown in figure 2.4.

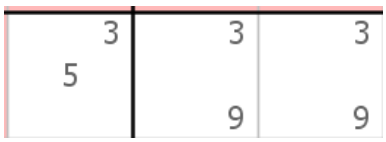


Figure 2.4: The two rightmost cells form a naked pair, leaving 5 as the only option for the leftmost cell (naked single)

2.3.7 Naked triplets and quartets

Naked triples and quartets follow the exact same logic as naked pairs but are far less common. As one would expect, when three cells with the exact same three pencil marks occur in the same house, those three pencil marks can be eliminated from every other cell in the same house. Similarly, the same applies for 4 pencil marks.

2.3.8 Omission (Pointing, blocking, claiming, intersection)

Omission is another strong technique for elimination of pencil marks. Various naming conventions exist in literature, however, all of them follow the same principle.

When a pencil mark occurs only within one block for any given row or column, the observed number must be placed in that row or column. Hence, the corresponding pencil mark can be removed from the rest of the block. Similarly, when a pencil mark occurs only within a single row or column for a specific block, it must be placed in that block and the corresponding pencil marks for the same row or column outside the block can be removed.

2.3.9 Hidden pairs

Hidden pairs are a more advanced version of naked pairs. While two pencil marks only appear for only two cells of the same house, they are not the only pencil marks found in those two cells.

2.3.10 Hidden triplets and quartets

Hidden triplets follow the same intuition as hidden pairs; however 3 pencil marks only appear on 3 cells only for the same house. Similarly for quartets.

This technique concludes the set of harder techniques and should be enough to solve at least 90% of well-formed puzzles. The techniques described below are incredibly rare and very hard to spot but are mentioned for completeness.

2.3.11 X wing

This technique involves a pair of rows or columns. If a pencil mark only appears exactly twice in the two rows at the same columns (or in the two columns at the same rows) then the same pencil mark can be eliminated from the rest of the columns (or rows). An example is shown in figure 2.5

2.3.12 Swordfish

The swordfish technique is a variation of the X wing technique, in which three rows (or columns) that contain a pencil mark exactly twice are examined. All candidates of that pencil marks in the corresponding columns (or rows) that are not part of the examined rows (or columns) can be eliminated. In the example shown in figure 2.6 the pencil mark 2 is used using 3 rows to eliminate the candidate in two other cells.

2.3.13 XY wing

The XY wing technique requires one cell with two possible candidates (pivot). We call these candidates X and Y. We then need two additional cells (pincers) facing the pivot with candidates X and Z and Y and Z respectively. Then, Z can be eliminated from the candidates of any cell facing both pincers

Figure 2.5: Xwing using two rows (orange) to eliminate a pencil mark on the column (green). Pencil mark 2 appears on all orange cells and can be eliminated from the green cell.

In the example shown in figure 2.7 the pivot has candidates 2 and 7 (X and Y respectively). the pincers have candidates 5 and 7 and 2 and 5 (Z is 5). The cell highlighted in green cannot contain 2, so that candidate is removed, revealing an open single.

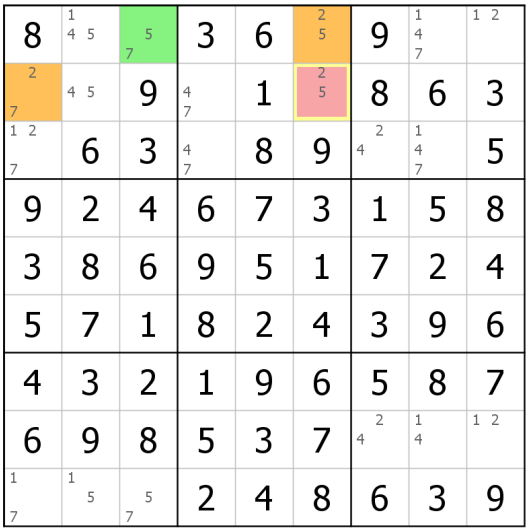


Figure 2.7: XY wing technique with pivot highlighted in green and pincers in orange to eliminate pencil mark 2 from cell highlighted in pink.

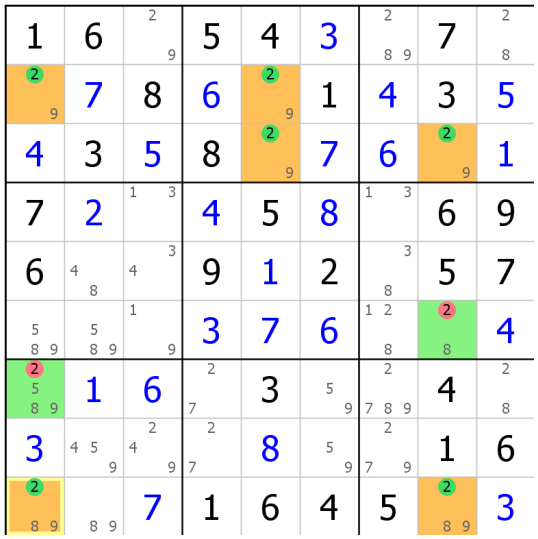


Figure 2.6: Swordfish using three rows (shown in orange) using pencil mark 2, which is then eliminated from cells shown in green, revealing a naked single.

2.3.14 Unique rectangle

Unique rectangles operate on the assumption that the puzzle has exactly one solution. As this is not always true for the puzzles discussed in this report, this technique is not discussed further.

2.4 Sudoku variations

Due to the popularity and the long history of Sudoku puzzles, different variants have emerged, some of which are quite popular. Variants be separated to two different categories. First, variants based on the size and representation of the puzzle or the symbols used. In that case, the same rules apply (i.e. no symbol must be placed in every house exactly once). Secondly, puzzles that add additional constraints to the puzzle exist, and can be much more interesting. Naturally, puzzles that combine multiple variants also exist. Expanding the solver implemented last year to accommodate for variants is discussed in section 3.4.2. Pictures of the variants that are discussed can be found in appendix A

2.4.1 Size and representation variants

Puzzles of different sizes are found across literature. It is worth noting that puzzles up to 25x25 can be filled using only a single character per cell, by using letter from the alphabet. Not all variants use square blocks. Some puzzles are divided into ”jigsaw-like” shapes and are therefore called jigsaw Sudoku. Jigsaw Sudoku are also referred to as geometric or irregular. Some common variants are:

- 4x4 grids with 2x2 houses (using numbers 1-4). See figure A.1
- 5x5 grids with jigsaws of size 5, also known as Logi-5 (using numbers 1-5). See

figure A.2

- 7x7 grids with jigsaws of size 7 (using numbers 1-7). See figure A.3
- 12x12 grids with 4x3 regions, also known as Sudozen (using literals 1-C or 1-12). See figures A.4 and A.5
- 16x16 grids with 4x4 houses, also known as hexadoku (using literals 1-F). See figure A.6
- 25x25 grids with 5x5 houses, also known as alphadoku (using numbers 1-25 or letters A-Y). See figures A.7 and A.8
- 100x100 grids with 10x10 houses, also known as Sudoku-zilla (using numbers 1-100). Figure not included due to low resolution.
- Colorku: A coloured circle is used to represent each number. Most commonly used for classic puzzles. See figure A.9

2.4.2 Rules variants

As with size variants, Sudoku with additional rules are also found across literature, increasing the constraints imposed. Some of the most common rule variations are discussed in this section. In most cases, additional symbols or shadowing are used to indicate the additional constraints.

- Sudoku X: Diagonals form two additional regions; numbers 1-9 can only appear once on each diagonal. See figure A.10
- Center dot Sudoku: Cells appearing at the center of each block form an additional region. In other words, numbers 1-9 can only appear one in the cells that are in the center of each block. See figure A.11
- Colour Sudoku: Cells are coloured using a variable number (x) of different colours, forming x additional regions. Numbers 1-9 can only appear once on each coloured region. See figure A.12. Not to be confused with Colorku.
- Hyper Sudoku: Four additional blocks (In which cells are shadowed), are added to the standard nine. See figure A.13
- Thermo Sudoku: Thermometers appear across the grid. Starting from the base of the thermometer, numbers in a thermometer have to be increasing towards the top. Thermometers can be overlapping. The number of givens in this variant is usually very low. The hardest examples of this variant usually have no givens. See figure A.14
- Arrow Sudoku: Arrows appear across the grid. The number appearing at the tail of the arrow, which is often circled, has to be equal to the sum of the numbers appearing in the path covered by the arrow. A similar variant with product instead of sum also exists. See figures A.15 and A.16
- Odd/even Sudoku: Symbols appearing in cells indicate that the cell should be filled with an even or odd number. Typically, circle is used for numbers and

square for evens. See figure A.17

- Consecutive Sudoku: Symbols (usually a small rectangle) appearing between cells indicate that the numbers placed in the two cells must be consecutive. See figure A.18

2.5 Existing applications

Due to the popularity of the puzzle, a plethora of applications allowing users to solve Sudoku exist. In an attempt to highlight interesting features, the most well-known applications are analysed in this section, followed by a comparison. As the purpose of this analysis is to determine how different apps are suited to the needs of the Sudoku community, emphasis will be given on the interesting features of each piece of software. Aspects such as bugs found or possible improvements to these apps were deemed irrelevant and are therefore not discussed. Some of the solutions analysed have been under constant development with the help of an active user base for many years. Inevitably, some of the features found are very sophisticated and polished. Such features would be impossible to develop to a satisfactory degree in a year by a single student. For completeness, software for different platforms was also evaluated, with additional attention to android apps.

2.5.1 Windows

2.5.1.1 Hodoku

Hodoku [2] is a Windows app developed in Java. The first version was published in 2009. It is regarded as one of, if not the most complete Sudoku solving app. It includes many different modes such as generating, solving, learning and analysing Sudoku puzzles. This section will only examine the solving features which are the most relevant to this project. More specifically, Hodoku implements:

- Pencil marks: Automatically generated and updated when a cell is filled by default. Can be changed in settings.
- Highlighting: Can select to include or exclude all cells that are candidates for a specific number from highlighting. An option to highlight cells with only two possible candidates is also available.
- Generate puzzles of variable difficulty, ranging from easy to evil.
- Vague hints: Suggesting use of a technique, optionally including the cells involved.
- Concrete hints: solving a cell or removing a pencil mark
- Colorku: Coloured circles are used instead of digits. Can be used as an aid for the visually impaired.
- Import/export puzzles with various formats

[2] http://hodoku.sourceforge.net/en/index.php

- Solution path for puzzles solved using the solving algorithm

2.5.2 Web

Web apps are more relevant to this project as some are, at least partially, compatible with mobile web. In this section, two of the most popular web apps are discussed.

2.5.2.1 Enjoy Sudoku (web)

Enjoy Sudoku [3] is a family of apps across all major platforms (Windows, Mac OS X, android, iOS, kindle, and web) with free and paid versions available for each platform, except web, where only a free version is available. Free versions include ads and limit the number of puzzles available per day.

The most attractive feature of enjoy sudoku is the simple and intuitive interface. Not many options are available, which makes the app friendly for users that might be new to Sudoku puzzles. Most important options have to do with pencil marks. Users can choose between automatic or manual generation and updating of pencil marks. The solution of the puzzle can be displayed with the click of a button. Puzzles of variable difficulty and small tutorials explaining the basics of Sudoku are also available. Hints are also very interesting. Variable ”levels” of hints are available upon requesting a hint. Initially, vague hints such as ” meditate about digit 2” are given. By clicking ”show more”, the corresponding house is highlighted and the hint becomes ”where in this block can you place a 2?”. Clicking ”show more” again fills the cell. Unfortunately, only the most basic techniques are implemented, with naked pairs being the most advanced technique witnessed. When a more advanced technique is required, a pencil mark is eliminated instead. It is assumed that the current state of the puzzle is compared to the solution to delete a candidate.

2.5.2.2 Sudoku slam

Sudoku slam [4] is a free web app developed since 2006 and is optimised for solving puzzles using a keyboard. The app allows for easy highlighting of different candidates and elimination of pencil marks. Two different solving modes are available, dictating when a cell is filled automatically and how pencil marks are kept. ”Traditional” solving mode requires the pencil marks to be filled in manually and only completes a cell automatically when it is the only unsolved cell in a given house. ”Sumo” mode automatically initialises and updates pencil marks when required. Hidden singles are also filled in automatically. As a result, solving speed is increased drastically. The app also features an offline mode, while it allows for puzzles to be created manually if desired. Puzzle progress is saved using cookies. Saving the current progress as a PDF is also available, allowing users to print the puzzle and continue using a pen and paper. Hint functionality is excellent, as all techniques discussed in section 2.3 are implemented. When a hint is requested, the corresponding cells are highlighted and the hint is dis-

[3] http://www.enjoysudoku.com/

[4] https://www.sudokuslam.com/

played. A short explanation of the hint is also provided, accompanied by a link to a website explaining the hint.

2.5.3 Android

As the objective of the project is to develop an android app, multiple android apps were analysed. These apps were selected based on recommendations from Sudoku solving forums and do not necessarily correspond to the apps with the most features or the most users. Apart from Enjoy Sudoku, all apps analysed are free and use ads to generate revenue, with an in-app purchase available to remove ads.

2.5.3.1 Enjoy sudoku

The web version of this app was discussed in section 2.5.2.1. Two android versions exist, a free and a paid[5] version. Both versions include all features mentioned for the web app. The only difference is that the free version allows the user to solve only one puzzle per day, using ads to generate revenue. The paid version was sold for $2.99 at the time of writing, with 3000 users. Unfortunately, the free version, which was far more popular, is no longer available as it was not updated in the recent years. It was therefore taken down as it is not compatible with newer android versions. It is interesting to note that the user interface was ported from the web interface. As a result, it is optimised and performs poorly.

2.5.3.2 Andoku 3

Andoku is one of the oldest android apps that maintain an active user base and are regularly updated. The first version available to the public was Andoku Sudoku 2 [6], which was released in 2011 and maintains 28,000 active users to this day. Its successor, Andoku 3 [7] was released in 2015 and has over a million downloads. The app implements all the features that are found in any modern Sudoku solving app, while maintaining a clean and simple UI. More specifically:

- Pencil marks: Can be edited with the click of a button. For harder puzzles, the option to automatically generate and update pencil marks is also available.
- Highlighting: When selecting a number from the input pad, all cells containing that number (either as a solution or a pencil mark) are highlighted.
- Puzzle variety: Available puzzle difficulty ranges from ”Easy” to ”Ultra extreme”. Apart from the traditional Sudoku format, variable difficulty puzzles for some of the variants mentioned in section 2.4.2 and their combinations are also available.

[5] https://play.google.com/store/apps/details?id=com.enjoysudoku.enjoysudoku&hl=en_GB&gl=US/

[6] https://play.google.com/store/apps/details?id=com.andoku.two.free&pcampaignid=pcampaignidMKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1/

[7] https://play.google.com/store/apps/details?id=com.andoku.two.free&pcampaignid=pcampaignidMKT-Other-global-all-co-prtnr-py-PartBadge-Mar2515-1/

- Hints: A plethora of hints are available, ranging from very easy (such as Hidden single) to ”Ultra extreme” (such as swordfish variations). Hints are given gradually. Initially, relevant houses are highlighted, followed by highlighting of the relevant pencil marks and concluding with the solution. Tutorials for all techniques are also available, with example puzzles and step by step guides.
- Importing puzzles: Puzzles can be imported using different features. Trivially, givens can be added to an empty board to construct a puzzle. Alternatively, A string containing the givens of the puzzle and '.' for empty cells can be input to generate the puzzle. Finally, printed puzzles can be scanned using the phone's camera.

2.5.3.3 Brainium Sudoku

Brainium Sudoku [8] is a regularly updated app, developed by Brainium studios with over 10 million downloads at the time of writing. The app contains all the basic features one would expect from a Sudoku app (pencil marks, highlighting, and basic hints) but not much else. Highlighting is done automatically and is not ideal as there is no apparent way to highlight all cells with a specific candidate. A gradual hint functionality is also implemented. The most advanced hint witnessed was intersection (referred to as omission in section 2.3). When more complex techniques are required, a pencil mark is deleted, or a cell is filled. Recent reviews of the app criticize the colour scheme used by the app, as it often painful to the eyes and the amount of information conveyed by the colours used is limited because of their excessive use. No options to import or export puzzles are available.

2.5.3.4 sudoku.com

sudoku.com [9] is the most popular android Sudoku solving app. Developed by Easybrain, it has over 50 million downloads and the ”editor's choice” at the time of writing. The app offers a large variety of puzzles of variable difficulty. Interestingly, Hexadoku (or 16x16 Sudoku) are also available within the app. In terms of functionality, pencil marks are implemented as usual, and an option to fill pencil marks automatically is available for harder puzzles. An option to fill singles automatically is also available. Due to the excessive number of advertisements in the app, the hint functionality was not thoroughly tested. However, during the limited testing, the result of a hint was either the deletion of a pencil mark or the filling of a cell, without explanation. All cells containing a specific number or pencil mark can be highlighted when clicking a number from the number pad. No option to import or export puzzles is available. Out of all the apps reviewed in this section, this app has the most intuitive and polished UI. It is very accessible for people new to the world of Sudoku, which explains its popularity.

[8] https://play.google.com/store/apps/details?id=com.brainium.sudoku.free&hl=en_GB&gl=US

[9] https://play.google.com/store/apps/details?id=com.easybrain.sudoku.android&hl=en_GB&gl=US

Comparison of Sudoku solving apps							
App name	OS	Free	Marks	highlighting	Hints	Portation	Colorku
Hodoku	Win	✓	Flexible	Flexible	Flexible	✓	✓
Enjoy Sudoku	Web	Limited	Auto	Auto	Gradual	✗	✗
Sudoku slam	Web	✓	Flexible	Flexible	Detailed	✓	✗
Enjoy Sudoku	And	✗	Auto	Auto	Gradual	✗	✗
Andoku 3	And	✓(Ads)	Flexible	Flexible	Gradual	✓	✗
Brainium	And	✓(Ads)	Flexible	Limited	Limited	✗	✗
sudoku.com	And	✓(Ads)	Flexible	Flexible	Auto	✗	✗

Table 2.1: Comparison of key features of Sudoku solving apps

2.5.4 Comparison of solutions

Applications from many different platforms were analysed in this section. Comparing them is no easy task, as a plethora of criteria must be taken into account. A summary of the key features relevant to this project are summarised in table 2.1. Hodoku is a clear winner on most categories. The only areas where it is lacking compared to the other apps is portability (taking into account that windows phones are not that popular) and variety, as it only offers traditional Sudoku only. It also does not allow scanning of puzzles using a camera, which could demotivate users from solving a puzzle they saw on a newspaper. Where applicable, the price to remove ads ranges from two to three pounds and should not be an issue for dedicated users. The quality of the UI is of prime importance, as a feature is not important unless it is used by the users. A simple and intuitive UI like the one implemented by the sudoku.com app is desired. Finally, excessive or unreasonable use of colour, as done by Brainium Sudoku, should be avoided.

2.6 Synchronization

To the best of our knowledge, multiplayer Sudoku is not discussed in literature. Furthermore, no multiplayer Sudoku implementations were found, the closest to a multiplayer realisation being websites that allow two users to compare their solving times on the same puzzle. As a result, the implementation of multiplayer functionality will be based on similar projects, such as online games synchronization.

Chapter 3

Design

3.1 User Interface

The user interface had to be adjusted allow users to solve puzzles on the phone, as last year's implementation prioritised displaying a puzzle over solving it. The main adjustments required boil down to three aspects: Displaying pencil marks, allowing users to edit them and highlighting all cells that contain a pencil mark.

3.1.1 Displaying pencil marks

Displaying pencil marks might seem like a trivial task at first. This however was not the case as such a feature was not forethought during last year's implementation, as the performance of the UI was a priority. A solution that introduced the least amount of changes was desirable. The only requirement in place regarding the display of pencil marks is that the each digit must appear on the same position relative to the cell when used as a pencil mark. Of course, the selected solution was also required to be efficient and not introduce any noticeable time delays or unpredictable behaviour.

Unfortunately, most pencil marks UI implementations using native android libraries that are available online are severely outdated and would not work properly on a modern android device. The code used by the apps analysed in section 2.5.3 is either not publicly available or based on a web framework. The latter option was deemed infeasible due to the author's inexperience with such frameworks. As the design decisions for the UI are implementation specific, further discussion is left for section 4.2

3.1.2 Editing of pencil marks

Adding or removing a pencil mark from a cell is a core feature of any Sudoku solving app. Keeping this feature as intuitive as possible while matching the implementation of similar apps was prioritised. Specifically, it was decided that switching between solving mode (filling empty cells) and pencil marks mode (editing pencil marks) will happen with the click of a button. In both modes, a button corresponding to each number 1-9 will be available below the puzzle. Users will be able to select a number

and then a cell. Depending on the mode that is active, the selected cell will be filled with that number or the corresponding pencil mark will be added or removed from the cell.

3.1.3 Highlighting cells

Selecting a number will also highlight all cells for which the number is a candidate. This is done by many Sudoku solving apps as it allows for easier spotting of techniques that can be used. More complex highlighting techniques were considered. For example, highlighting manually, highlighting all cells with two candidates, or highlighting all cells that do not include a selected candidate. These were deemed too complex for a general audience that is not required to have experience with Sudoku solving apps or Sudoku solving in general. As a result, highlighting functionality was kept to a minimum.

3.2 Hints

In this section, the design of the hints functionality is discussed. The goal of this functionality is to search for occurrences at which a solving technique could be used. Due to time and space constraints, techniques were limited to those discussed in section 2.3.

When a hint is found, a relevant message explaining the technique is displayed, and the relevant cells will be highlighted. The amount of pseudocode provided in this section was minimised to favour clarity and conciseness.

Unfortunately, the algorithms used by similar apps are not available in the public domain. As such, algorithms discussed were implemented from scratch and may not be optimal. Each algorithm will then be run in order of appearance until a hint is found. This ensures that the simplest possible hint will be given at any given time. Furthermore, it allows for optimisation of algorithms associated with more advanced techniques because assumptions can be made about the puzzle. For example, an algorithm for hidden pairs can assume that no hidden singles occur in the puzzle.

Most websites that explain the techniques outlined in section 2.3 claim that they are sufficient for solving 90-95% of all Sudoku puzzles. Experimenting with some puzzles confirmed that this number is not an exaggeration, as they are often sufficient for solving puzzles classified as ”evil”.

3.2.1 Initialising and updating candidates

Before any hint can be given, the candidates of each cell have to be initialised. For each cell, a set containing the legal values that can be entered into that cell is created. Sets are then stored in a list. The process followed for the initialisation is described in algorithm 3. It is worth noting that this algorithm goes over every cell exactly twice when creating a set. (once for the block and once for the row or block. The only exception is the cell for which the cell is created, which is visited exactly three times. Due to

the small size of the grid, the overhead associated with ensuring that each cell is visited exactly once is likely to outweigh the performance benefit of such an optimisation. As the algorithm is quite simple and is expected to terminate instantaneously for standard Sudoku, such an optimisation was deemed unnecessary complex.

Algorithm 3 Initialise constraints(grid)

```
initialise empty list
for cell in grid do
    if cell is not empty then
        set = {}
        add set to list.
        continue to next cell
    else
        set ={1..9}
        for nonempty cell in same house do
            set = set - {nonempty cell value}
        end for
        add set to list
    end if
end for
return list
```

When a pencil mark is edited, the corresponding set is adjusted accordingly. When a cell is filled, the sets of all cells in the same house are updated, as described in algorithm 4

Algorithm 4 Initialise constraints(grid, filled cell, filledNumber)

```
for cell in same house as filled cell do
    cellSet = cellSet - {filledNumber}
end for
```

A list containing the index of non-empty sets in ascending size order is also maintained. The list is created by sorting the list of sets in parallel with a list of indices. [1] Sorting is performed using Quicksort, which has an $\theta(nlogn)$ complexity on average. Due to space limitations, the implementation of Quicksort is omitted. Concretely, the first element of the list contains the index of the cell with the least candidates. The sorting algorithm is stable, meaning that the order is maintained in case of a tie. As such, cells with the smallest index will appear higher up on the list in case of a tie. This will allow for consistency during development and testing.

Henceforth, algorithms will assume the existence of the aforementioned populated data structures and any mention of their existence will be omitted to reduce repetition.

[1] Recall that the index of a cell is defined to be $row * 9 + col$ and ranges from 0 to 80

3.2.2 Open and naked singles

When the first element with the least candidates has only one candidate, that cell is a naked single. Determining whether it is an open single requires it to be the only empty cell in its row, column or box. Checking if the condition is filled is trivial but unnecessary, as all open singles can be generalised to be naked singles. When a naked single is spotted, the candidate and cell in question are trivial to find without requiring any additional computation.

3.2.3 Hidden singles

Hidden singles are slightly more complicated to spot, as the algorithm must check each individual house. It is worth noting that no apparent simplification can be made. For example, checking all rows and columns for a hidden single does not eliminate the possibility of a hidden single being present in a box. Finding the corresponding cell is then trivial. The process is described in algorithm 5. For efficiency, the algorithm marks the first time it finds a digit occurring as a candidate in every house. Traversing the house stops when a digit is found a second time, as that digit cannot be a hidden single. A hidden single is found if a house is traversed from start to finish.

Algorithm 5 HiddenSingles(grid,candidates)

```
1:  result = -1 { Index of hidden single}
2:  for digit ∈ 1..9 do
3:      found = false {reinitialised when checking a different house}
4:      for cell ∈ row, column, box do
5:          if digit ∈ candidates(cell) then
6:              if !found then
7:                  result = cell
8:                  found = true
9:              else
10:                 result = -1
11:                 next house
12:             end if
13:         end if
14:         return index {-1 if not found}
15:     end for
16: end for
```

3.2.4 Naked pairs, triples and quartets

Finding naked pairs is slightly more complicated as it involves two distinct stages. A textual description was deemed more suitable for this technique, mainly for simplicity and conciseness purposes. The first stage involves finding two cells in the same house that have the same two candidates. The candidates are then extracted. For the second stage, the same house is traversed a second time. If a third cell that has at least one of the two candidates is found in the same house, then a naked pair is detected. Otherwise,

the algorithm returns to the first stage and the next house is checked. For the hint functionality, the first two cells must be highlighted in a different colour than the third. It is worth noting that multiple cells might be benefit from this technique for a given house. The second stage described can be trivially adapted to return a list of cells from which pencil marks can be omitted instead of a single cell.

The same two-stage algorithm can then be easily adapted to finding naked triples or quarters. For this to happen, the first stage is changed to find two cells that share the same three or four candidates, respectively. The second stage is identical.

3.2.5 Hidden pairs, triples and quartets

Identification of hidden pairs by a human is often compared to identification of naked pairs. However, from a computation standpoint, the analogy is not entirely accurate. Hidden pairs only require identification of two cells. As such, the second stage of the algorithm described for naked pairs is not necessary.

A hidden pair is identified if two pencil marks appear only in two cells for any given house. One solution would be to check all 72 combination of digits for every house. This approach would not be too detrimental to the performance of the algorithm, as the search space is quite small. Alternatively, the search space is limited to pairs that exist in a cell in a given house. Tuples of digits that are checked for a given house are saved to ensure that each pair of digits is searched only once.The two cells are then returned.

Of course, the algorithm works only on the assumption that naked pairs have already been eliminated. If this prerequisite is not guaranteed, then the algorithm could return a naked pair, without identifying cell(s) in which pencil marks can be eliminated. In fact, such cells may not exist. As such, an additional constraint is added to ensure that at least one of the two cells has more than two candidates.

Hidden triples and quartets can then be found by adapting the algorithm described above.

3.2.6 Omission

Omission is a harder technique to spot computationally as multiple different scenarios that can be categorised as omission exist. As discussed in section 2.3, these scenarios are often separated using different naming conventions (Pointing, blocking, claiming and intersection). As the philosophy of identifying each one is the same, a distinction is not beneficial.

Put simply, the omission technique can be used if all occurrences of a given pencil mark for any row or column are in the same block. In that case, all other occurrences of that pencil mark in that row or column can be omitted. The symmetric nature of the Sudoku puzzles means that omission can happen in a block. All four possible scenarios can be summarised into one.

As an attempt to simplify the terminology, the house from which pencil marks are omitted from is referred to as the ”primary house” and the other one is referred to as

the ”secondary house”. The algorithm traverses all secondary houses (rows, columns, blocks) and checks if all occurrences of some digit appear within the same primary house (blocks, rows, columns). This idea can be generalised by grouping cells within secondary houses into groups of three cells. If a digit appears only in one group of cells, then there is a possibility that at least one pencil mark can be omitted from the primary house.

For the sake of an example, imagine that all occurrences of digit 3 in the first row appear within the first block. The algorithm detects this by traversing the first row and observing that the digit 3 appears as a candidate only in the first group of three cells as described above. (Recall that digit 3 must appear at least twice in a given group, otherwise a hidden single is found, checks for which were already performed.) Then, the algorithm traverses the second and third row of the same block to check if digit 3 appears in a different row in the same block. In that case, the omission technique can be used. To make the hint as intuitive as possible, all cells that contain the digit as a candidate in the given block are highlighted.

Unfortunately, finding cell(s) for which the technique can be applied requires multiple traversals of houses. Performance wise, this should not be an issue as the search space is quite small

3.2.7 X wing

X wing is another technique that involves removal of pencil marks. Contrary to previous techniques, it only requires pairs of rows or columns (not blocks).

First, we need to find two rows (or columns) that contain a digit exactly twice. Next, we need to check if the pairs of cells for each row (or column), exist in the same column (or row). Given two cells with index[2] i_1 and i_2 respectively, cells are in the same column if $(i_2 - i_1) \bmod 9 = 0$, Where mod is defined as the remainder of division. Furthermore, two cells are in the same row if $i_1 \text{ div } 9 = i_2 \text{ div } 9$ where div is defined as integer division. Once two pairs of such cells are found, the corresponding pair of columns (or rows) is traversed. If at least one additional cell with that candidate is found, then the X wing technique can be used. The digit under inspection can then be eliminated from the additional cells found.

For the purposes of providing a hint, the algorithm needs to return the digit for which an X wing was found, the two pairs of cells identified and the cells from which the pencil mark can be removed. The two types of cells must be highlighted using a different colour.

It is worth reiterating that the technique requires that the pencil mark appears exactly twice for the pair of rows or columns. Limiting the search space to pairs of cells that appear on the same columns or rows is tempting but incorrect.

[2]Recall that the index of a cell was defined to be row*9+column, and ranges from 0 to 80

3.2.8 Swordfish

The swordfish technique is a more general case of the X wing technique. The main difference between the two is that for the identification of a swordfish technique, pairs of three rows (or columns) are examined instead of two. Similarly, the algorithm focuses on a single digit at a time. First, the same digit must appear exactly twice in all three rows (or columns). Second, the pencil mark must appear in exactly three columns (or rows) across all three rows (or columns). In that case, if the digit appears as a pencil mark in any other cell in the same columns (or rows), that pencil mark can be removed. It is worth noting that this algorithm works on the assumption that no X wings exist in the puzzle.

The digit and 6 cells used for identification are then returned, along with the cells from which the pencil mark will be eliminated. Again, the two categories of cells are highlighted using a different colour.

3.2.9 XY wing

XY wing is the last technique that will be implemented for this project. The technique requires three cells that have exactly two candidates. Finding all cells that have exactly two candidates can be easily found with the setup described at the start of the section (Section 3.2.1). All cells that have more than two candidates are irrelevant to this technique. The algorithm for detection examines each cell that has exactly two candidates separately. The candidates of each such cell (pivot) are named X and Y. Then, the algorithm tries to find two other cells (pincers) that share a house and a pencil mark with this cell. In other words, the candidates of the pincers are (X, Z) and (Y, Z) respectively. If three such cells are found, then the XY wing technique can be used. The algorithm traverses all cells that share a house with both pincers and returns the subset of these cells that have Z as a candidate.

3.2.10 Multiplayer

3.2.10.1 Multiplayer modes

As previously mentioned, no implementation of multiplayer Sudoku was found online. As a result, the design of such functionality was kept as simple as possible. This section is mainly concerned with the concepts and challenges associated with the implementation of multiplayer Sudoku.

Both cooperative and competitive multiplayer modes were considered. However, as the project was limited to one academic year and the idea of multiplayer Sudoku is completely new, implementing both cooperative and competitive multiplayer modes was deemed too difficult. Implementation of cooperative multiplayer was preferred. The reasoning behind this decision can be broken down to two factors.

The most apparent problem with an implementation of a setting where two users compete to solve the same puzzle in real time is the user interface. As the purpose of the project is to develop an android app, showing two boards side by side while keeping the numbers readable was a tough ask. Such implementation would be much more

suitable for a web app or a desktop app. Furthermore, implementation of such a user interface could prove troublesome, due to the author's limited experience with android development.

The second reason a cooperative multiplayer was preferred was that the potential requirement of synchronization techniques. Such an implementation was deemed more interesting from an academic standpoint. Even though competitive multiplayer was not implemented, its design is discussed in section 3.4.1.

3.2.10.2 Multiplayer summary

Cooperative multiplayer, in its simplest form, can be summarised as follows: Two users look at the same board, one of them solves the puzzle while the other one provides assistance. Assistance can be given by highlighting specific cells to point out that a specific technique can be used. Furthermore, hints or explanations can be given through a chat box.

3.2.10.3 Choosing a database

For the purpose of synchronization, an online database was required. As the needs of the project are straightforward and simple, any database would suffice. Two different databases were considered: Faircom EDGE and Google's Firebase realtime database.

Faircom EDGE [3] is one of the most popular database solutions and is widely used in the industry. Owned by Faircom, initially released in 1972, it provides a relational DBMS across many platforms, including android. It is accompanied with ample documentation and provides support for multiple scenarios, with cutting edge technology when it comes to performance and security. Unfortunately, the service is not provided to students for free. Furthermore, the initial learning curve seemed quite steep, as most guides found online often assume prior knowledge with the software.

Firebase Firestore [4] is a cloud-based realtime document store. Created by Google in 2012, it is primarily aimed towards android developers, and is the official recommendation for android apps. A trial period for students to test small projects is also available. The author experimented with the Firestore SDK for a personal project in the past, something that is likely to reduce the time required to set up the database.

The comparison between the two databases leaves Firebase as a clear winner for the needs of this project.

3.3 Requirements

In this section, the requirements that each individual component of the app must fulfill to be considered successful are specified.

[3] https://www.faircom.com/products/faircomedge-iot-database
[4] https://firebase.google.com/docs/firestore

3.3.1 User interface

- **Intuitiveness:** The implemented interface must be easy to understand and must be usable without any prior training.
- **Responsiveness:** The interface must be responsive to the user's actions without any noticeable delay. Loading icons must be displayed when a delay is anticipated.
- **Robustness:** The app must not crash under normal circumstances.
- **Functionality:** The interface must be capable of displaying the current state of the puzzle, while responding to the user's actions. Users must be able to solve a puzzle, edit pencil marks, highlight cells and request a hint.
- **Speed:** The interface must take no longer than 0.5 seconds to update after a state change, such as editing of a cell.

3.3.2 Hints

- **Intuitiveness:** Hints must be displayed in an intuitive way that requires no prior knowledge.
- **Functionality:** Given a puzzle, the user must be able to request a hint. Hints should include as much information as possible. Hints should include an explanatory message while potentially highlighting the cells involved or the digit of interest.
- **Speed:** Providing a hint should take no longer than 2 seconds. More basic hints are expected to be delivered faster.

3.3.3 Multiplayer

- **Intuitiveness:** Multiplayer functionality should be easy to use.
- **Functionality:** Users must be able to connect using Firebase Firestore and cooperate to solve a puzzle, as described in section 3.2.10.2,
- **Responsiveness:** State changes must appear in real-time. Ideally, it should not take more than one second for the board to be updated after a state change.

3.4 Conceptual Design

Due to time constraints, only some of the proposed ideas were implemented. However, significant effort was put in designing the potential implementation of some of them. In this section, two such ideas are discussed.

3.4.1 Competitive multiplayer

Competitive Sudoku is an interesting subject that is unfortunately not found in literature or online. The only form of competitive Sudoku available is comparison of solving times for a specific puzzle. In this section, two possible implementations of real-time competitive Sudoku are discussed. The difference between the two lies in whether a player can see their opponent's board.

3.4.1.1 Blind multiplayer

The simplest possible enhancement of existing competitive Sudoku solutions would involve asking two users to start solving a puzzle simultaneously and aim for the fastest time. Without any additional modifications, this process only adds psychological pressure to the players.

To further improve this mode, players could receive indication of their opponent's progress. This could be done in many different ways, such as displaying percentage of empty cells filled or highlighting cells solved by the opponent using a different colour.

A clear advantage of this mode is that it requires less development time as the UI does not need to be significantly altered. Furthermore, it can be easily scaled to accommodate 3 or more players simultaneously.

3.4.1.2 Viewing opponent's progress

A different implementation of competitive multiplayer would be one that allows players to see their opponent's grid. As previously mentioned, this requires adjustments to the UI and would be much more suitable for a desktop or web app.

The simplest implementation would not force players to be competitive. On the contrary, it would allow players to speed up their progress by "copying" missing cells from their opponent's grid. In that case, the first player to complete a puzzle is not necessarily the best solver. One way of enforcing fairness would be to implement a scoring system. Coming up with a fair and balanced scoring system is far from trivial, something that at least partially explains the absence of something similar in the Sudoku community, despite the puzzle's popularity for decades. Multiple parameters need to be taken into account when designing such a scoring system. Some of these parameters overlap but can be important at different scenarios. The weight of each parameter can only be determined experimentally, something that requires observing multiple users of variable expertise solve puzzles of variable difficulty. These weights could also be adjustable by users to offer a tailored experience. Some factors that could be taken into account when deciding how to award or deduct points when a user fills a cell are listed below:

- **First player to solve a cell:** The purpose of this parameter is twofold; Discourage players from waiting for their opponent to solve a cell, and add additional pressure to solve as much of the puzzle as possible before their opponent. It is important to note that choosing a different "solving path" is very common. In that case, this parameter is not completely disregarded, as some paths allow

for quick solving of most ”easy” cells. Such behaviour might seem unwanted at first, but is something that could be factored in by experienced players when solving a puzzle.

- **Time between previous filled cell:** The main purpose of this parameter is to promote a systematic approach to solving a puzzle and push players to improve instead of staring at a puzzle waiting for their opponent to solve a cell.
- **Technique required to fill cell:** Solving a cell that requires a more advanced technique should award more points than solving an open single. The hint functionality implemented can be adapted to find the technique required to solve a cell.
- **Hints requested:** Competitive players could still ask for a hint when stuck. Some sort of penalty is desirable to prevent players from exploiting this feature.
- **Mistakes made:** Whenever a user fills in a cell, the value entered can be compared with the solution(s) found by the solving algorithm to determine whether the value filled is correct. Deducting points for a mistake would discourage guesswork, as almost every puzzle can be solved using appropriate solving techniques, without requiring any guessing. This factor is probably the most interesting. First of all, it is common to believe that guessing is part of the spirit of Sudoku puzzles, arguing that players that unnecessarily guess will be certainly penalised by the emerging need to backtrack. Second, a question that arises is whether the score deduction should be communicated to the player. Such an implementation could encourage guesswork at the later stages of the game, as players could overcompensate for any points penalty by guessing by solving the puzzle faster. Conversely, if players are not aware of their mistake, then deducting points for mistakes could be seen as a double penalty.
- **Cell correctly filled:** Similarly to deducting points for mistakes, points could be awarded when a cell is correctly filled. A similar dilemma also emerges in this case. More importantly, an indication whether the opponent filled a cell correctly or not will lead players to blindingly copying the correct value or eliminating a pencil mark.
- **Win bonus:** Winning the game should have some bonus associated with it. After all, this is the primary purpose of the game. From a gaming standpoint, it increases the excitement of the players and motivates them to stay focused at the latest stages, which are often trivial.

3.4.2 Extending solution for variants

In this section, ways in which the solver and hints functionality could be extended for Sudoku variants are discussed.

Extending the solver to accommodate larger puzzles is trivial with the current implementation. The same applies for the hint functionality, as the techniques discussed, and the algorithms implemented are not depending on the size of the puzzle.

In the case of puzzles with additional regions, the problem boils down to taking these regions into account when initialising or updating the candidates for each cell. The same can be said about the hint functionality. Not all techniques can be applied to all the regions. As such, Variant-specific techniques could be implemented but are probably unnecessary for most puzzles of this kind.

Adapting for variants that introduce additional symbols is not as straightforward, as the representation of the grid must be accompanied by the the positioning of these symbols. An implementation that favours simplicity, but might not be the most efficient, would be one that adjusts the candidates of all cells affected by such a symbol whenever a relevant cell is filled. The same could happen for cell candidates. Everything else remains unchanged.

With this implementation, the internal representation of these symbols poses the final hurdle. Thermometers used in thermo Sudoku are always forming a straight line. As such, storing a tuple with the cells at which the base and top of the thermometer appears is sufficient. Of course, this representation requires additional logic to determine the orientation at which the thermometer is placed. In the case of arrow Sudoku, all cells that an arrow passes by must be stored. Finally, Adapting for Odd/Even Sudoku is trivial as the corresponding candidates could be eliminated during the initialisation phase. No further adjustments would be required in that case.

Chapter 4

Implementation

4.1 Puzzle representation

The addition of pencil marks meant that the way that puzzles are stored had to be changed. Last year's implementation stored a string containing the contents of each row sequentially, using 0 to represent empty cells. As a result, every puzzle was represented by an 81-character string. This could be adapted to a string that contains 9 characters for each cell. Variants of this representation are commonly used among similar apps, such as Hodoku[1] With such a setup, A cell is determined to be filled when only one nonzero value is present in the 9 characters corresponding to the contents of the cell. This system is nearly perfect, but there is a caveat. Unfilled cells with only one candidate must be handled separately. Hodoku handles this edge case by assuming that the puzzle represented is correctly formed. When such a cell is encountered, all other cells that share a house with the cell in question are examined. If the number corresponding to that cell does not appear as a pencil mark in any other relevant cells, it is assumed to be filled.

Incorrect puzzle representations cannot be eliminated completely, especially when multiple people are editing the same puzzle. As such, a different representation was used. With the order of appearance remaining unchanged, the contents of each cell are separated with a comma. Furthermore, substrings corresponding to unfilled cells start with 0, leaving no ambiguity. The resulting string's length ranges from 161, for a fully filled puzzle, to 891, for an empty puzzle. A size that is still quite small and can be easily handled by any modern mobile device and internet connection. For easier development and testing, a method that parses a string corresponding to a puzzle was implemented. Such strings are available in most Sudoku websites, including Hodoku.

Board values and pencil marks are then separated internally, for more efficient computation of hints. Pencil marks appearing in each cell are stored in a set as described in section 3.2.1. A fixed-sized array is used for the cell values, with 0 representing an empty cell. When a cell is filled, its pencil marks are discarded.

[1] `http://hodoku.sourceforge.net/en/show_example.php?file=fh02&tech=Full+House`

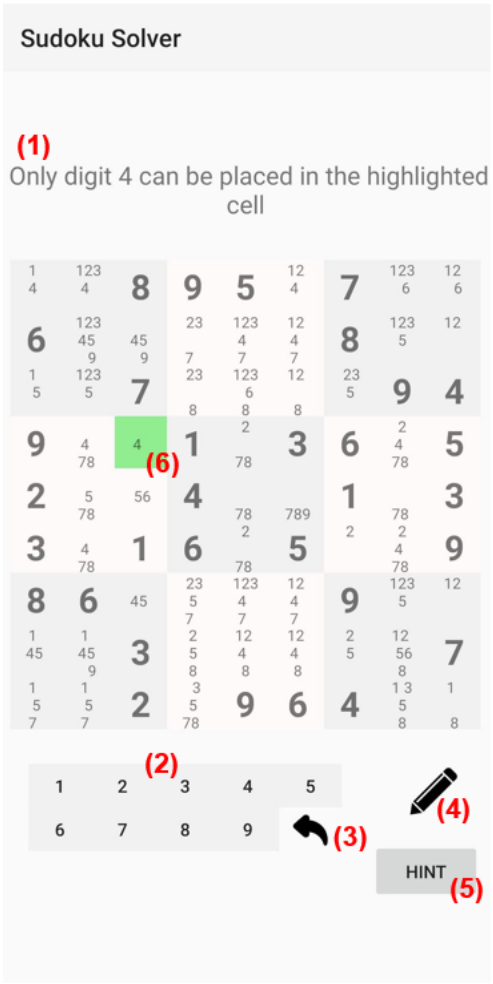


Figure 4.1: Interface of the app. (1) Text box for hints and errors. (2) Buttons used to select digit. (3) Undo button. (4) Toggle pencil marks. (5) Request a hint. (6) Cell highlighted for hint

4.2 User interface

Adjusting the grid for pencil marks turned out to be much harder than originally estimated. As previously mentioned, no plans for such adjustments were made during the first year of the project, and no easy solution was found online. The official recommendation for such tasks is the use of table layout [2]. Unfortunately, this layout is known to perform poorly, especially on older devices. Even the official documentation suggests the use of ConstraintLayout instead. As the responsiveness of the UI is a requirement, alternatives were considered.

Designing the individual components a board may consist of was attempted. The process proved time consuming and preliminary testing showed unpredictable behaviours across devices. Such behaviours could prove catastrophic in a project that involves user evaluation. As a result, a safer alternative was preferred. Specifically, each cell of the grid was divided into 3 ”rows”, in attempt to replicate the intended behaviour. This

[2] https://developer.android.com/guide/topics/ui/layout/grid

solution did not hinder the performance of the UI. Unfortunately, for some devices, pencil marks are not always aligned consistently, as shown in figure 4.2. Android provides a variety of options for typography [3], however, experimentation did not yield a solution.

An overview of the different components that are included in the solving screen is shown in figure 4.2. Additional screenshots showcasing the functionality of the app are available in appendix B. From last year's implementation, 9 numbered buttons were placed beneath the grid (number 2). Three additional buttons were added. The pencil button (number 4) is responsible for toggling between solving and editing mode. The hint button (number 5) is responsible for displaying a hint. A text box above the grid (number 1) is used to display the details of the hint, such as the type of the hint and the digit of interest. A text box was preferred over a popup to offer an uninterrupted solving experience and ensure that users can see the entire board along with the hint. The box is also used if a cell is filled incorrectly (figure B.6). For the implementation of this feature, the puzzle is compared to the solution(s) of the puzzle. An undo button (number 3) was also implemented. The button appears only when the initial puzzle was edited at least once. Every time the puzzle is edited, the previous state is saved in a stack. Even though size of each entry of the stack is negligible for modern devices, it was limited to 50. The undo functionality is demonstrated in figure B.4.

The functionality of the number buttons was enhanced, such that all cells that contain the corresponding number as a pencil mark are highlighted. This feature allows for easier detection of cells where solving techniques can be used.

For a cell to be filled, the desired number is clicked, followed by the desired cell. Then, the pencil marks for all unfilled cells that share a house with the edited cell are adjusted accordingly. The sequence required for the filling of a cell is demonstrated in figure B.2. Similarly, to add or remove a pencil mark, the pencil button is clicked, followed by the desired number button and then the desired cell. One such example is showed in figure B.3.

The initial state of the puzzle is saved in a dedicated variable to ensure that givens cannot be altered. When the user attempts to edit a given, an error message is displayed (figure B.5).

4.3 Hints

To determine the type of hint that needs to be provided, the algorithms described in section 3.2 are run sequentially until a hint is found. Additional care was taken to ensure that only the necessary computation is performed. For example, traversing a house stops as soon as the required condition is guaranteed to be false. As a result, hints are displayed instantaneously, with the most advanced instances requiring no more than 2 seconds.

[3] https://medium.com/google-design/the-android-developers-guide-to-better-typography-97e11bb0e261

4.4 Multiplayer

As described in section 3.2.10.2, the purpose of the multiplayer mode is to allow a spectator to join another player solving a puzzle and view their progress, highlight cells, or send messages through chat. This setup is convenient as it means that the contents of the board are controlled entirely by the player, eliminating the need for sophisticated synchronisation techniques.

4.4.1 Spectator UI

The UI for the first player is identical for the multiplayer mode. This is desirable as no additional time is required for users to be familiarised with the mode.

The UI was adapted for the spectating user. In an attempt to minimise the learning curve associated with spectating, no buttons were placed in the spectator's UI. Instead, they can highlight a cell by tapping on it and send a message to the player by typing it in a text box placed below the grid. The resulting UI for the spectating player is shown in figure B.7. Additional care was taken to ensure that the board is fully visible while the spectator is typing a message. This feature is extremely important in a real-time application as the state of the puzzle might change significantly while the spectator is typing a message. Furthermore, it allows the spectator to highlight cells while typing, resulting in a friendlier UI.

A simple starting screen that allows a user to create a new multiplayer "room" or spectate another player was also created. When a room is created, the user is asked to enter a password. A potential spectator can then use the password to join the room.

4.4.2 Connecting the app to firebase

Connecting the app to firebase is usually a trivial task, as detailed documentation is provided. Unfortunately, this was not the case as firebase requires use of the latest libraries (androidX[4] instead of the support library). Last year's implementation experimented with multiple deprecated libraries for the vision part of the project. After updating the libraries accordingly, the app was connected to a firebase project.

4.4.3 Data exchange

The data sent between the two players was kept to a minimum. Specifically, the current state of the board is sent to the database, every time a change is made, overwriting the previous. For this to happen, the board is converted to a string, as outlined in section 4.1. The spectator's app polls the database every one second. This ensures consistency between the two users. When a spectator highlights a cell, its id is added in a database array, which is then polled by the user's app every one second. A similar process is followed for the chat messages, which appear in bubbles at the bottom of the player's screen. An example of the multiplayer mode in use is demonstrated in figures B.8 and B.9

[4] `https://developer.android.com/jetpack/androidx`

Chapter 5

Evaluation

Evaluation is a core part of every successful app that is designed for a general audience. The most common evaluation methods for apps are user evaluation and Jakob Nielsen's 10 heuristics [37, 42, 40, 41]. Both methods could be used to provide quantitative evaluation metrics in a small period of time. As the app is relatively simple with only one screen, use of Nielsen's heuristics would not provide much benefit. Hence, user evaluation was preferred.

Due to the COVID-19 pandemic, in-person evaluation was not possible. To ensure that the app could be evaluated by as many people as possible, evaluation was limited to the parts developed this year. The vision part developed last year would be hard to evaluate as it would require participants to have a printed Sudoku. Evaluation of the multiplayer mode was also harder to evaluate as it required evaluation between two participants. Alternatively, evaluation could happen with the author taking the role of one of the two required users, simplifying the process. Organising meetings for evaluation proved impractical and time consuming. As a result, no significant user evaluation was performed for that part of the implementation.

5.1 User evaluation

The evaluation performed was certified according to the Informatics Research Ethics Process, RT number **2019/45209**. The full participant information sheet associated with this evaluation is available in appendix C. No personal information or information that could lead to the identification of participants was stored as part of this study. As no in-person evaluation was possible, the next best option that was the evaluation through a questionnaire.

5.1.1 Preparation of the app for evaluation

Before the app was published for user evaluation, minor changes were made to make the app as user friendly as possible. First, any requests for permissions were removed to eliminate any fears of malicious behaviour. Unfortunately, this means that the current state of the puzzle is no longer saved to the phone's internal storage. As a result,

the user's progress is reset when the app is closed. Second, a welcome message that is displayed the first time that the app is opened was added. The message is meant to introduce the user to the rules of Sudoku and provide information about using the app. The welcome message is shown in figure B.1.

Third, the app was limited to a single medium puzzle. Allowing users to enter their own puzzle was desirable but impractical, as the app requires no Sudoku knowledge. Giving a selection of puzzles of variable difficulty was also considered but rejected as such a choice could discourage people new to Sudoku from evaluating the app.

Fourth, use of pencil marks was enforced to make the hint functionality more intuitive, as most of the hints do not make sense without pencil marks. Furthermore, pencil marks are updated automatically when a cell is filled, leading to a smoother experience.

Finally, the app was tested extensively across multiple devices and android versions, revealing no bugs.

5.1.2 Publishing the app

Ideally, the app would be uploaded to the play store[1], allowing users to conveniently download the app on their device. Uploading the app to the play store could result in unpredictable delays, mainly due to the author's inexperience with the play store and the variety of libraries used. To avoid such delays and allow as much time as possible for evaluation, the app was bundled in an APK (Android Package), which would be used for evaluation.

5.1.3 Evaluation platform

Choosing the correct platform to host the questionnaire was not as trivial as expected. Use of the university's online survey tool[2] was suggested. Use of the tool by students is limited to postgraduate students and was therefore eliminated.

Use of Microsoft forms was considered next. Unfortunately, the formatting and customisation options available were very limited. This is the case as most options are exclusive to Microsoft forms pro, which is now re-branded to Microsoft Dynamics 365 Customer Voice[3]. As a results, alternatives were considered.

The most popular alternatives used for surveys are SurveyMonkey[4], Google Forms, and Qualtrics [5]. GDPR legislation required that the data gathered for the purposes of this study are handled and stored in the European Union. This was not the case for the first option and was ambiguous for the other two options. Data entered in Microsoft forms are stored in servers within the EU. Use of Microsoft forms was also considered safer as it was used during the Human Computer Interaction course in the

[1] https://en.wikipedia.org/wiki/Google_Play

[2] https://www.ed.ac.uk/information-services/learning-technology/survey-tools/online-surveys/introduction

[3] https://dynamics.microsoft.com/en-gb/customer-voice-transition/

[4] https://www.surveymonkey.co.uk/

[5] https://www.qualtrics.com/uk/

same academic year. To avoid any legal issues, Microsoft forms was used to host the developed questionnaire, despite the limited text formatting options.

Fortunately, Microsoft forms provided two very interesting and important features. First, forms created are guaranteed to be mobile friendly. This was of particular importance for the purposes of this project, as it allowed users to complete the questionnaire on their phone, directly after interacting with the app.

Second, questions can be made completely optional with the use of branching. Specifically, the answer to a question can influence the subsequent questions that are displayed. As a result, the questionnaire experience can be tailored to ensure that the maximum amount of useful information is gathered from each participant.

Another feature worth mentioning is the built-in immersive reader, which makes the survey accessible to people with disabilities.

5.1.4 Questionnaire design

Designing User Experience Questionnaires is a popular field of study with a wide range of literature. For the purposes of this project, the UEQ[6][26, 52, 53] handbook was consulted. Use of a 7-point scale was advised closed-type questions to reduce the effect of central tendency bias. As our questionnaire is quite simple, such a scale was deemed overly complex. The standard 5-point scale was used instead. Each question should be concerned with the evaluation of one aspect of the app.

The questionnaire was broken into 7-10 quasi-qualitative compulsory questions. Furthermore, 3 open ended questions were used to allow participants to include additional information if they want to, without the process becoming onerous. One such questionnaire should take less than 5 minutes to complete.

The resulting questionnaire was distributed to University of Edinburgh students. It is worth noting that the only requirement set was access to an android phone. As a result, participants are expected to have variable knowledge regarding Sudoku puzzles and experience with similar apps. The questionnaire is available in appendix D.

The questionnaire and app were both versioned to ensure that potential changes were documented. Fortunately, no changes were required. The survey was segmented into four sections. The title and version of the survey along with the author's contact information were placed at the top of each section.

The first section (figure D.1 contains the Participant information sheet, as shown in appendix C. This approach ensures that every participant at least scrolled past the information sheet before answering any questions.

The second section (figure D.2) is concerned with the installation of the app. Specifically, instructions to download, install and uninstall the app are given. If the participant states that they could not download and install the app, an additional question prompting them to provide additional details and optionally contact the author appears. No additional questions are given to users that fail to install the app. If the participant states

[6] https://www.ueq-online.org/

that they managed to successfully install the app, even with some troubleshooting, then the survey proceeds to the third section.

The third section (figures D.3, D.4 ,D.5) aims to evaluate the app with close-ended questions. The first two questions are compulsory and aim to evaluate the enjoyment and ease of use of the app. The third question asks the user if they have used the hint functionality. Answering positively results in an additional question regarding the user's satisfaction with the hint functionality. Next participants are asked whether they have used another Sudoku solving app. A positive answer results in two additional questions regarding the frequency of use of that app and a comparison between our app and the other app. Comparing the app with available solutions is extremely important for evaluation purposes and identifying weak spots of the app.

The fourth and final section (figure D.6) involves three open-ended questions. Specifically, users are asked to report any bugs, suggestions, and additional comments.

5.1.5 Evaluation results

In this section, the results of the evaluation are analysed. The distribution of the answers for the quasi-quantitative questions is available in appendix E. Despite the simplicity of the questionnaire, various insights can be gained by examining the results.

The survey received 41 responses, 71% of which are from people that managed to download and install the app successfully. We can conclude that the instructions given were clear and effective. The actual number of people that have completed the survey might be slightly lower as some people might have submitted a negative response before troubleshooting or contacting the author for help. Regardless, the questionnaire received enough responses for the results to be considered significant.

The questionnaire took an average of 6 minutes and 33 seconds to complete, which is slightly longer than expected but can be justified when the time required to read through the PIS is considered. Another possible explanation could be that participants that had not used the hint functionality tried to test it while completing the questionnaire.

5.1.5.1 Installation problems

Some users that were unable to install the app cited the fact that the app is unavailable on the play store as the reason. This was unfortunate but expected and understandable. Some other users cited fears about malware and data theft. As previously mentioned, additional care was taken to eliminate unwanted behaviours, and ensure that the app does not have unnecessary privileges. Some other users had trouble following the guide to enable installation of third-party apps. Any such problems were easily resolved in the cases that the author was contacted.

Finally, one participant claimed that they were unable to install the app as it was not compatible with their operating system version. The app requires android SDK 24, which is equivalent to android 7.0, released in 2016. Even actively maintained apps do

not provide support for this version, as security patches are no longer provided[7]. At the time of writing, android studio states that such a minimum requirement supports 73.7% of all devices. Even so, omitting this requirement from the questionnaire was a mistake which fortunately did not affect many of the participants.

5.1.5.2 Reported bugs

Even with extensive testing performed on a variety of devices and operating systems, discovery of some bugs is inevitable the first time any app is released to a wider audience. Excluding cases where intended behaviour was reported as a bug, (which will be addressed shortly) two bugs were identified. Both bugs were related to the UI and the settings of the specific users.

For the first bug, the participant reported an usual colour scheme. Fortunately, the affected user kindly emailed the author with a screenshot of the problem. Indeed, the colour scheme displayed was significantly darker than the implemented one. Research and experimentation revealed that this behaviour was caused by an experimental feature, which forces a ”dark mode” theme to all layouts. The problem was fixed by prompting the user to exclude the app from this feature. Unfortunately, ensuring that the app is always excluded from such changes is not possible. One way to mitigate the problem would be to use colours with higher contrast, making the interface friendlier when such features are enabled. However, caution must be taken to ensure that the appearance of the app under normal circumstances is not worsened. It is worth noting that the feature mentioned is different from the commonly used ”dark mode”, for which the compatibility of the app was thoroughly tested prior to its public release.

For the second bug, unusually large digits were reported. This was most likely due to the user's font settings. The app was designed according to the official implementation guidelines, which aim to respect the user's settings. Admittedly, different font sizes were not tested before the app was released. To fix this issue, the units used to determine the text size were changed to ignore this setting. As stated in the official documentation[8]: *The sp unit is the same size as dp, by default, but it resizes based on the user's preferred text size.* This is not ideal as it is a bad practice. However, the UI was cautiously designed to be easy to read.

Both reported bugs highlight the importance of user evaluation and the importance accessibility in software development. The latter is, unfortunately, often grossly overlooked in the industry.

5.1.5.3 User satisfaction

The app received mostly positive feedback, especially regarding its ease of use, for which 77% of participants were satisfied. Some aspects of the app received mixed feedback. For example, the lack of options was praised by most users as it allowed them to focus on the task at hand. Some other users criticised this as they believe the

[7] https://en.wikipedia.org/wiki/Android_version_history

[8] https://developer.android.com/training/multiscreen/screendensities.html#TaskUseDP

app is too restrictive. This problem can be easily mitigated by introducing an options menu. In that case, additional care must be taken to ensure that use of the app remains simple. It is worth noting that the automatic editing of pencil marks was praised by most users, as it kept their interest piqued. Furthermore, it reduced the solving time, allowing them more time to fill in the questionnaire. Another aspect of the app that was frequently praised is the absence of advertisements. These two factors can be used to justify the preference of 47% of participants that have used a similar app in the past.

The biggest criticism observed was related to the lack of variety in puzzles, something mentioned by multiple users. As previously mentioned, this was a conscious decision rather than an oversight, as the volume of responses was prioritised. Adding additional puzzles of varying difficulty is trivial with the existing implementation.

Another issue that was caused by the medium difficulty of the puzzle is that only 59% of the participants used the hint functionality. A harder puzzle could potentially motivate more users to request a hint. Most of the users that requested a hint were satisfied. Two participants complained about the hint functionality, stating that they would expect the hint to solve a cell or remove a pencil mark, as done by some other apps. This criticism was unexpected as hints were primarily designed as a learning method. To accommodate for these users, an option could be added that removes a pencil mark from the least constrained cell or solves singles when the user requests a hint.

The most common suggestion, other than the inclusion of more puzzles, was the inclusion of a tutorial section, explaining the rules of Sudoku, the various types of hints, and the logic behind them. Inclusion of a timer was also suggested. This was overlooked during research and implementation, as it is common in most similar apps. Minor improvements for the UI were also proposed. Finally, some users mentioned the lack of acknowledgment when the puzzle was completed. In hindsight, a popup congratulating users and redirecting them to the questionnaire could have been implemented.

5.2 Requirements evaluation

Overall, the app meets most of the requirements set in section 3.3. Specifically, the user interface was proved to be intuitive, responsive and robust through user evaluation. The only minor issue is related to the position of the pencil marks, which is not always consistent.

The hint functionality works as expected, while being intuitive and responsive.

The simple implementation of multiplayer Sudoku that was implemented is believed to be functional and intuitive. The responsiveness of the mode was inconsistent during testing, taking up to 2.5 seconds for some state updates. This can be partially attributed to the testing environment, as an emulator was used as a second device. Any induced delays would be unnoticeable for an offline app but are crucial for an app that responds in real time. Using a premium instance of the firebase database could also improve the responsiveness of the system. Additional evaluation is required to accurately assess the performance of this part of the implementation.

Chapter 6

Conclusions

6.1 Project achievements

The achievements of the project in the second year can be summarised as follows:

- Summary of previous work and extensive literature review covering human solving techniques, Sudoku variants and analysis and comparison of similar apps across various platforms.
- Design and implementation of efficient algorithms used to generate and display solving hints based on the current state of the puzzle.
- Adaptation of user interface and internal representation used from last year's project to handle pencil marks.
- Design and implementation of an intuitive app for Sudoku solving, including highlighting, undo functionality, detection of errors and display of detailed hints.
- Design of a questionnaire which was used for the evaluation of the implemented app. The results of the questionnaire were thoroughly and critically evaluated.
- Design of two novel real-time multiplayer Sudoku modes; competitive and cooperative.
- Implementation of cooperative Sudoku using Google's Firebase Firestore.

6.2 Project limitations

When the app is compared to the apps analysed in section 2.5.3, some limitations are immediately made obvious. Namely, the UI of these apps looks much more professional while they offer a plethora of customization options and tutorials. This was expected as these apps were developed by professional software companies, over many years. Naturally, the implementation produced by a single student over the course of an academic year could not directly compete with such apps. Regardless, some evaluation participants expressed a preference for the developed app.

Another important limitation of the developed app lies in the variety of puzzles available. An issue that can be easily mitigated at first, as inclusion of additional puzzles is trivial. However, the app would benefit greatly with the addition of Sudoku variants, such as those described in section 2.4. Such addition would be significantly harder, as it requires changes to the UI and potentially the hint functionality and solver.

6.3 Possible extensions

Despite two years of continuous implementation, the project can be extended in multiple ways, including:

- Integration of features developed during the first year of the project, after which users should be able to scan a puzzle using the phone's camera and continue solving it on the app.
- Additional implementation to accommodate Sudoku variants, as described in section 3.4.2
- Extension of the developed multiplayer mode, adding more options and potentially implementing synchronization techniques. Generalisation to more than two users is also possible.
- Implementation of a competitive multiplayer mode, as discussed in section 3.4.1
- Refinement of the user interface, aiming for a more professional appearance and more customisation options.
- Additional evaluation, especially regarding the puzzle recognition feature and the multiplayer mode.

6.4 General remarks

In this report, the implementation of an android app for solving Sudoku with pencil marks and hints was discussed. Furthermore, the concepts of cooperative and competitive multiplayer Sudoku were analysed thoroughly, and a prototype of the former was implemented. The single player features of the app would benefit greatly from customization options and aesthetic improvements. The multiplayer concepts discussed showed great potential towards innovation in Sudoku puzzles. Concepts that were, unfortunately, overlooked prior to this project. All the goals of the project were achieved to a satisfactory degree. Therefore, the project is considered successful.

Appendix A

Sudoku variants

This section is used to demonstrate examples of Sudoku variants discussed in section 2.4. The examples used were found online.

A.1 Size variations

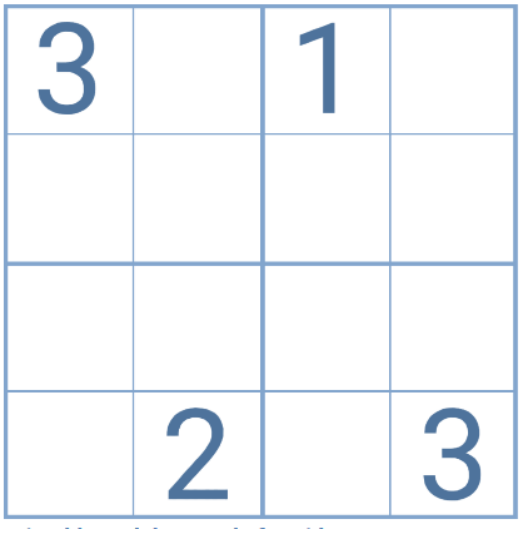


Figure A.1: 4x4 Sudoku. Mainly intended for kids. Source: `sudokuonline.io`

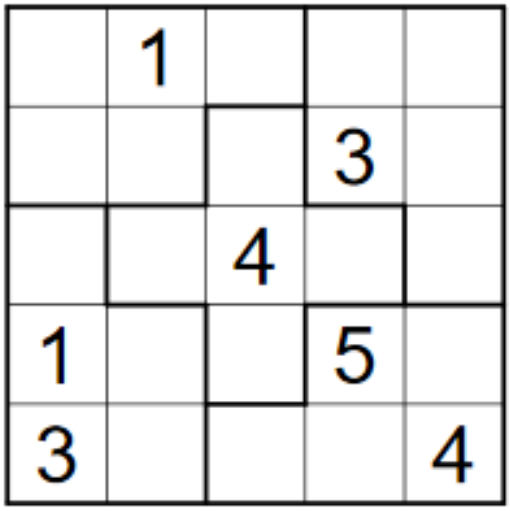


Figure A.2: 5x5 Sudoku. The smallest available jigsaw/irregular Sudoku, forming 5 different regions. Source:`sudoku-download.net`

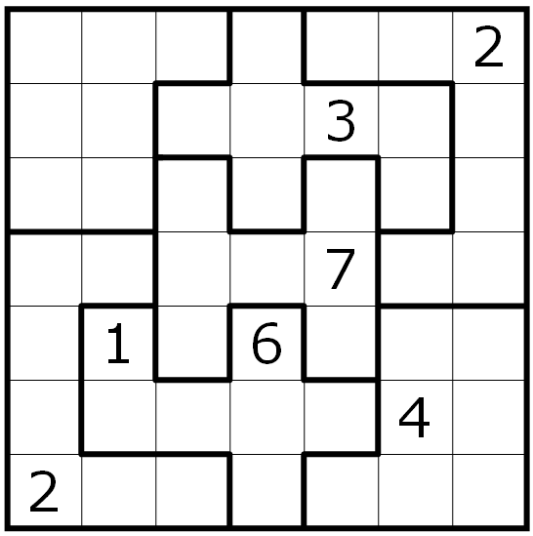
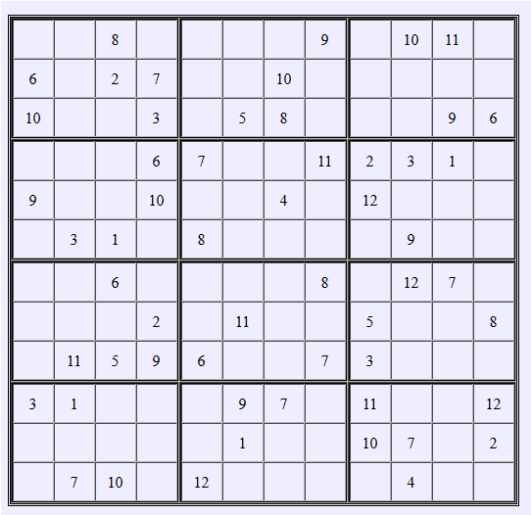
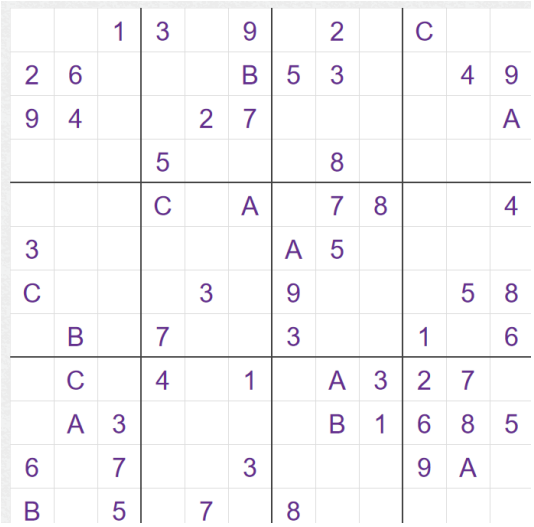


Figure A.3: 7x7 Sudoku. Jigsaw/irregular Sudoku, forming 7 different regions. Source: `logicmasters.de`



		8					9		10	11	
6		2	7			10					
10			3		5	8				9	6
			6	7			11	2	3	1	
9			10			4		12			
	3	1		8					9		
		6					8		12	7	
			2		11			5			8
	11	5	9	6			7	3			
3	1				9	7		11			12
					1			10	7		2
	7	10		12					4		

Figure A.4: 12x12 Sudoku, also called Sudozen. Similar to traditional Sudoku, blocks are three rows long and four columns wide. Filled with numbers 1-12. Source: `http://12x12sudoku.com/`

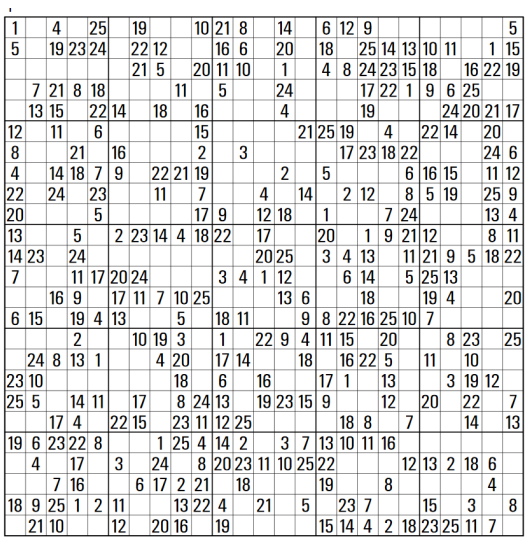


		1	3		9		2		C		
2	6				B	5	3			4	9
9	4			2	7						A
			5				8				
			C		A		7	8			4
3						A	5				
C				3		9				5	8
	B		7			3			1		6
	C		4		1		A	3	2	7	
	A	3					B	1	6	8	5
6		7			3				9	A	
B		5		7		8					

Figure A.5: 12x12 Sudoku alternative, filled with characters 1-C. Source: `https://puzzlemadness.co.uk/12by12giantsudoku`

	B		E							D	C	6			G
		C		4	F	E	3	G	8		B	7			A
	8												E		D
3	A	5						1	9					B	F
	5		G		8	1	4					3			C
8		A	6		E		B	9		3					
9						7			D	C			1		
7		E		2	A		C		B	4	F				
	7					B		E		1	G	5	A	9	3
	1			6	D	2			7		3		C		
		3	B	E		G	F				4	1	D	7	
A	E	8	F	1		3	9	5		6	D				2
	D		A		9		E			B		G	3		1
E	2		3	F					6		8				
			8	7			G	F			A		B	2	5
				D			1	3	5			4			8

Figure A.6: 16x16 Sudoku, also known as hexadoku. Similar to the traditional Sudoku, filled with characters 1-F. Source: https://puzzlemadness.co.uk/16by16giantsudoku/



1		4		25		19			10	21	8		14		6	12	9							5
5		19	23	24		22	12			16	6		20		18		25	14	13	10	11		1	15
						21	5		20	11	10		1		4	8	24	23	15	18		16	22	19
	7	21	8	18				11		5			24				17	22	1	9	6	25		
	13	15		22	14		18		16				4				19				24	20	21	17
12		11		6					15					21	25	19		4		22	14		20	
8			21		16				2		3					17	23	18	22				24	6
4		14	18	7	9		22	21	19				2		5				6	16	15		11	12
22		24		23			11		7			4		14		2	12		8	5	19		25	9
20				5					17	9		12	18		1			7	24				13	4
13			5		2	23	14	4	18	22		17			20		1	9	21	12			8	11
14	23		24									20	25		3	4	13		11	21	9	5	18	22
7			11	17	20	24				3	4	1	12			6	14		5	25	13			
		16	9		17	11	7	10	25				13	6			18			19	4			20
6	15		19	4	13			5		18	11			9	8	22	16	25	10	7				
			2			10	19	3		1		22	9	4	11	15		20			8	23		25
	24	8	13	1			4	20		17	14			18		16	22	5		11		10		
23	10							18		6		16			17	1		13			3	19	12	
25	5		14	11		17		8	24	13		19	23	15	9			12		20		22		7
		17	4		22	15		23	11	12	25					18	8		7			14		13
19	6	23	22	8			1	25	4	14	2		3	7	13	10	11	16						
	4		17		3		24		8	20	23	11	10	25	22				12	13	2	18	6	
		7	16			6	17	2	21		18				19			8					4	
18	9	25	1	2	11			13	22	4		21		5		23	7			15		3		8
	21	10			12		20	16		19					15	14	4	2	18	23	25	11	7	

Figure A.7: 25x25 Sudoku, Similar to traditional Sudoku but with blocks of size 5x5. Filled with numbers 1-25. Source: `Sudoku-download.net`

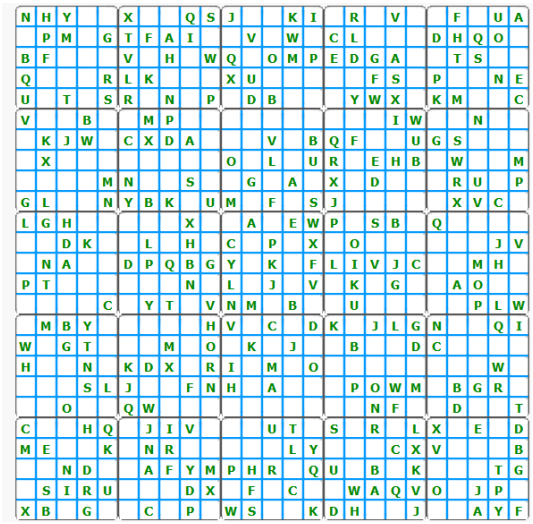


Figure A.8: 25x25 Sudoku known as alphadoku. Filled with Characters A-Y. Source: `https://www.sudoku-puzzles-online.com/`

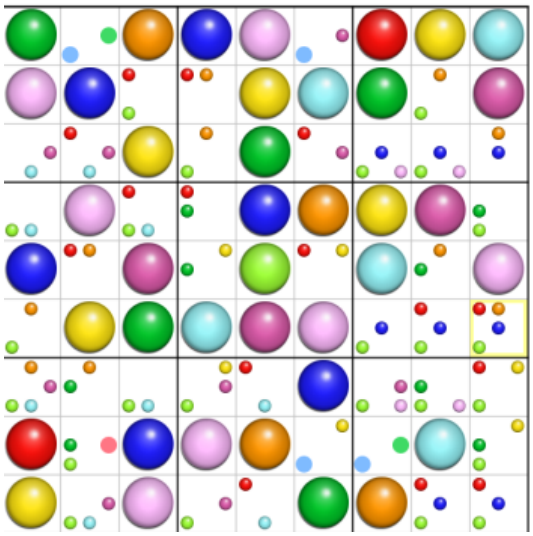
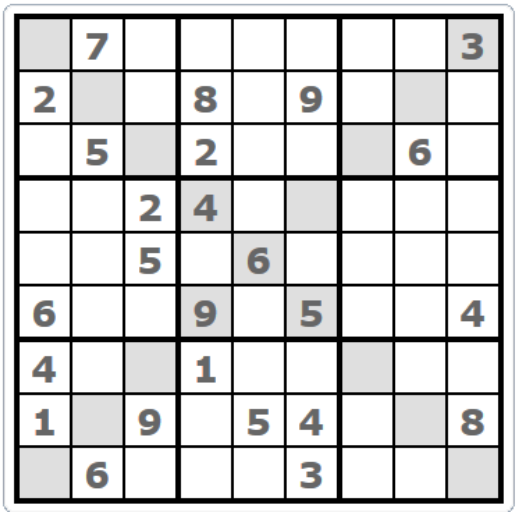


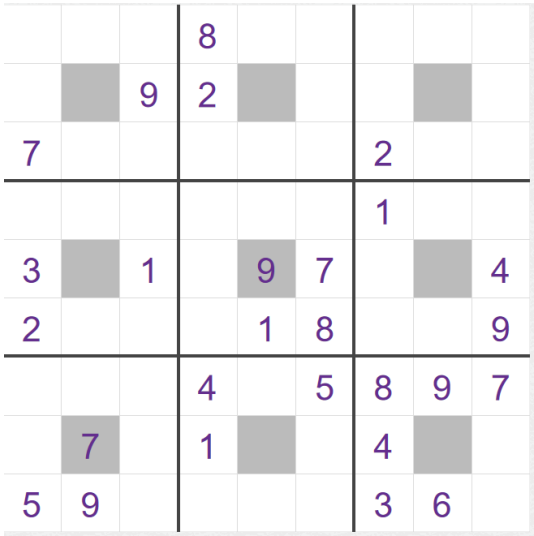
Figure A.9: Colorku. Regular Sudoku filled with coloured circles instead of numbers. Source: Hodoku

A.2 Rules variants



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

Figure A.10: X sudoku. Cells on the diagonals (shadowed) form two additional regions. Source:`http://www.sudoku-space.com/x-sudoku/`



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

Figure A.11: Center dot Sudoku. Cells on the middle of every block (shadowed) form one additional region. Source:`https://puzzlemadness.co.uk`

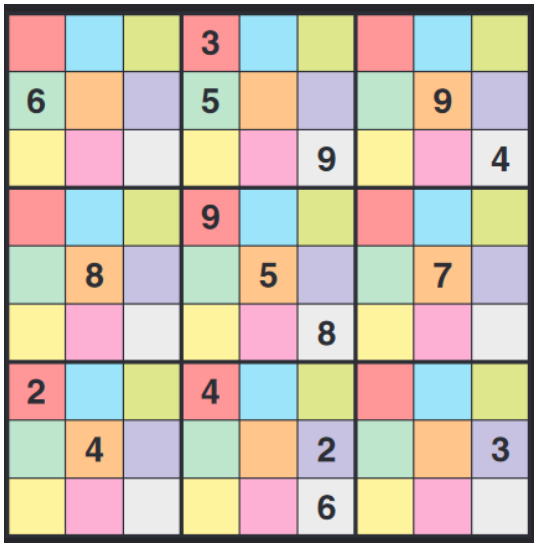


Figure A.12: Colour sudoku using 9 different colours to create 9 additional regions. Source:`https://puzzlephil.com/`

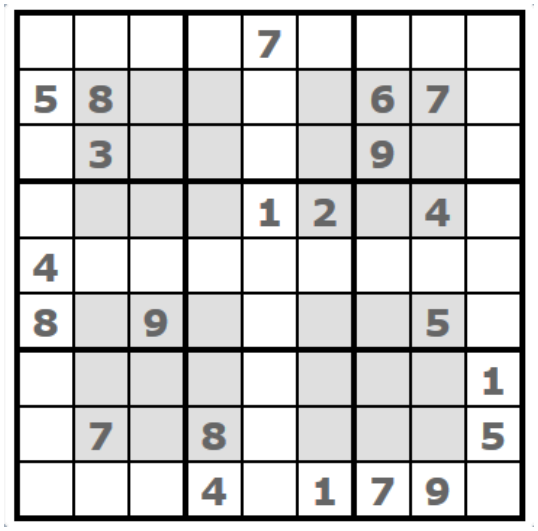


Figure A.13: Hyper Sudoku. Shadowed cells form four additional regions. Source:`http://www.sudoku-space.com/hyper-sudoku/`

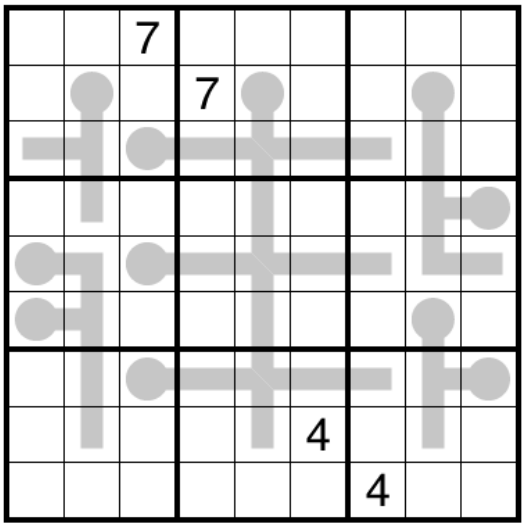


Figure A.14: Thermo Sudoku. Numbers in cells from the base of the thermometers (shadowed) can only be increasing towards the top Source:`https://www.gmpuzzles.com//`

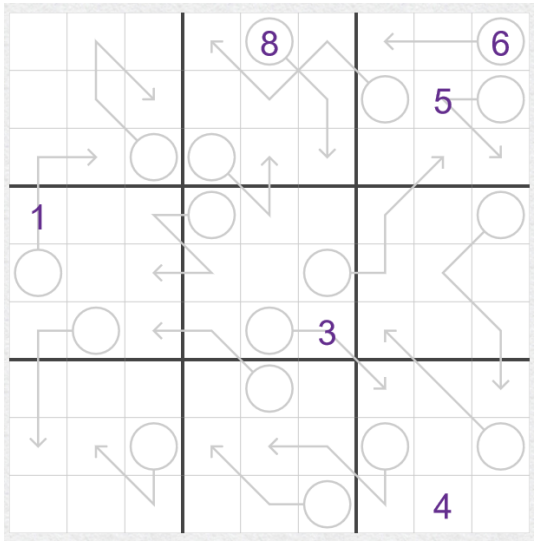


Figure A.15: Arrow Sudoku. The cell in every circle must be equal to the sum of the numbers in the cells the corresponding arrow passes through. Source:`https://sudokucentral.co.uk/`

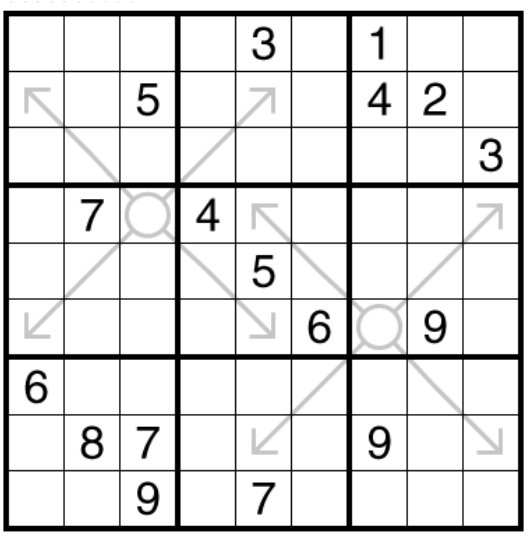


Figure A.16: Arrow Sudoku. The cell in every circle must be equal to the product of the numbers in the cells the corresponding arrow passes through. Source:`https://www.gmpuzzles.com`

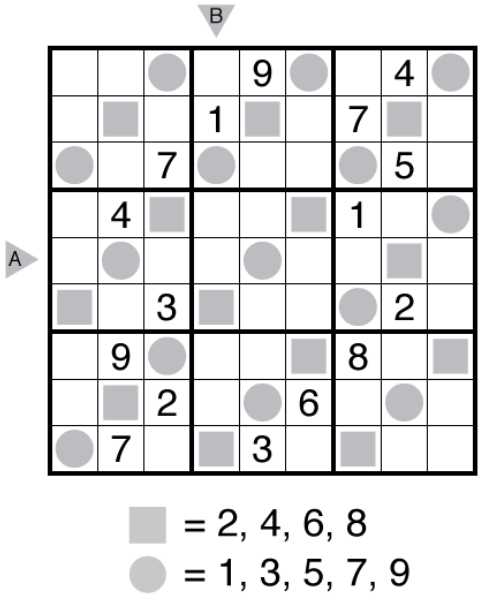


Figure A.17: Odd-even Sudoku. Squares must be filled with odd numbers and circles must be filled with odd numbers. Source:`https://www.gmpuzzles.com/`

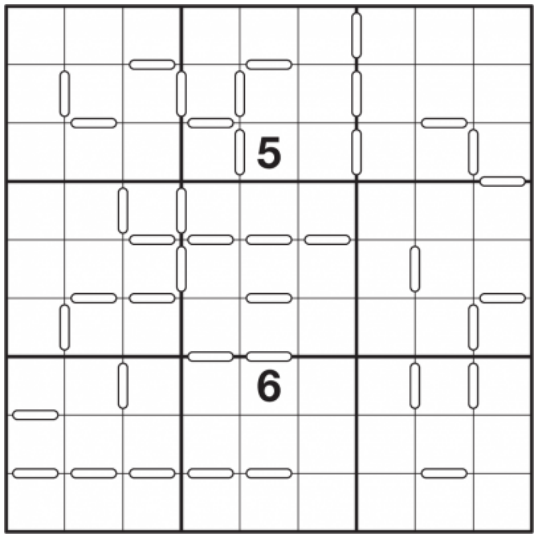


Figure A.18: Consecutive Sudoku. When an edge between two cells is marked, the number placed in those two cells must be consecutive. Source:`http://www.anypuzzle.com/`

Appendix B

Implementation screenshots

This appendix contains sequences of screenshots that demonstrate the functionality available in the app.

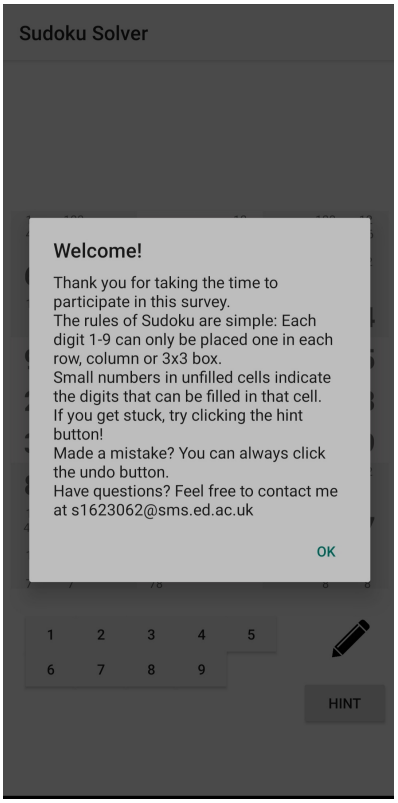


Figure B.1: Welcome message shown the first time the app is opened.

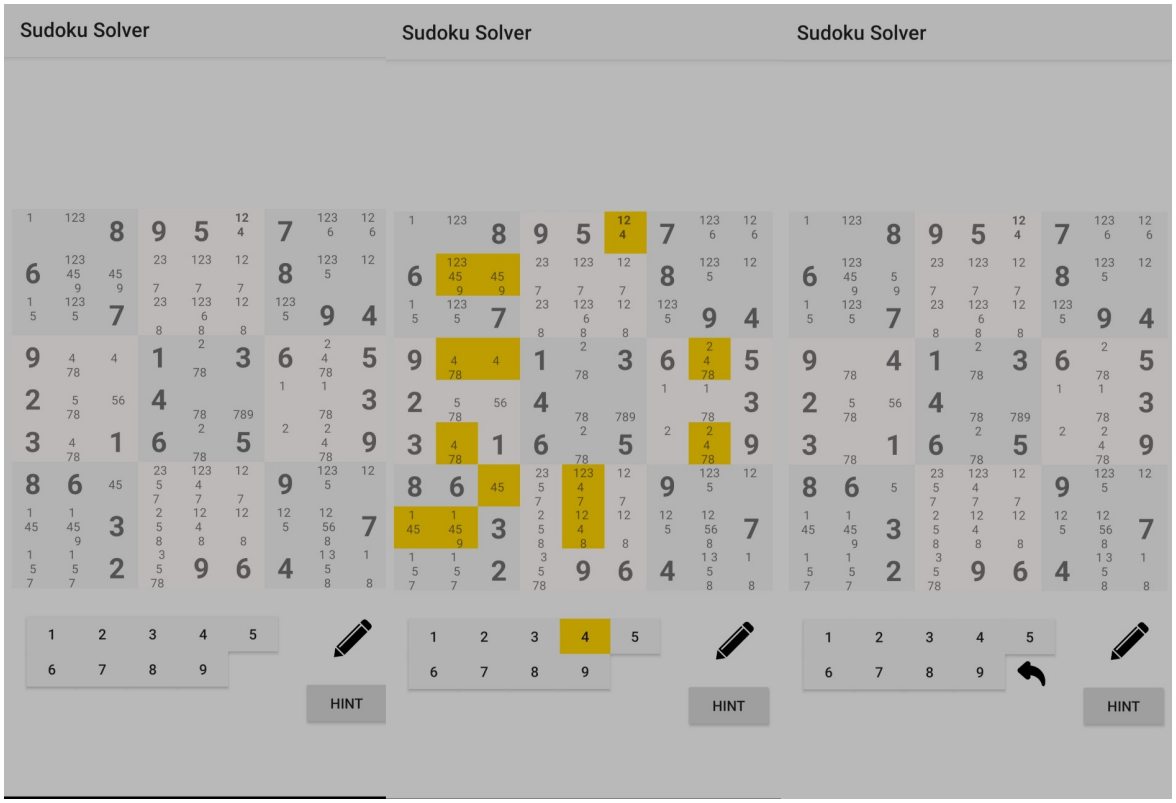


Figure B.2: Steps required to fill the cell in row 4, column 3. The user selects the number from the buttons below the grid (middle) and then clicks on the cell.(Right). The undo button appears once a change is made to the board.

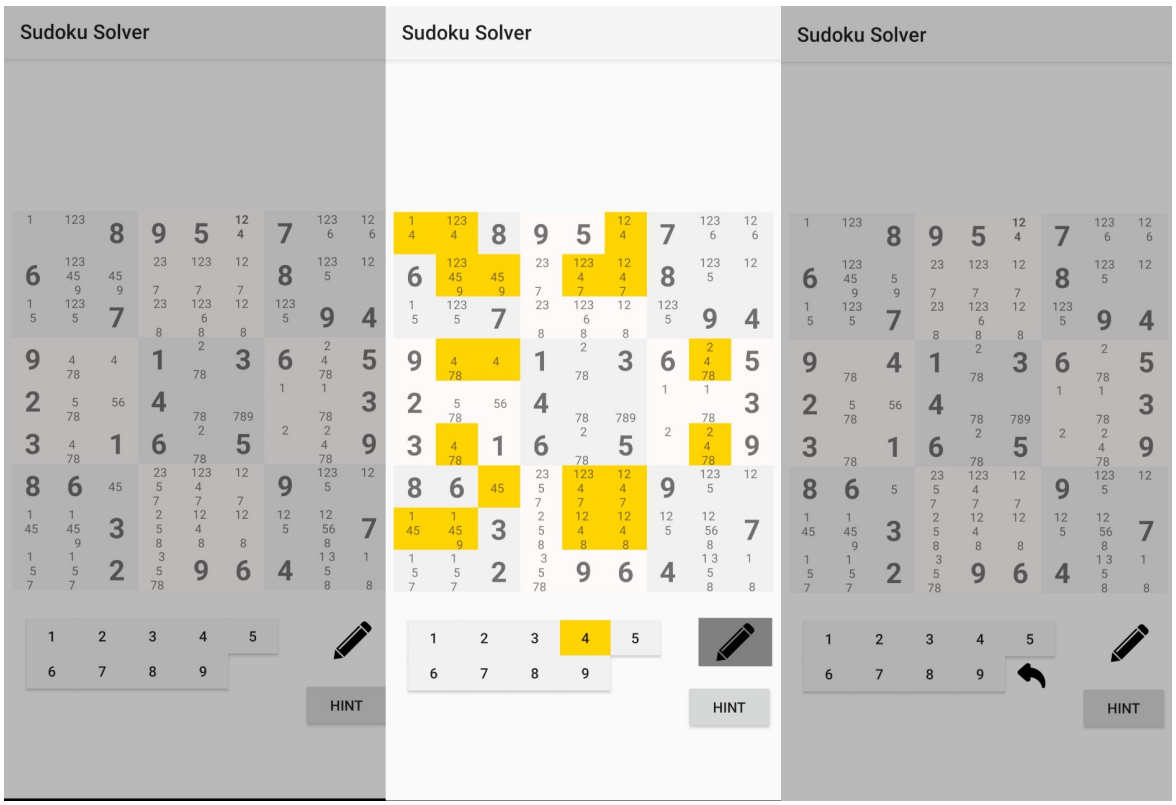


Figure B.3: Steps required to fill remove pencil mark 4 from row 2 column 3. The user selects the number from the buttons below the grid (middle) and then clicks on the cell.(Right).

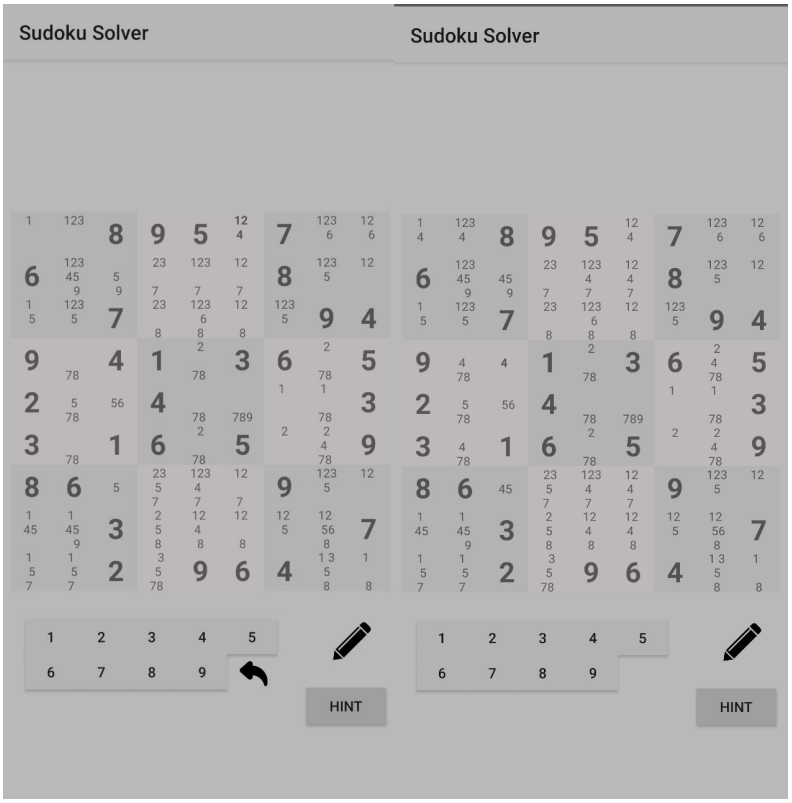


Figure B.4: User clicks undo, removing the filled value from the cell in row 4 column 3.

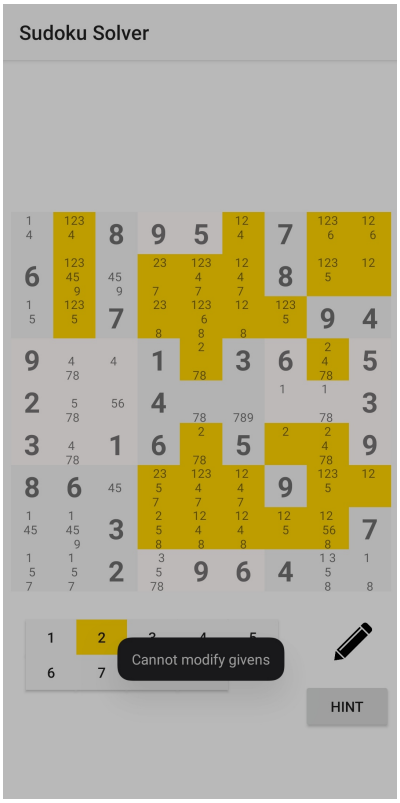


Figure B.5: Error message displayed when user attempts to modify givens.

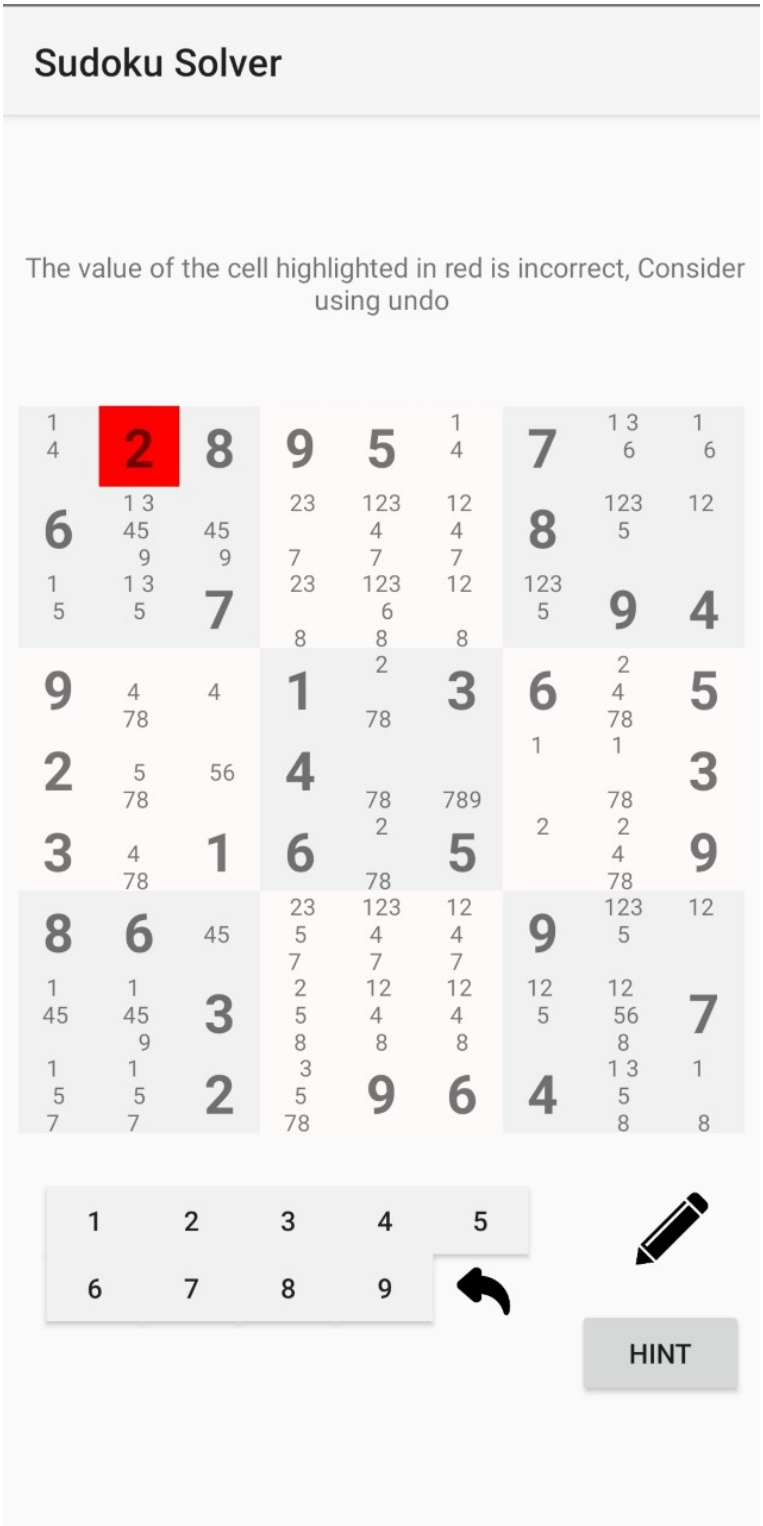


Figure B.6: Example of UI response when a cell is filled incorrectly. Incorrect cell is highlighted in red and an explanatory message is shown in the text box

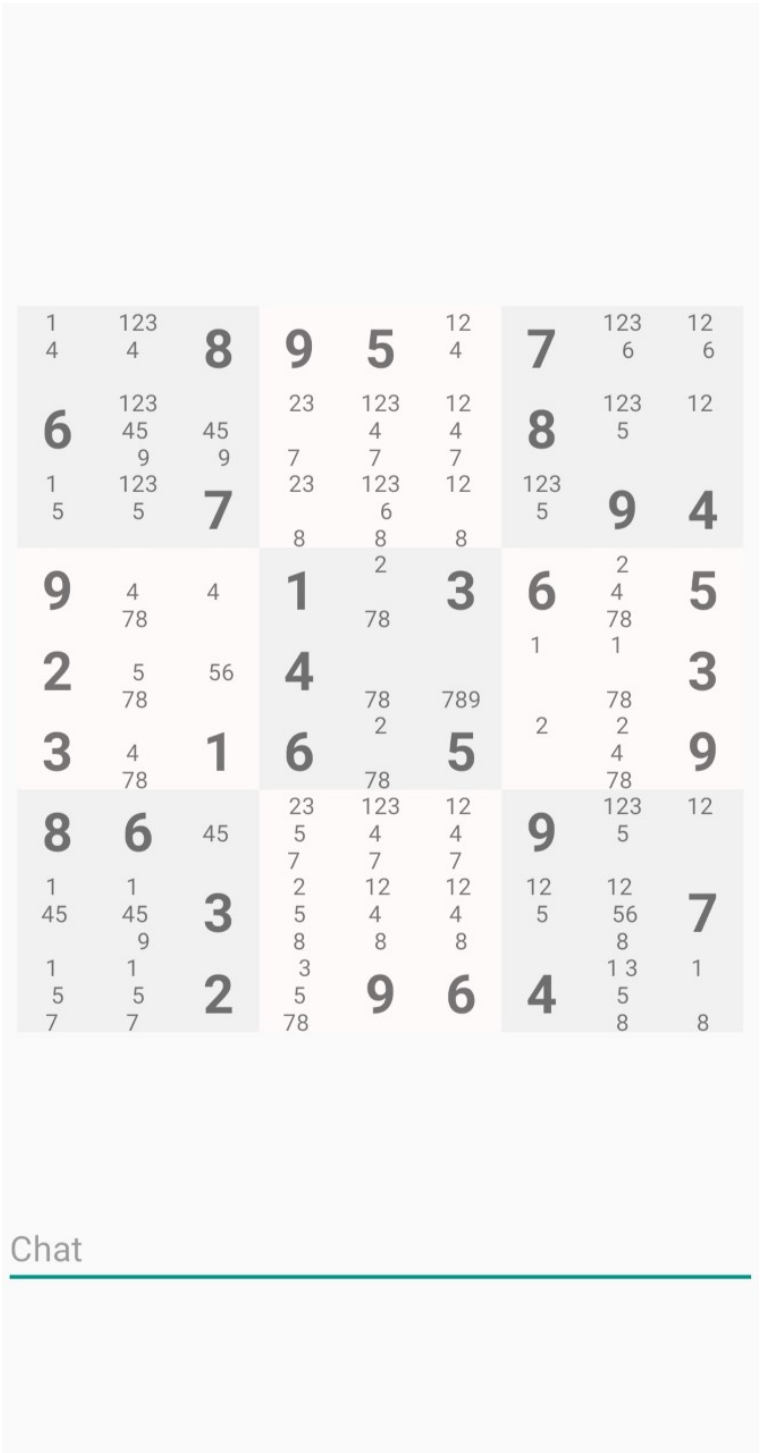


Figure B.7: Spectator user interface

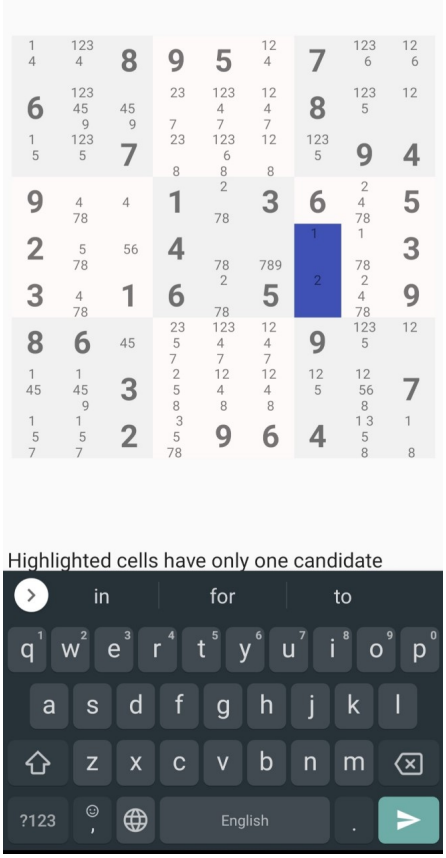


Figure B.8: Screen of spectator highlighting cells and sending a message.

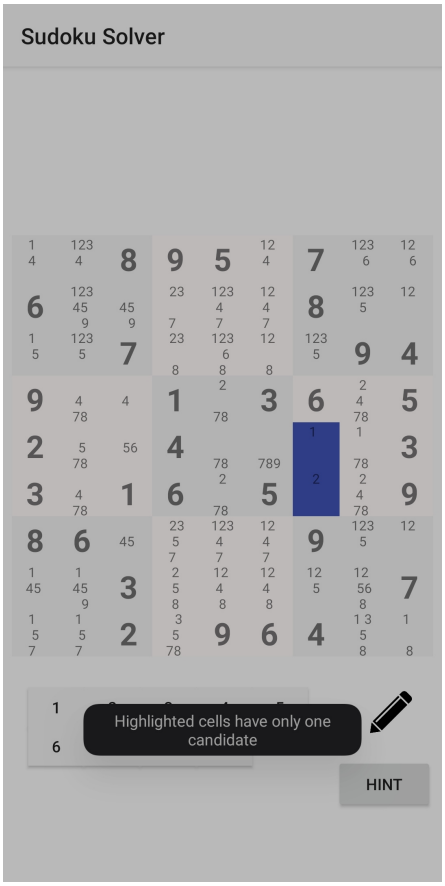


Figure B.9: Screen of player with highlighted cells and message at bottom.

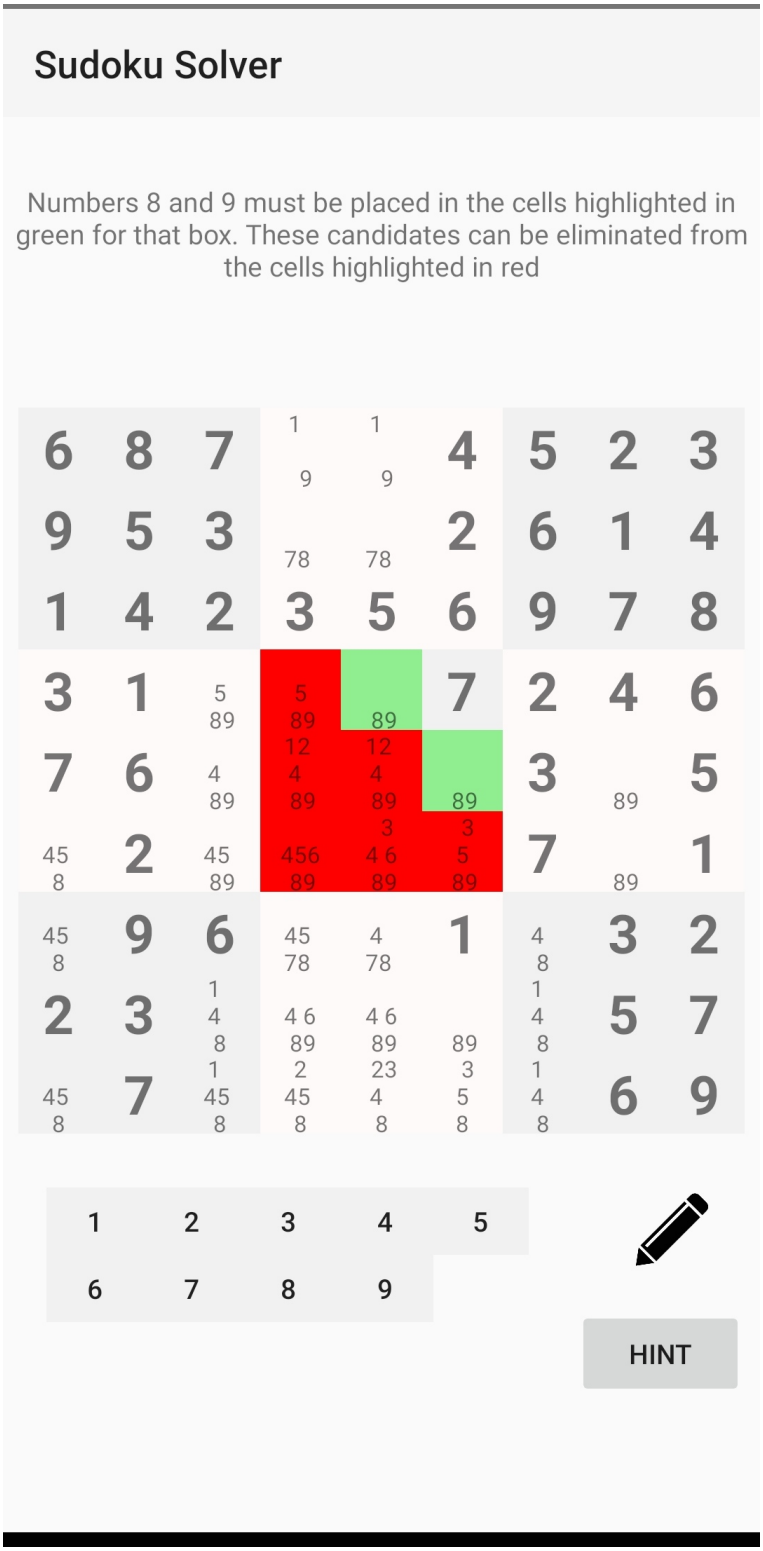


Figure B.10: Naked pairs hint example

Figure B.11: Hidden pairs hint example

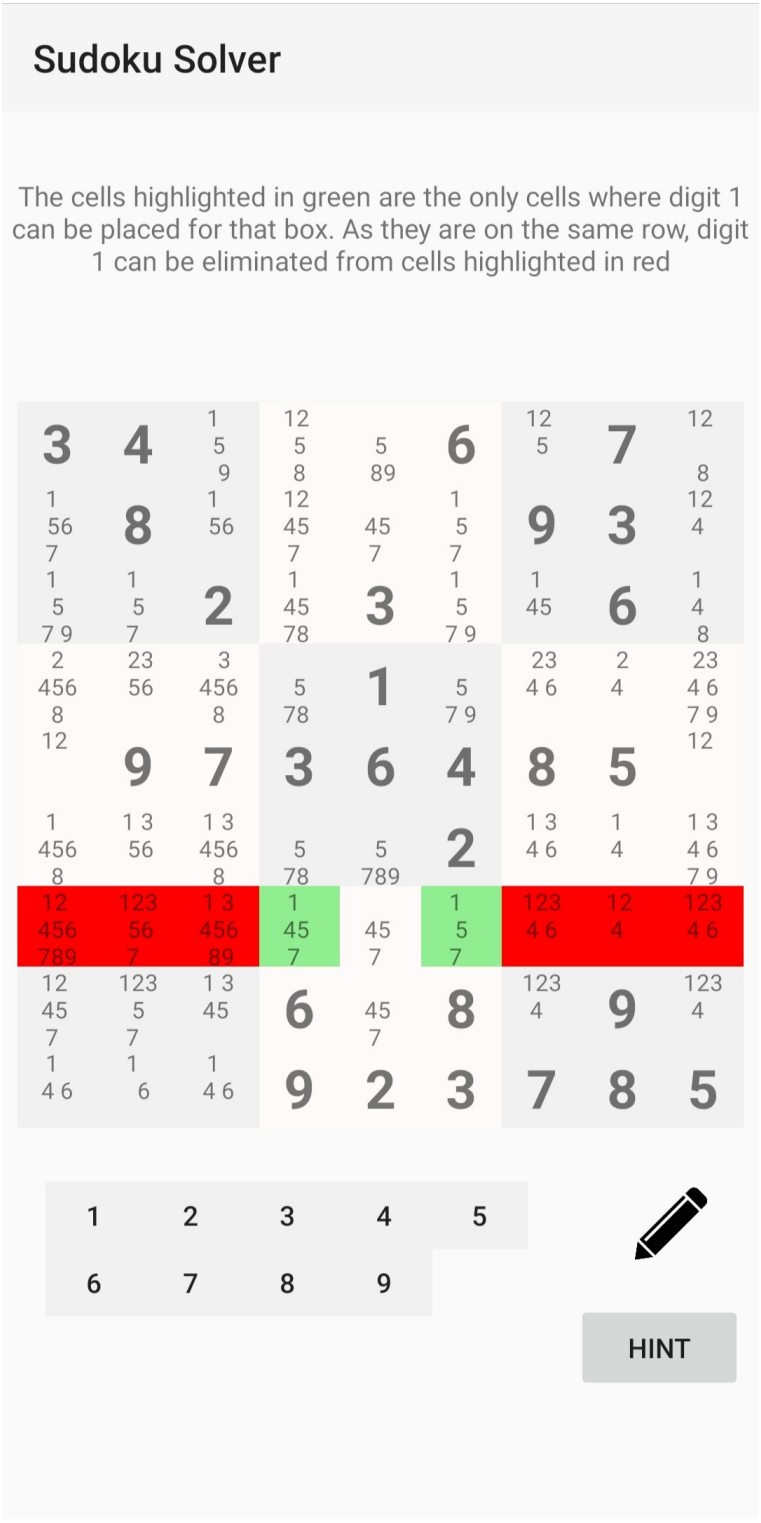


Figure B.12: Omission hint example

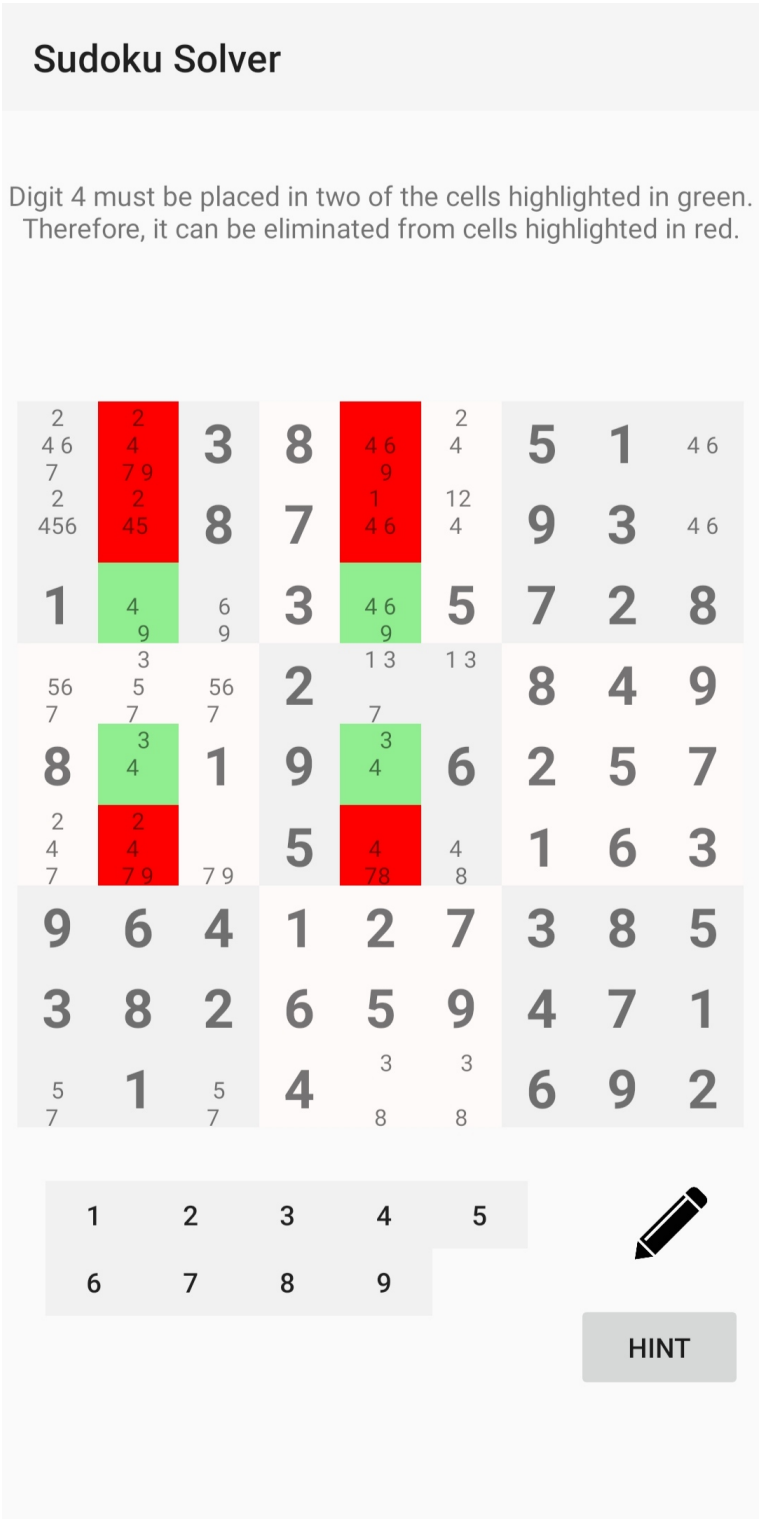


Figure B.13: X-wing hint example

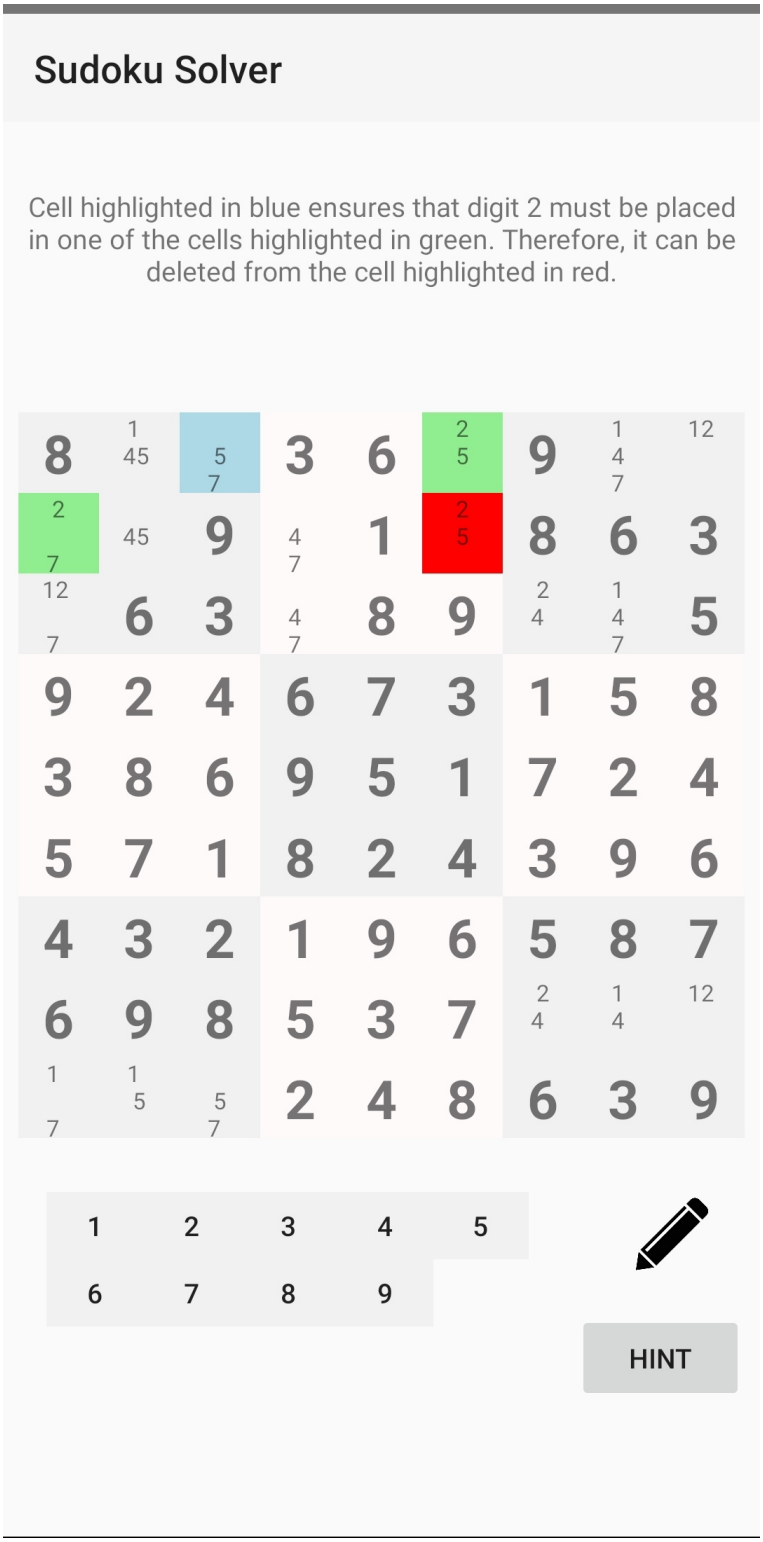


Figure B.14: XY-wing hint example

Appendix C

Participant Information Sheet

This study was certified according to the Informatics Research Ethics Process, RT number **2019/45209**. Please take time to read the following information carefully. You should keep this page for your records.

Who are the researchers?

Nikolas Pilavakis (Author) Nigel Topham (Supervisor)

What is the purpose of the study? Evaluate the usability and performance of a Sudoku solving android app.

Why have I been asked to take part? Anyone who enjoys Sudoku and has a few minutes to spare can take part. An android phone is required.

Do I have to take part? No – participation in this study is entirely up to you. You can withdraw from the study at any time, up until completing the questionnaire, where the study is concluded, without giving a reason. After this point, personal data will be deleted and anonymised data will be combined such that it is impossible to remove individual information from the analysis. Your rights will not be affected. If you wish to withdraw, contact the PI. We will keep copies of your original consent, and of your withdrawal request.

What will happen if I decide to take part? You will be asked to install an android app on your personal phone and interact with it for 5-10 minutes before completing a short questionnaire. No personal data will be required and you will not be contacted again (unless desired).

Are there any risks associated with taking part?

There are no significant risks associated with participation.

Are there any benefits associated with taking part?

No

What will happen to the results of this study?

The results of this study may be summarised in published articles, reports and presentations. Quotes or key findings will be anonymized: We will remove any information that could, in our assessment, allow anyone to identify you. With your consent, information can also be used for future research. Your data may be archived for a maximum of 6 months. All potentially identifiable data will be deleted within this timeframe if it has not already been deleted as part of anonymization.

Data protection and confidentiality.

Your data will be processed in accordance with Data Protection Law. All information collected about you will be kept strictly confidential. Your data will be referred to by a unique participant number rather than by name. Your data will only be viewed by the researcher/research team. I.e. Me (Nikolas Pilavakis) and my supervisor (Nigel Topham), if necessary All electronic data will be stored on a password-protected encrypted computer, on the School of Informatics' secure file servers, or on the University's secure encrypted cloud storage services (Sharepoint) and all paper records will be stored in a locked filing cabinet in the PI's office. Your consent information will be kept separately from your responses in order to minimise risk.

What are my data protection rights?

The University of Edinburgh is a Data Controller for the information you provide You have the right to access information held about you. Your right of access can be exercised in accordance Data Protection Law. You also have other rights including rights of correction, erasure and objection. Accessing data will be impossible after the submission of the questionnaire, as data submitted is not associated with the participant. For more details, including the right to lodge a complaint with the Information Commissioner's Office, please visit www.ico.org.uk. Questions, comments and requests about your personal data can also be sent to the University Data Protection Officer at dpo@ed.ac.uk.

Who can I contact?

If you have any further questions about the study, please contact the lead researcher, Nikolas Pilavakis on s1623062@sms.ed.ac.uk If you wish to make a complaint about the study, please contact inf-ethics@inf.ed.ac.uk. When you contact us, please provide the study title and detail the nature of your complaint. Updated information. If the research project changes in any way, an updated Participant Information Sheet will be made available on http://web.inf.ed.ac.uk/infweb/research/study-updates.

Alternative formats

To request this document in an alternative format, such as large print or on coloured paper, please contact Nikolas Pilavakis on s1623062@sms.ed.ac.uk

General information.

For general information about how we use your data, go to: edin.ac/privacy-research

Appendix D

Evaluation questionnaire

For demonstration purposes, screenshots of the questionnaire used for evaluation are provided. These screenshots correspond to the desktop version. Similar results are displayed when the form is visited from a mobile device.

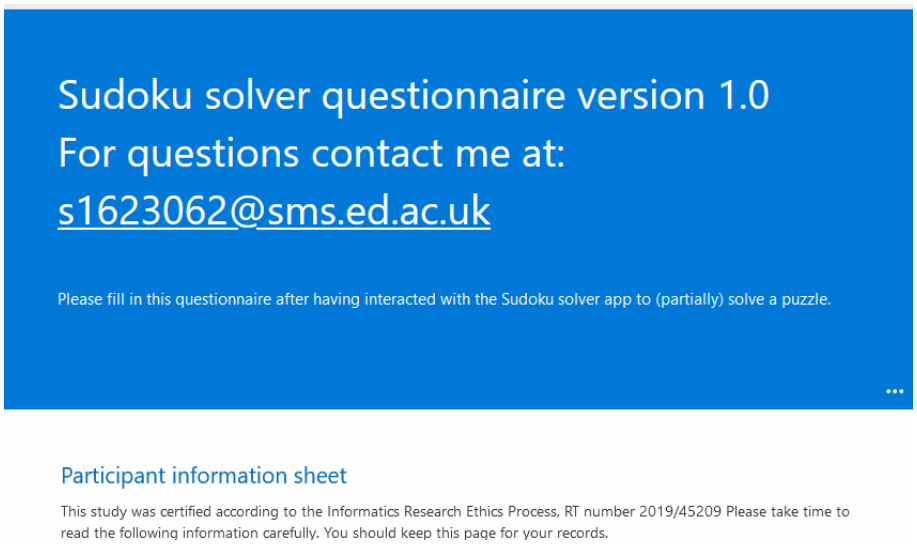


Figure D.1: First section of the questionnaire. The Participant Information Sheet (as shown on appendix C is displayed first to ensure that users at least scroll past it.

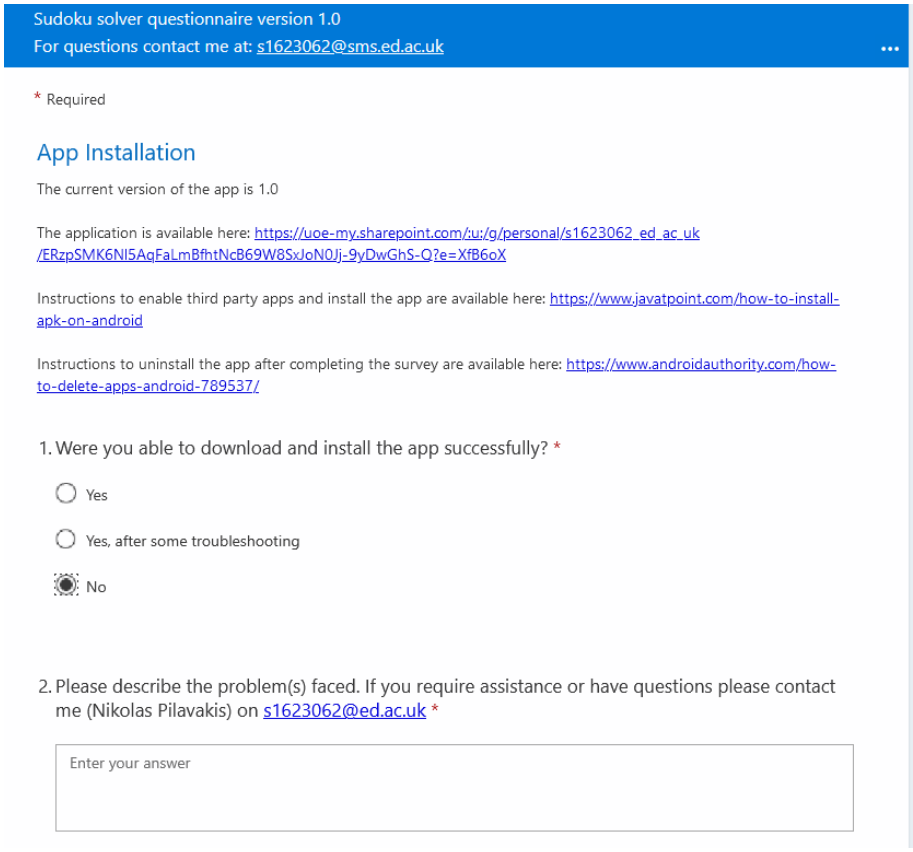


Figure D.2: Second section of the questionnaire. The section begins with instructions to install and uninstall the app. If the user was unable to install the app, an additional question (2) appears, asking for details and prompting the user to contact the author.

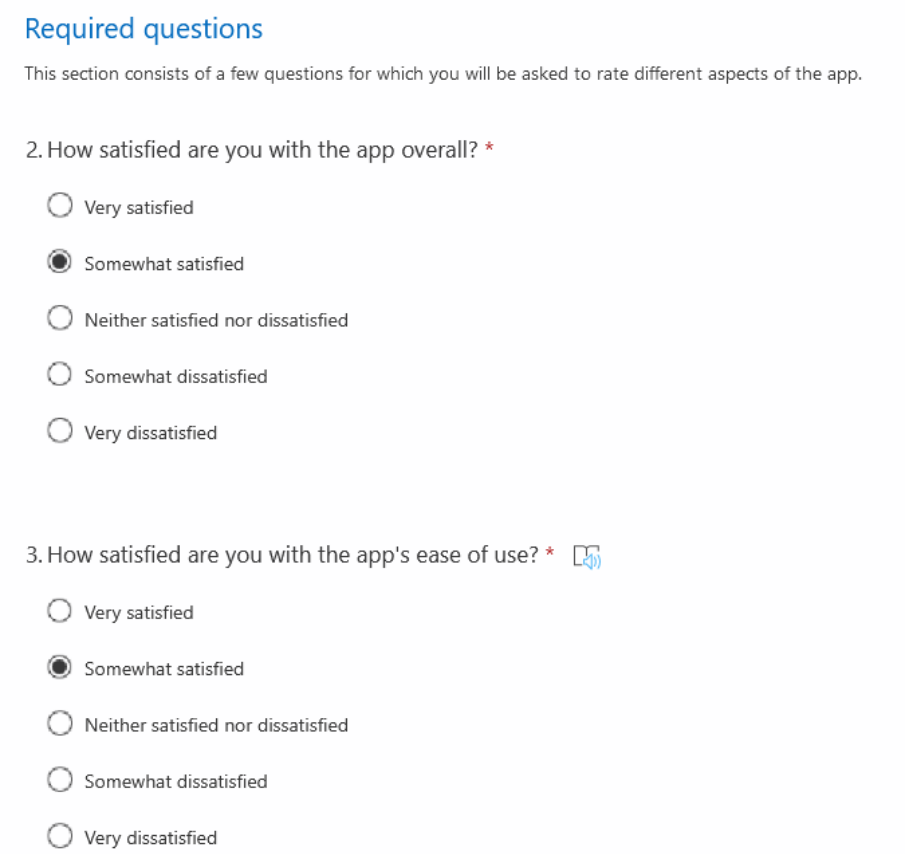


Figure D.3: First part of third section of the questionnaire. This part is mandatory and does not involve branching.

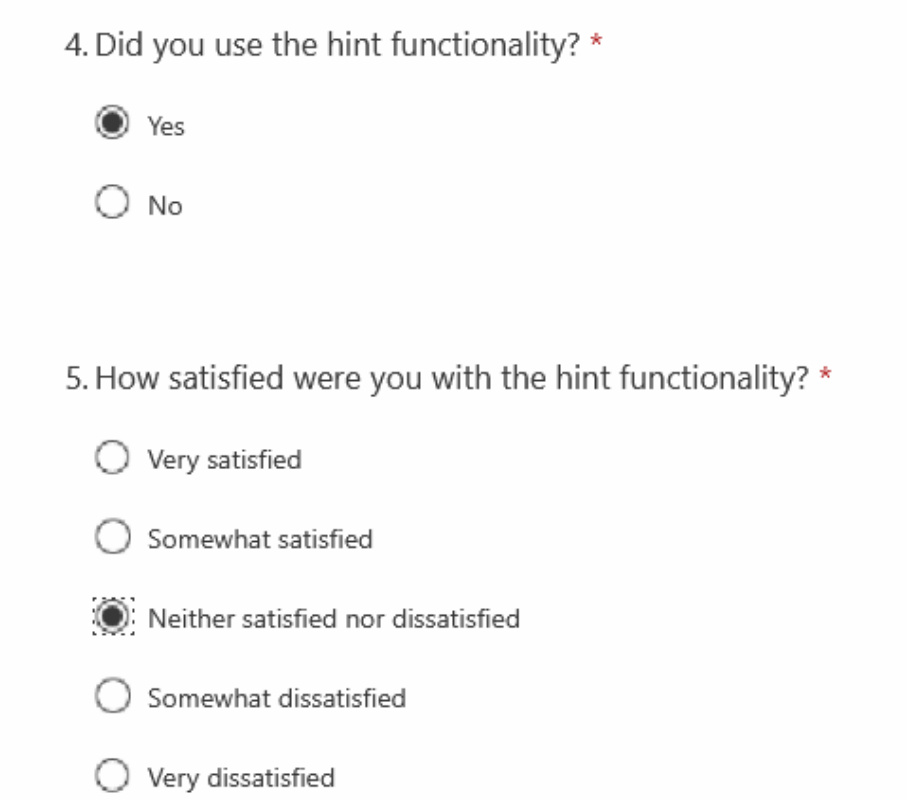


Figure D.4: Second part of the third section of the questionnaire. Question 5 is displayed only if the user answered ”Yes” in question 4.

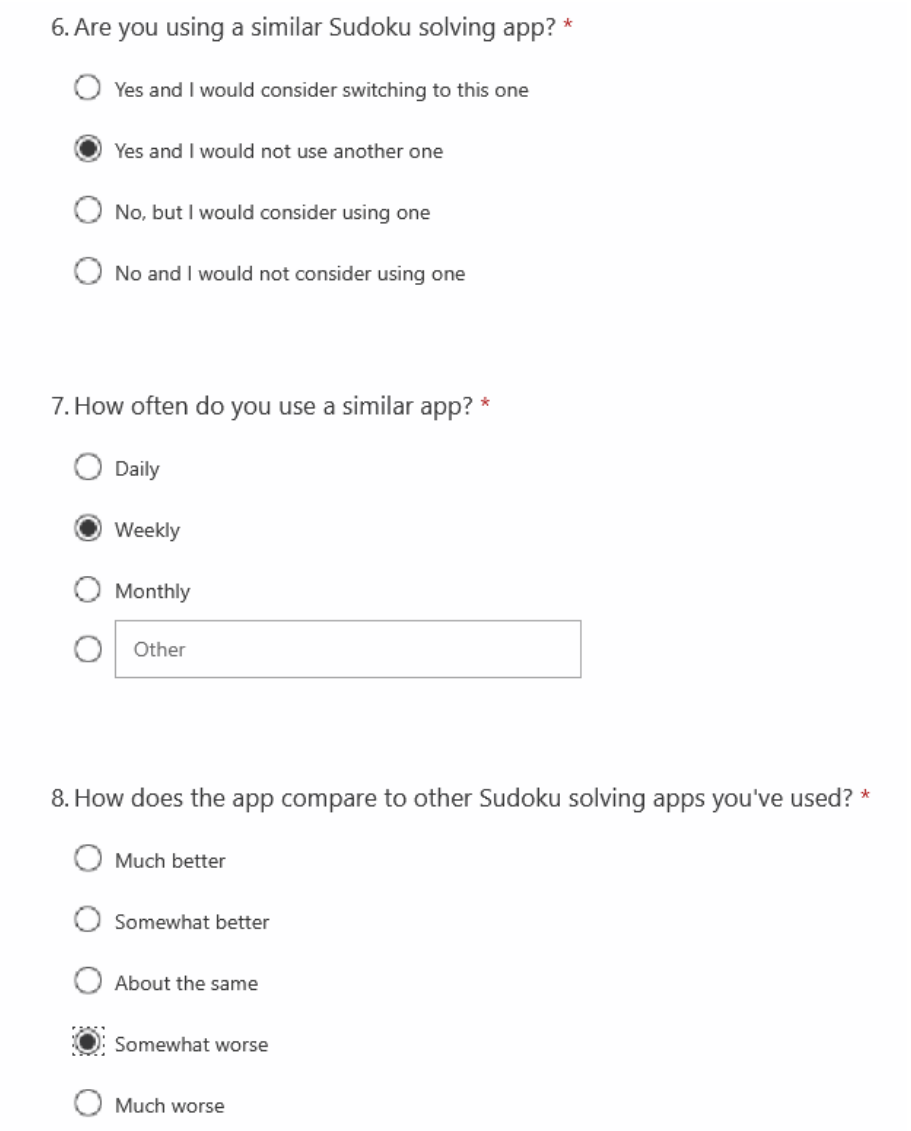


Figure D.5: Third part of the third section of the questionnaire. Questions 7 and 8 are displayed only if the user answered positively in question 6.

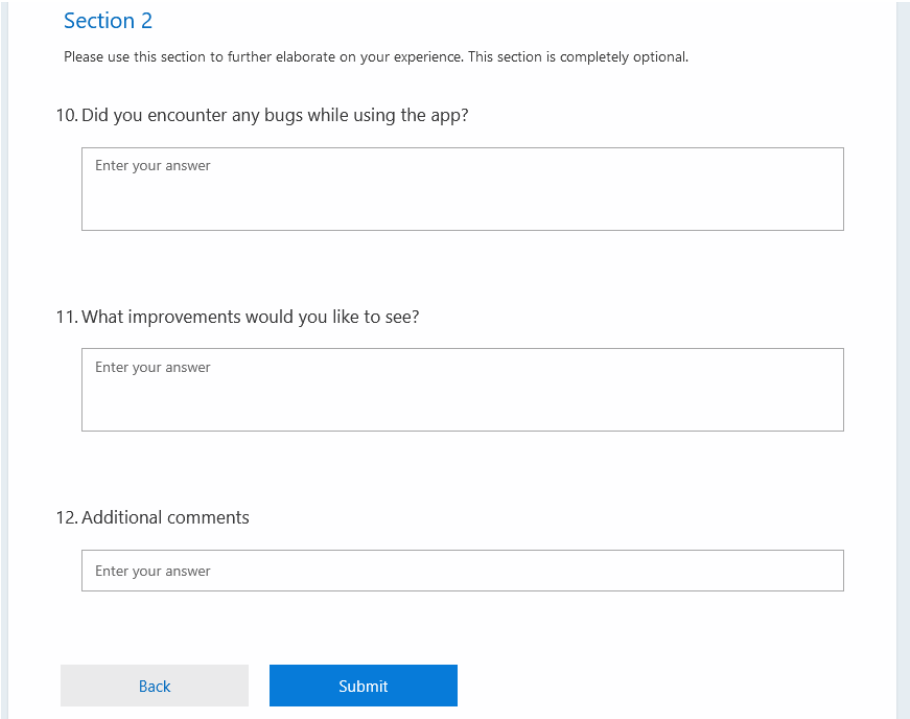


Figure D.6: Last section of the questionnaire. Three optional open-ended questions.

Figure D.7: Navigation buttons, found at the bottom of every section.

Appendix E

Responses distribution

This appendix contains supplementary plots of the distribution of the responses given by users that filled in the questionnaire.

Figure E.1: Were you able to download and install the app successfully?

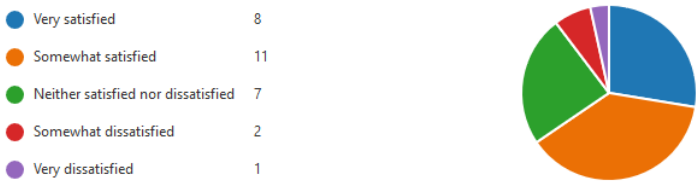


Figure E.2: How satisfied are you with the app overall?

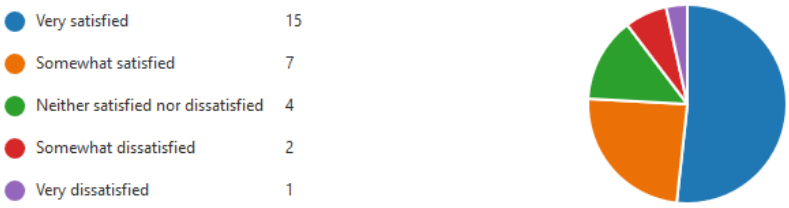


Figure E.3: How satisfied are you with the app's ease of use?

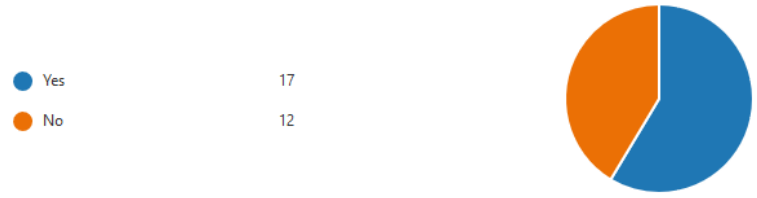


Figure E.4: Did you use the hint functionality?

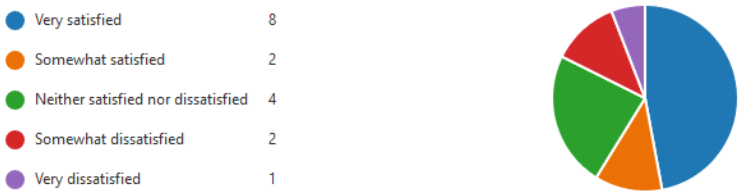


Figure E.5: How satisfied were you with the hint functionality?

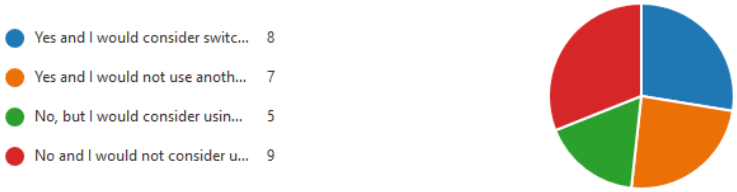


Figure E.6: Are you using a similar Sudoku solving app?

Figure E.7: How often do you use a similar app?

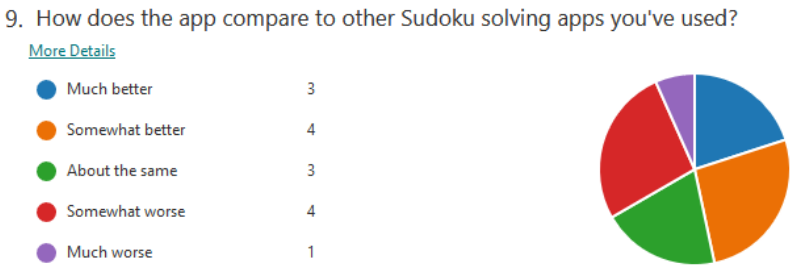


Figure E.8: How does the app compare to other Sudoku solving apps you've used?

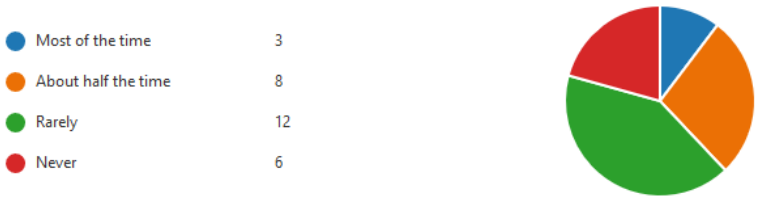


Figure E.9: How often do you need help when solving a Sudoku puzzle?

Bibliography

[1] Google groups. https://groups.google.com/forum/#!topic/rec.puzzles/A7pi7S12oFI.

[2] An incomplete review of sudoku solver implementations, Jul 2011. https://attractivechaos.wordpress.com/2011/06/19/an-incomplete-review-of-Sudoku-solver-implementations/.

[3] Krzysztof R. Apt. Principles of constraint programming. 2003.

[4] A. C. Bartlett, Timothy P. Chartier, Amy Nicole Langville, and Timothy D. Rankin. An integer programming model for the sudoku problem. 2006.

[5] Haradhan Chel, Deepak Mylavarapu, and Deepak Sharma. A novel multi-stage genetic algorithm approach for solving sudoku puzzle. *2016 International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT)*, pages 808–813, 2016.

[6] Abu Sayed Chowdhury and Suraiya Akhter. Solving sudoku with boolean algebra. 2012.

[7] Broderick Crawford, Margaret Aranda, Carlos Castro, and Eric Monfroy. Using constraint programming to solve sudoku puzzles. *2008 Third International Conference on Convergence and Hybrid Information Technology*, 2:926–931, 2008.

[8] Dipti Deodhare, Shailesh Sonone, and Anubha Gupta. A generic membrane computing-based sudoku solver. *2014 International Conference on Issues and Challenges in Intelligent Computing Techniques (ICICT)*, pages 89–99, 2014.

[9] Michael Dittrich, Thomas B. Preußer, and Rainer G. Spallek. Solving sudokus through an incidence matrix on an fpga. *2010 International Conference on Field-Programmable Technology*, pages 465–469, 2010.

[10] Mária Ercsey-Ravasz and Zoltán Toroczkai. The chaos within sudoku. In *Scientific reports*, 2012.

[11] Bertram Felgenhauer and Frazer Jarvis. Enumerating possible sudoku grids. 2005.

[12] Bertram Felgenhauer and Frazer Jarvis. Mathematics of sudoku ii. 2006.

[13] Zong Woo Geem. Harmony search algorithm for solving sudoku. In *KES*, 2007.

[14] Cristina Gonzalez, Javier Olivito, and Javier Resano. An initial specific processor for sudoku solving. *2009 International Conference on Field-Programmable Technology*, pages 530–533, 2009.

[15] Peter Gordon and Frank Longo. *Mensa guide to solving sudoku: hundreds of puzzles plus techniques to help you crack them all*. Sterling Pub. Co., 2006.

[16] Jacob H. Gunther and Todd K. Moon. Entropy minimization for solving sudoku. *IEEE Transactions on Signal Processing*, 60:508–513, 2012.

[17] Agnes M. Herzberg and M. Ram Murty. Sudoku squares and chromatic polynomials. 2007.

[18] Ellis Horowitz, Sartaj Sahni, and Sanguthevar Rajasekaran. Computer algorithms. 1996.

[19] Rohit Iyer, Amrish Jhaveri, and Krutika Parab. A review of sudoku solving using patterns. 2013.

[20] Sunanda Jana, Arnab Kumar Maji, and Rajat Kumar Pal. A novel sudoku solving technique using column based permutation. *2015 International Symposium on Advanced Computing and Communication (ISACC)*, pages 71–77, 2015.

[21] Dhanya Job and Varghese Paul. Recursive backtracking for solving 9*9 sudoku puzzle. 2016.

[22] Snigdha Kamal, Simarpreet Singh Chawla, and Nidhi Goel. Detection of sudoku puzzle using image processing and solving by backtracking, simulated annealing and genetic algorithms: A comparative analysis. *2015 Third International Conference on Image Information Processing (ICIIP)*, pages 179–184, 2015.

[23] Snigdha Kamal, Simarpreet Singh Chawla, and Nidhi Goel. Identification of numbers and positions using matlab to solve sudoku on fpga. *2015 Annual IEEE India Conference (INDICON)*, pages 1–6, 2015.

[24] Dervis Karaboga. An idea based on honey bee swarm for numerical optimization. 2005.

[25] Donald E. Knuth. Dancing links. 2000.

[26] Bettina Laugwitz, Theo Held, and M. Schrepp. Construction and evaluation of a user experience questionnaire. In *USAB*, 2008.

[27] Wei-Meng Lee. *Programming Sudoku*. Apress, 2006.

[28] Rhyd Lewis. Metaheuristics can solve sudoku puzzles. *Journal of Heuristics*, 13:387–401, 2007.

[29] Hung-Hsuan Lin and I-Chen Wu. Solving the minimum sudoku poblem. *2010 International Conference on Technologies and Applications of Artificial Intelligence*, pages 456–461, 2010.

[30] Arnab Kumar Maji and Rajat Kumar Pal. Sudoku solver using minigrid based backtracking. *2014 IEEE International Advance Computing Conference (IACC)*, pages 36–44, 2014.

[31] Pavlos Malakonakis, Miltiadis Smerdis, Euripides Sotiriades, and Apostolos Dollas. An fpga-based sudoku solver based on simulated annealing methods. *2009 International Conference on Field-Programmable Technology*, pages 522–525, 2009.

[32] Timo Mantere. Improved ant colony genetic algorithm hybrid for sudoku solving. *2013 Third World Congress on Information and Communication Technologies (WICT 2013)*, pages 274–279, 2013.

[33] Timo Mantere and Janne Koljonen. Solving and rating sudoku puzzles with genetic algorithms. 2006.

[34] Timo Mantere and Janne Koljonen. Solving, rating and generating sudoku puzzles with ga. *2007 IEEE Congress on Evolutionary Computation*, pages 1382–1389, 2007.

[35] Gary McGuire, Bastian Tugemann, and Gilles Civario. There is no 16-clue sudoku: Solving the sudoku minimum number of clues problem. *Experimental Mathematics*, 23:190–217, 2012.

[36] P. Mincheva and R. Anastasova. Tele-occupational therapy: Experience with bulgarian children during covid-19 pandemic. In *2020 International Conference on Assistive and Rehabilitation Technologies (iCareTech)*, pages 67–70, 2020.

[37] R. Molich and J. Nielsen. Improving a human-computer dialogue. *Commun. ACM*, 33:338–348, 1990.

[38] Todd K. Moon, Jacob H. Gunther, and J. J. Kupin. Sinkhorn solves sudoku. *IEEE Transactions on Information Theory*, 55:1741–1746, 2009.

[39] Alberto Moraglio, Julian Togelius, and Simon M. Lucas. Product geometric crossover for the sudoku puzzle. *2006 IEEE International Conference on Evolutionary Computation*, pages 470–476, 2006.

[40] J. Nielsen. Enhancing the explanatory power of usability heuristics. In *CHI '94*, 1994.

[41] J. Nielsen. Heuristic evaluation. 1994.

[42] J. Nielsen and R. Molich. Heuristic evaluation of user interfaces. In *CHI '90*, 1990.

[43] Jaysonne A. Pacurib, Glaiza Mae M. Seno, and John Paul T. Yusiong. Solving sudoku puzzles using improved artificial bee colony algorithm. *2009 Fourth International Conference on Innovative Computing, Information and Control (ICICIC)*, pages 885–888, 2009.

[44] M. Pesce. Peering into the pandemic end game: Before covid–19 fades, we'll see a flurry of advances in contact tracing, cloud computing, surveillance, and online gaming. *IEEE Spectrum*, 58(1):22–25, 2021.

[45] Haiyan Quan and Xinling Shi. On the analysis of performance of the improved artificial-bee-colony algorithm. pages 654 – 658, 11 2008.

[46] Francesca Rossi, Peter van Beek, and Toby Walsh. Handbook of constraint programming (foundations of artificial intelligence). 2006.

[47] Ibrahim Sabuncu. Work-in-progress: Solving sudoku puzzles using hybrid ant colony optimization algorithm. *2015 1st International Conference on Industrial Networks and Intelligent Systems (INISCom)*, pages 181–184, 2015.

[48] Liu San-yang. Algorithm based on genetic algorithm for sudoku puzzles. 2010.

[49] Gustavo Santos-García and Miguel Palomino. Solving sudoku puzzles with rewriting rules. *Electr. Notes Theor. Comput. Sci.*, 176:79–93, 2007.

[50] Yuji Sato and Hazuki Inoue. Genetic operations to solve sudoku puzzles. In *GECCO*, 2010.

[51] Moriel Schottlender. The effect of guess choices on the efficiency of a backtracking algorithm in a sudoku solver. *IEEE Long Island Systems, Applications and Technology (LISAT) Conference 2014*, pages 1–6, 2014.

[52] M. Schrepp, Andreas Hinderks, and J. Thomaschewski. Applying the user experience questionnaire (ueq) in different evaluation scenarios. In *HCI*, 2014.

[53] M. Schrepp, Andreas Hinderks, and J. Thomaschewski. Construction of a benchmark for the user experience questionnaire (ueq). *Int. J. Interact. Multim. Artif. Intell.*, 4:40–44, 2017.

[54] Helmut Simonis. Sudoku as a constraint problem. 2005.

[55] Ricardo Soto, Broderick Crawford, Cristian Galleguillos, Eric Monfroy, and Fernando Paredes. A hybrid ac3-tabu search algorithm for solving sudoku puzzles. *Expert Syst. Appl.*, 40:5817–5821, 2013.

[56] Ricardo Soto, Broderick Crawford, Cristian Galleguillos, Eric Monfroy, and Fernando Paredes. A prefiltered cuckoo search algorithm with geometric operators for solving sudoku problems. In *TheScientificWorldJournal*, 2014.

[57] Ricardo Soto, Broderick Crawford, Cristian Galleguillos, Fernando Paredes, and Enrique Norero. A hybrid alldifferent-tabu search algorithm for solving sudoku puzzles. In *Comp. Int. and Neurosc.*, 2015.

[58] Ricardo Soto, Broderick Crawford, Cristian Galleguillos, Francisca C. Venegasy, and Fernando Paredes. A marriage theorem based-algorithm for solving sudoku. *2015 Fourteenth Mexican International Conference on Artificial Intelligence (MICAI)*, pages 117–121, 2015.

[59] Kees van der Bok, Mottaqiallah Taouil, Panagiotis Afratis, and Ioannis Sourdis. The tu delft sudoku solver on fpga. *2009 International Conference on Field-Programmable Technology*, pages 526–529, 2009.

[60] Zhiwen Wang, Toshiyuki Yasuda, and Kazuhiro Ohkura. An evolutionary approach to sudoku puzzles with filtered mutations. *2015 IEEE Congress on Evolutionary Computation (CEC)*, pages 1732–1737, 2015.

[61] Baptiste Wicht and Jean Hennebert. Camera-based sudoku recognition with deep belief network. *2014 6th International Conference of Soft Computing and Pattern Recognition (SoCPaR)*, pages 83–88, 2014.

[62] Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. 2003.

[63] John Paul T. Yusiong and Jaysonne A. Pacurib. Sudokubee : An artificial bee colony-based approach in solving sudoku puzzles. 2010.