A Sudoku Solving App

Student Name: Lendl Munoz
Student Number: 33468452
Degree Title: BSc Computer Science
Supervisor Name: Dr Nikolay Nikolaev
Institution: Goldsmiths University of London
Word Count: 11,924

Contents

1 Introduction

1.1 Overview

Sudoku is a number placement puzzle that dates back to the 18th century game called “Latin Squares.” However, the modern Sudoku that we know today was developed in 1979 by Howard, Garns, an American architect (The Economist, 2005). The name is derived from the Japanese word *Su-ji wa dokushin ni kagiru* which translates to "the numbers must be single."

The global craze of sudoku puzzles was particularly relevant in 2005 when newspapers around the world were competing feverishly to publish as many sudokus as possible. At one point, The Guardian had posted a puzzle on every page of their Features section. Although the business of puzzle games is difficult to measure, at the height of its time, the combined revenues of magazines, newspapers and puzzle books had made $200 million annually (The Economist, 2005). Fortunately, there are 6,670,903,752,021,072,936,960 (~10^{21}) possible solved sudoku grids (Kiersz, 2019).

Sudoku is played on a 9 x 9 grid of 81 cells. These cells create nine 3 x 3 squares, where each square, column and row must be filled with the numbers 1 - 9 without repeating in the square, column, or row. Rather than being a math-based puzzle, it is best described as a logic-based one, making it the perfect subject to explore as a computational problem.

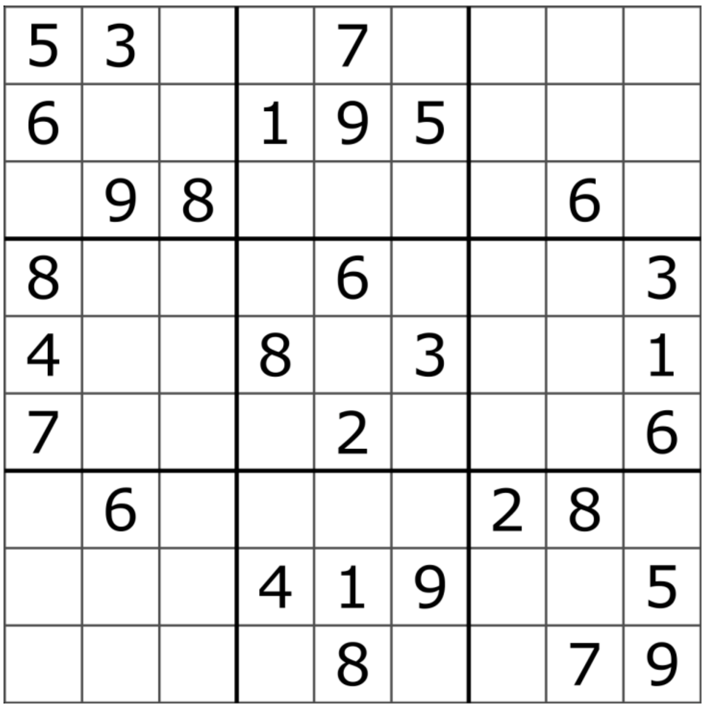


Figure 1. Diagram of a sudoku

This project aims to create a program that will enable a camera to scan and solve a game of sudoku. The technology it will use is image recognition using a smartphone camera and a Sudoku-solving algorithm. Whilst these technologies have developed in their own respects, there is a lack of its combined use within physical puzzle games, which should be investigated as many of their advantages could greatly improve the player's experience.

Although the mobile alternative is readily available, many people continue to opt for the analog versions. It has become significantly evident this year, as the COVID-19 pandemic has brought people indoors and seen an increase in puzzles and online subscriptions as a method to curb at-home anxiety (Williangham, 2020). Furthermore, 360 Research's report predicted that the puzzle industry's value could potentially reach $730 million by 2024 (More, 2020). Thus, showing that the demand for this app is still relevant and possibly making an even bigger comeback from 2005.

With this program, I am hoping to solve the problem users may experience when dealing with the more traditional sudokus found in newspapers, puzzle books, etc. Users who prefer the paper-format sudokus have a difficulty in retrieving hints without having the entire grid revealed. Building an app like this would allow these puzzles to be at an almost equal footing to its mobile counterpart. It would give these users the ease of hints in a similar fashion that mobile users have become accustomed to. The program I aim to create will give its users access to as many answers they wish to see in order to help them solve their puzzle.

1.2 Motivation

I believe sudokus are a great way to stimulate your mind as you are doing more logical thinking when solving them, whilst also improving memory through pattern recognition. However, not everyone may find it easy to finish a sudoku or have the motivation to complete it if they encounter a block and cannot continue logically. Even experienced solvers, who are efficient at solving sudokus at varying difficulties, may find extremely difficult puzzles hard to even begin without a few extra numbers.
Currently, sudoku apps on smartphones would usually offer hints for the users but this is not often available to those playing on a paper-formatted one. I want to create an application for mobile phones that can quickly help the users if they are stuck. I would also give the users more freedom on how much help they would like from the application i.e. if they are unsure of a single number, cannot find any other number in the grid logically or just want the whole sudoku solved - as this is also not an option for existing applications.

1.3 Overview of the Chapters

The remaining report will be structured in the following way: Chapter 2 will cover the background secondary research on search algorithms and computer vision. It will weigh out the advantages and disadvantages as well as uses of the aforementioned and will end with an evaluation of what would be the most efficient and suitable choice. Chapter 3 presents the primary research undertaken to provide data on what users are looking for in this app. Chapter 4 explains the specification of the project. Chapter 5 will contain the design of the application, followed by Chapter 6 which will describe the way it will be implemented. Chapter 7 covers the testing sets, looking specifically at two individual users. Chapter 8 will discuss the successes of the project and how this can be improved with further development. Lastly, Chapter 9 will conclude the report and give an overview of what has been completed. This will all be accompanied with a relevant bibliography and appendices, to present further detail to any work achieved in this project.

2 Secondary Research

2.1 History of Search Algorithms

Search algorithms are used to check or retrieve an element from any data structure that it is stored in. Typically, these algorithms are separated into two categories:

1. Sequential search - in data structures such as lists or arrays, they are considered to have a linear relationship among the data items. Each item is stored in a positive relative to the others and in most cases, are indexed individually. This allows us to visit each of them in sequence (Miller and Ranum, 2006). E.g. linear search.
2. Interval search - these algorithms are designed for data structures that are already sorted. They would commonly start around the centre of the structure and divide the search space in half to begin that process again until the data item has been found (Miller and Ranum, 2006). E.g. binary search.

Interval search algorithms can be considered more efficient than sequential ones. This is because in sequential searches, the worst case scenario would be if the last item in the data structure was the one you are looking for therefore making its search time ***O(n)*** where ***n*** is the number of comparisons made. However, a binary search would recursively divide the data structure and compare whether the target item is larger or smaller than the breakpoint until it is found, meaning the maximum number of comparisons are logarithmic with respect to the number of items in list, meaning its search time is ***O(log n)***. With that said, in binary search you will also have to factor in the cost of sorting the items which may not be beneficial when it comes to sorting data structures of small and large lists as its costs may not be efficient.

2.1.1 The Use of Search Algorithms

These algorithms are used for retrieving data stored within a data structure. While the uses of it are limited, the vast number of different search algorithms means there are specific applications for each one.

These algorithms can be applied into complicated problems such as ones in combinatorial optimization, which involves finding an optimal object from a finite set of objects (Schrijver, 2003). Problems in this area include the knapsack problem, where you are given a set of items, each with its own weight and value, and will have to find a combination of the items that will have a total weight less than or equal to a given limit and will also provide the largest total value. Solutions to these types of problems can be applied into areas like job allocation or logistics, where efficient distribution of given resources is necessary.

Another application of search algorithms can be found in problems in constraint satisfaction. This involves finding a solution to a problem containing a set of constraints that impose conditions the variables must satisfy (Tsang, 2014). These types of problems exist when given a set of variables that can be used in a possible world to achieve a positive outcome to the problem. A possible world is defined as a possible way a world (real or imaginary) could exist (Poole and Mackworth, 2018). Solving a sudoku can be considered a constraint satisfaction problem as the constraints you are given are each row, column, and 3 x 3 grid within the sudoku can only contain the numbers 1 - 9 only once. Given these nonlinear constraints, I may use techniques such as backtracking or stochastic optimisation.

```
procedure EXPLORE(node n)
    if REJECT(n) then return
    if COMPLETE(n) then
        OUTPUT(n)
    for $n_i$ : CHILDREN(n) do EXPLORE($n_i$)
```

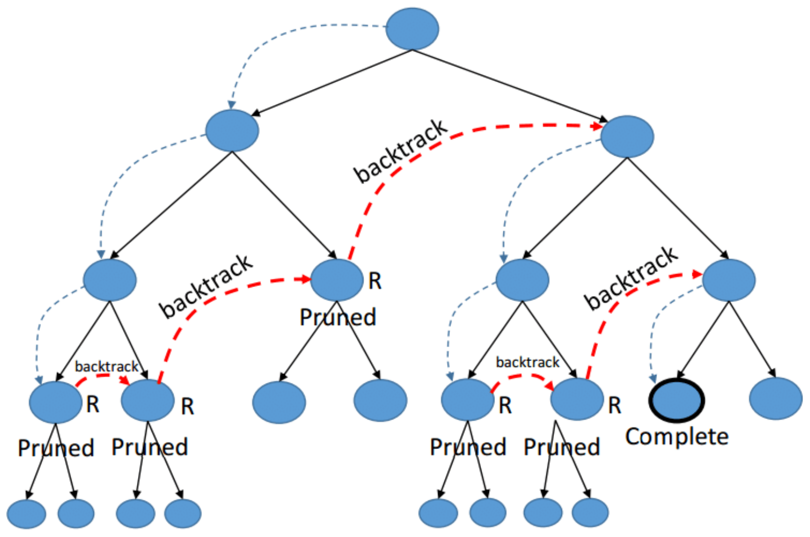


Figure 2. Diagram of a backtracking algorithm visualized (Verenich et al., 2017)

2.2 Backtracking

Backtracking is a solution to computational problems, many of which involve constraint satisfaction. It begins by inputting a new value into the first vector then continues onto the next stage where it can input the next value possible. Once the algorithm deems a particular value inputted cannot represent a partial solution, due to a violation of constraints, then the algorithm will traverse back (“backtrack”) to a previous vector and then proceed with a different value (Knuth, 2011).

Each state that is traversed in the algorithm can be represented as nodes in a tree structure, where a state would be a parent node to another set of states that will branch further. For example, if used in a game of sudoku, the first set of states would be the first possible input of a number between 1 and 9 on the grid then the states proceeding will follow the same pattern, ending a branch (“pruning”) once it cannot input anymore numbers on the grid successfully. This algorithm traverses the tree structure recursively, starting from the root node going down to the end of a branch, in a depth-first order.

2.2.1 Components

Backtracking requires a set of parameters in order for it to work. These are:

- The computational problem and data for the given instance.
- A set of constraints that need to be satisfied in order to solve the problem.
- A solution set that will be added onto incrementally as the algorithm builds up a solution (Chaturvedi, 2018).

2.2.2 Uses of Backtracking Algorithms

Given the parameters needed to apply backtracking into certain computational problems, it is clear that this algorithm works with puzzles involving nonlinear constraints. As well as sudokus, crosswords are also solvable using this algorithm and the *n*-queens puzzle - a puzzle where you are given *n* number of queens on a *n* x *n* sized board and you have to place the queens in such a way where no two queens are threatening each other i.e. they cannot be on the same row, column or diagonal.

It is also able to solve a number of combinatorial optimization problems like the aforementioned knapsack problem. These types of problems provide a set of constraints as well as given values for the problem to compute into the dataset which is ideal for backtracking.

2.2.3 Advantages and Disadvantages

Advantages:

- An effective method in constraint satisfaction problems
- It is easy to implement and code
- As it is a brute force method, a solution is guaranteed (granted there is one possible) since it is traversing through every possible outcome to the problem

Disadvantages:

- Because it is checking every possible outcome, it will need to be allocated a large amount of memory space thus making it quite costly
- It is not an efficient way of solving problems

2.3 Stochastic Optimisation

Stochastic optimisation methods are used to find the global optimum of a problem where many local optima can exist (Moraglio and Togelius, 2007). These methods involve generating random variables and assigning them into the problem. The number of errors are then calculated in its current iteration and the assigned variables are then randomised again to be calculated until a global optimum can be found. Due to its stochastic nature, application of this method does not require the problem to be logic-solvable, therefore allowing it to be applied to a wider range of problems.

There are many techniques involving stochastic optimisation that can be applied to the same problem while yielding different results. One technique in particular is the Genetic Algorithm optimisation, a population-based optimisation technique inspired by the Darwinian theory of Evolution. The population is formed of potential solutions to a given problem. Each possible solution, referred to as individuals, consist of a set of traits. Any individual with traits closely matching the most optimal solution is deemed "fit" and will combine with individuals which were also deemed fit. While bad traits may still be present within the fit individuals, the majority of them would have been eliminated allowing good traits to become more prevalent as the algorithm reproduces more individuals. This will continue until the optimal solution is found or it is as close to it as possible (Cantú-Paz and Goldberg, 2000).

Another technique is the Particle Swarm Optimisation which is also a population-based technique made to simulate the way bird flock towards food. When given a number of possible solutions, this technique will begin to iteratively improve each of them while maintaining a given degree of quality. These solutions, referred to as particles, begin moving around in the search-space where its position and velocity are adjusted via simple mathematical formulae. Each particle will begin to locate the best-known position in the search-space and will update its position accordingly while also influencing over particles to "swarm" towards the best solutions.

Simulated Annealing is also a stochastic optimisation technique but unlike the Genetic Algorithm and Particle Swarm Optimisation, it is not population based. This technique is inspired by the annealing process used to strengthen glass by heating it until molten and then cooling it slowly, allowing the molecules to settle into lower energy states. It alters and tracks the state of an individual while monitoring its energy state. As the temperature lowers, the individual will randomly select a different solution neighbouring the current one, measure its own energy state and decide to move to it according to the temperature-dependent probabilities - the lower the temperature, the less likely it will be to move to a worser solution. This temperature dependency allows the individual to move around the search-space to find better solutions without it getting stuck only on a local optimum. This process will continue once the global optimum is found or until the temperature reaches zero, where it will return its best solution (Kirkpatrick et al. 1983).

2.3.1 Components

In all stochastic optimisation methods, the shared features among them include:

- An initialisation process, where a solution space will be defined and represented.
- A fitness function that will ensure that the objectives and constraints are being adhered to (Perez and Marwala, n.d.).

2.3.2 Uses of Stochastic Optimisation

There are many uses of stochastic optimisation, many of which are involved in biology or economics. A few examples of its uses are (Spall, 2005):

- Managers at a firm are faced with making short- and long-term investments in order to increase profits.
- An aerospace engineer runs computer simulations and conducts wind tunnel tests to refine the design of a missile or aircraft.
- Researchers at a pharmaceuticals firm design laboratory experiments to extract the maximum information about the efficacy of a new drug.

2.3.3 Advantages and Disadvantages

Advantages:

- Applicable to many fields allowing these methods to be developed constantly and increase in efficiency
- Ability to solve even non logic-based problems

Disadvantages:

- When dealing with problems with a larger search-space, limitations must be set to avoid exhausting resources, thereby decreasing the accuracy of the final outcome
- While it is known to be fast, stochastic-based algorithms may not be faster than deductive techniques

2.4 Overview of Approaches

As stated in Perez and Marwala's findings (2008), they concluded that stochastic search techniques are an efficient way of solving a sudoku. A couple of their approaches were not able to find a solution to the sudoku problem - Repulsive Particle Swarm Optimisation and their first approach with Cultural Genetic Algorithm. This was due to how the particles and individuals were represented and manipulated. The search operations on both approaches implemented were not able to naturally adapt to generating better possible solutions.

The most efficient approach they had was a hybrid of the Genetic Algorithm and Simulated Annealing. This algorithm combines the parallel searching power of the Genetic Algorithm with the flexibility of Simulated Annealing. With this, they were able to find a solution to the puzzle in 1.447 seconds and only after 435 iterations. The research they made on this paper has shown me how effective a stochastic search algorithm is and has inspired me to use a similar approach to my solving algorithm.

In regard to backtracking, a brute force method does not seem as efficient as the stochastic optimisation methods even with its random nature. However, due to its recursive structure, it can be quickly implemented. It can also be used as a means to quickly determine if certain sudokus are solvable since it is a method that can guarantee a solution if there is one. With that reasoning, I believe my best approach will be to create a backtracking algorithm first as a benchmark for my application then use my remaining time to develop a stochastic search algorithm.

2.5 History of Computer Vision

Computer vision is an interdisciplinary field focusing on how computers are able to achieve a high-level of understanding from processing digital images or videos. In the early 1970s, computer vision was viewed as a visual perception component for an ambitious agenda to mimic human intelligence and apply it into robots with intelligent behaviour (Szeliski, 2010). From an engineering perspective, it aims to build autonomous systems which can perform similar tasks to the human visual systems to an equal degree of accuracy or, in some cases, greater (Huang, 1996). This field is tasked with producing numerical or symbolic values extracted from high-dimensional data. This can be obtained from digital images or videos through acquisition, processing, and analysing (Klette, 2014).

2.5.1 The Use of Computer Vision

There are many applications of computer vision in the real world making it one of our most important researches in the modern era. One significant use of computer vision is autonomous image analysis which is used in many fields. This particular technology can be used in the medicinal industry as a way to diagnose a patient. It is able to identify defects in the organs as well as take measurements. Computer vision can also help with enhancing images, such as X-rays or ultrasonic images.

Another application of computer vision is navigation. Autonomous vehicles make use of it by recreating its environment, knowing its own position and detecting obstacles to be avoided. This sort of technology allows it to be used in fields like the military - for missile guidance or reconnaissance - or space exploration - NASA’s autonomous rover on Mars, Curiosity.

2.6 Image Recognition

The ability to analyse an image and deduce whether certain objects are present within, was a common problem in computer vision and image processing. In order for the recognition to function, it is required to be trained beforehand for its specific task. It is given a set of images, most of which will contain the subject that the model will learn to recognise. This is done through deep learning methods involving convolutional neural networks, a class of deep neural networks. These neural networks contain multiple layers between the input and output layers. Each layer holds a number of neurons that each connect to every neuron on the next layer. These neutrons are manipulated by mathematical operations calculating the probability of each output. Convolutional neural networks are specialised in processing data that are known to have a grid-like topology, perfect for image data - a two-dimensional grid of pixels. What makes it different from other deep neural networks is the inclusion of a mathematical operation called convolution in at least one of the layers (Goodfellow et al., 2016).

2.6.1 Components

In order to build an image recognition system, it will require:

- A model - this is what will be trained for the system to work. The layers of the model can be adjusted and will determine the accuracy of the overall system.
- Training dataset - this set will contain annotated images relating to the subject. This includes whether the subject is in the image or not, as well as its position. Annotations may also contain information like shape, features or colour of the subject, all depending on the subject and the defining features you would like to train the model on.
- Testing dataset - after training, this set will be used to determine the accuracy of the model. If it is not desirable, then the model may be adjusted and trained further until it is - a new training set may be required as the model would have learnt the set already.

2.6.2 Types of Image Recognition

Image recognition systems have branched into many fields allowing it to adapt to specific situations. One example of this is object detection - a system trained to scan for a specific condition. This technology is used in medical fields when finding abnormalities in cells or tissues. It works on the fact that objects in their own class will have unique features that help classify them - for example, all balls are round - but if the ball were to have a sharp object sticking out of it, the anomaly can be detected by the system. The same technology can also be seen in facial recognition as eyes, nose and lips are all features that can be used to classify a face. Identification is another form of classification but takes only a single instance of an object to be recognised. This can be used in the identification of handwritten letters and digits, or vehicles. The flexibility of image recognition allows it to be implemented into many situations given the right training.

2.6.3 Advantages and Disadvantages

Advantages:

- Its adaptability allows it to be applied into many fields
- Its use in autonomous systems - like driving - is very convenient

Disadvantages:

- It is not always accurate as anomalies exist in many classes and will be hard to train a system to detect all of them
- The ability to recognise faces and potentially store that data is a concern for security and privacy

2.7 Real World Applications of Image Recognition

2.7.1 Google Translate

Google Translate (Google, 2011) may be considered the most popular translation service in the market today, with over 500 million users, as of April 2016, and over 100 billion words translated each day (Turovsky, 2016a). It uses Google's own neural machine translation engine to predict the likelihood of a sequence of words using an artificial neural network. This allows it to translate whole sentences at a time, rather than piece by piece, thereby adjusting the translation to be more human-like in terms of grammar (Turovsky, 2016b).

Its mobile app supports 109 languages and is able to translate up to 88 languages with the use of the camera or from a photo in the phone's camera roll. The app's camera translation feature also has a more intuitive design where all three translation features are conveniently located at the bottom of the screen: "Instant" allows you to translate text in real-time and have the foreign text translated right from the camera's screen. "Scan" lets you take a photo of the foreign text and allows you to highlight portions of it to be translated instead. Lastly, "Import" translates from an existing photo in your camera roll (Gu, 2019).

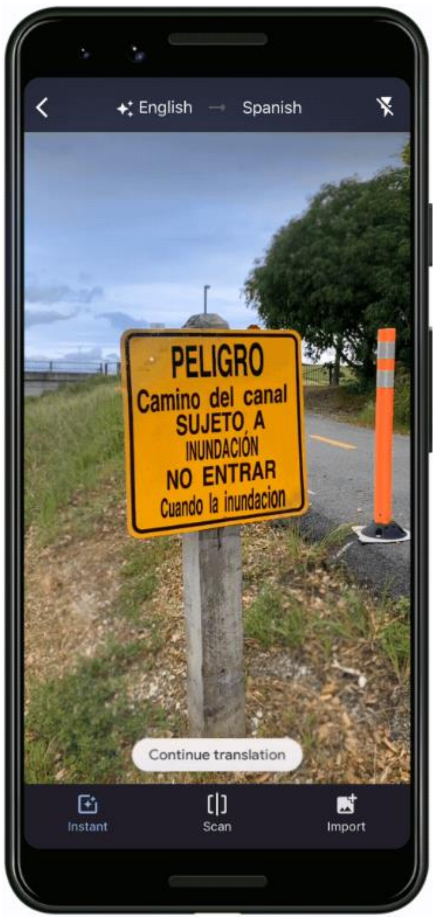


Figure 3. Google Translate in practise (Gu, 2019)

With its use of neural machine translation and the vast amount of resources available to Google, it is hard to find any flaws with its recognition technology. Instead, I will be using this research as an influence when designing my own app which will be discussed further in Chapter 6.

2.7.2 Credit Card Scanning on the iOS

In the iPhone's operating system, is a built-in credit card scanner that allows you to quickly input important card details without the need of using the UI's keyboard. As this technology is built into the OS, Apple has allowed app developers to implement it into their own applications. The scanner uses edge detection to mark the frame of the credit card and is trained to scan for certain numeric patterns like the 16-digit card number or the 4-digit expiry date. I would like my app to process data similar to this technology as it is efficient.

2.7.3 Sudoku Solver: Hint or Solve

Sudoku Solver: Hint or Solve (Anderson, 2011) is a sudoku solving app that allows the user to input numbers into the grid then choose to solve the puzzle by either revealing a single square at a time or revealing the completed sudoku. Alternatively, you are able to use your camera or access your camera roll and take a photo of a sudoku, where it will then be analysed for numbers on the grid and inputted accordingly.

The overall concept of the app is similar to the application I intend to build. However, from a design perspective, I found the app to be lackluster. I found the UI to be bland and unintuitive - the ability to input numbers manually is clunky and if I wanted to solve a single number multiple times, I needed to press “Solve” then “Solve One” every time. I was also not able to find certain features easily, like the clear board button, which would not be available to me until I attempted to solve the current board I had displayed. The top of the app is mostly ad space, which is covering a lot of the screen when it should be utilised in bettering the UI of the app.

2.8 Overview of Approaches

These applications showcased many benefits in their own approaches to image recognition as well as drawbacks to their designs. The intuitive design of the Google Translate app allows for ease of access for the user to quickly choose how they would like to use the app. The sudoku app was also lacking in features and a clear design that I felt would benefit the user for example, easier use of the solve buttons. The ability to understand what works in the market currently and what is missing from their designs is important as it helps produce something greater.

2.9 Evaluation

I found many approaches that will benefit me when considering implementation of algorithms and design of the app. A stochastic search algorithm will be an effective approach to building my solving algorithm but will also build a backtracking algorithm to use as a benchmark when designing the stochastic search algorithm. I will also be using different features from the image recognition applications I researched as they offered a good user experience. Many applications use image recognition systems as a means of convenience, to allow the user to easily input data through an image without having to input it themselves. However, some apps will only use the image to input the data and then the user will resume control. Whereas I intend to recognise the data from the image then quickly solve the puzzle, so it can move onto the next stage of the application autonomously which is choosing how the sudoku will be solved. Overall, I have used a wide and relevant range of sources which will provide me with a good foundation prior to building my application.

3 Primary Research

3.1 Survey

I developed a survey that would help me understand what users want from this type of app. This information would be particularly helpful when designing the app and considering the UX. I chose to carry out a survey of 7 questions via Survey Monkey. This would be the most cost-effective mode of survey research and allowed me to administer and conduct the survey remotely as there was no possibility of meeting respondents due to COVID-19. The third-party system also had the option to collect the results automatically in different formats, e.g. bar charts or tables. Hence, conducting my survey through an online server was the best option for my project.

3.2 Participants

For the purpose of my project, I wanted results from 20-30 people. It was important that the participants had experience playing sudoku, so I chose to share the link to a handful of people that play sudoku on a regular basis. From this, I would ask them to share the link to any other people they knew were familiar with the puzzle so the type of participant would not be biased to completing the survey in my interest. I was able to receive 21 responses, as of July 29 2020, which was plenty of data for me to apply to the design and implementation of my application.

3.3 Data Analysis

Please refer to Appendix C for a full list of the questions and results.

3.3.1 Q1: How often do you play sudoku?

The results show 61.9% of respondents play sudoku either daily or weekly, whereas 28.6% play sudoku monthly or fewer times. As mentioned, I had attempted to circulate the survey to people who play sudoku but two of my respondents had answered they never play sudoku. I imagine some of the responses in the other questions will reflect the answers of these two participants. On the other hand, more than half of the people who participated play sudoku on a regular basis, which is enough motivation and demand for me to continue creating the application.

3.3.2 Q2: What format do you play sudokus on?

Whilst it is evident that 16 people play on their phones, 15 people have opted for the paper formats. This shows me how the preference is relatively similar and those who prefer the mobile counterpart does not critically exceed those who play on paper. Thus, the need of my application may be greater than I initially thought.

3.3.3 Q3: How difficult do you find sudokus?

Only 23.8% of respondents state that they complete every sudoku. Although most people (52.4%) said they complete almost every sudoku, it still opens up the possibility for the need of help in an attempt to increase the percentage of people capable of completing every sudoku, which this application is trying to solve.

3.3.4 Q4: Where are you most likely to use the application?

My data shows me no participant plays on the move, however, a majority like to play on public transport. This is helpful for me in deciding the UX later on in the project as many users would have the option to take a seat on public transport but if they were standing, it is important for me to consider how they would be able to use my app in this situation.

3.3.5 Q5: Which features would you most likely use?

17 of the 21 respondents collectively chose "to provide hints" and "check current inputs are correct." Whereas only 6 participants chose to solve the rest of the sudoku – something that is readily available. Thus, people who do play sudoku on a somewhat regular basis are looking for other options during their game, other than completely solving the puzzle.

3.3.6 Q6: Is this application something that would appeal to you?

Over 80% were interested in the application which shows that this app would be in demand to current players.

3.3.7 Q7: Any features you think you need that are currently missing in similar applications.

Most of the responses were regarding a smaller number of hints or to highlight where an input is wrong. This is something I will try to include in my work. One answer I had not considered yet was being able to keep a photo of the paper-format game stored on the phone so the user can play it at a later time. This is an interesting addition that I would like to implement if there is time.

3.4 Evaluation

The data I retrieved is essential to the remaining project as I was able to receive insight into what users are looking for. Although I am happy with the number of participants, it is always better to get as large of a data set as possible to gain a wider understanding. Secondly, for the question regarding what format they would play on, I did not consider that they may play on multiple platforms or not play at all. Alternatively, I could have reworded the question to be “What is your preferred method of playing sudoku?”

With regards to the two participants who do not play at all, some of my data may be inaccurate. Nevertheless, it was a good idea to complete the survey online and anonymously as I think it helped the respondents answer the questions honestly. Other than these points, I believe the survey was a success as I will be able to implement this data when designing the application which will be demonstrated in Chapter 6.

4 Specifications

4.1 Aims

To achieve a complete and detailed study of the most efficient sudoku solver app, made specifically for paper-formats, the following aims should be accomplished within this project:

- Complete relevant research to select the best methods possible (fulfilled in Chapter 2).
- Using both backtracking and stochastic optimisation to create separate solving algorithms.
- Design the most efficient sudoku solver - with the resources available.
- Implement the use of a camera to recognise a sudoku and solve it on the app.
- Use survey research to design an app with functions that will benefit the user.
- Implement the design of the system within an android-based application.
- Consider the UX design to allow the application to be suitable and accessible for all.
- Carry out tests on potential users and evaluating their experiences to be used when further developing the work.
- Insightful discussion of overall results.
- Evaluate my own work with respect to the original proposal and the final outcome.
- Provide a relevant and wide range of sources.
- Provide a relevant appendix to detail work referenced in the report.

4.1.1 Project Requirements

Prior to implementation, I will require the following files to be created:

- The game, sudoku
  I will need to create my own version of the game in order to create a solver that will work together with the sudoku. This will include:
  - Initialising the grid with a given set of numbers,
  - The ability to insert and remove numbers on the grid,
  - when a number has been added, the game must check if it is valid within its current state,
  - A way to check if the grid has been completed.
- A solving algorithm
  This will be necessary for the app to function. I will need it to perform a full solution of the sudoku given and have it return the values in a clear data structure that will be easy to retrieve.

5 Design

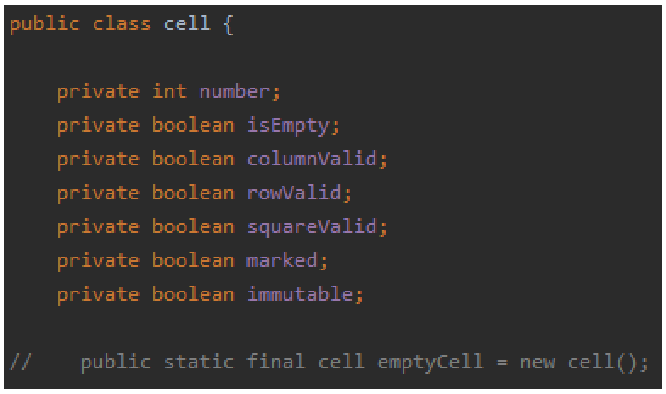
5.1 Change in Circumstance

I will be elaborating on my experience due to the lockdown in Chapter 8. For this reason, this chapter will discuss how I achieved my goals through backtracking as well as how I had intended to design the sudoku through stochastic search algorithms.

5.2 Building the Sudoku

I chose to build my own version of sudoku as it had its own advantages. Breaking it down into several components and functions allowed me to gain a better understanding of the game. It also made writing the solving algorithm easier as I was able to call on certain variables within the sudoku to make a clearer algorithm. The two main components of the sudoku were the individual cells and the sudoku itself that the cells will reside in.

5.2.1 Cells



```java
public class cell {

    private int number;
    private boolean isEmpty;
    private boolean columnValid;
    private boolean rowValid;
    private boolean squareValid;
    private boolean marked;
    private boolean immutable;

//    public static final cell emptyCell = new cell();
```

Figure 4. List of variables initialised in *cell*

To begin, I initialised several variables most of which will allow it to interact with the grid and the other cells. The *number* and *isEmpty* variables are self-explanatory but the -*Valid* boolean variables are used to represent if the cell’s number is valid within the current state of the sudoku. If an invalid number is inserted then that cell’s and any other affected cell’s -*Valid* variables will be adjusted accordingly, i.e. if the number 9 is inserted into a cell but there already existed a number 9 in the same row then both cells containing the number 9 in that row will have their *rowValid* variable changed to false. The *marked* boolean variable works in tandem with the -*Valid* variables - if any of those variables are false then *marked* will be true. This variable is used to quickly check if the cell is invalid at all first before checking which -*Valid* variable is invalid. The last boolean variable is *immutable* and will only be true to the numbers the sudoku started with as they cannot change. I also have a final static variable named *emptyCell* that is just an empty cell but is not used as whenever I called it to change a cell into an empty one, if I was to change its number then any cell following that would also call for *emptyCell* would change into that number which was not intended. I found initialising the cell as *new cell()*, which is an empty cell, worked better and used that instead.

The constructors for the cell will be primarily used when initialising the sudoku. This meant I only needed to construct an empty cell and an immutable cell. All constructed cells at the beginning will have all the *-Valid* variables set to true and once all cells are constructed will the grid check for any invalid numbers and set them accordingly. To construct an empty cell, no parameters are required, and the cell's number is set to 0. *isEmpty* is set to true and will be mutable to allow the user to alter the cell. Next, to construct an immutable cell it will need two parameters: the number being inserted and a boolean that will set the *immutable* variable to ensure the cell's number cannot be changed.

All other functions in the class are getters and setters. The *setNumber* function checks if the number being inserted is between 1 - 9 and will not allow any other number. If a number is inserted then it will change the number, set *isEmpty* to false, and then set all *-Valid* variables to true to be ready for checks. All *set-Valid* functions also work with the *setMarked* function. If any of the *-Valid* functions are set to false then *marked* will be set to true but if they are set to true then it will check if all the other *-Valid* variables are also true and if that is the case, it will set *marked* to false.

5.2.2 The Sudoku

The sudoku only needed two variables of its own to be initialised. The first variable is a 2D array, *grid*, which will be used to house all the cells in the sudoku. The cells are stored in a 2D array to simulate the 9 x 9 format of a sudoku, where the first square bracket calls for the y-position in the grid and the second square bracket calls for the x-position. The second variable is a double, *percentOfZeros*. This variable is used to check how many empty cells are left in the grid. This is useful for when I need to call the *checkIfCompleted* function which will check if the grid is filled and if the entire grid is valid using the *checkIfGridValid* function. *percentOfZeros* is updated when *updatePercentOfZeros* is called which iterates through the *grid* array and if a cell is empty then a counter will increment. A percentage is taken of the number on the counter divided by the number of cells in the grid (81). This function is only called when checking if the sudoku is completed to save time since that is the only time when the *percentOfZeros* variable is used.

In the function *checkIfGridValid*, it uses a nested for loop to iterate through the grid and will check if each cell is valid using *checkIfCellValid* (empty cells are skipped in the loop to save time). When *checkIfCellValid* is called, it takes the y- and x- positions of the cell as parameters and checks if that cell number is the only number in its column, row, and 3 x 3 square. If any one of these conditions have been violated, then the cell and the cell(s) also violating the conditions have their respective *-Valid* variable altered accordingly and the function will then return false. Incidentally, if any cell returns as invalid, then *checkifGridValid* will also return false. *checkIfGridValid* works by calling *checkIfCellValid* multiple times but *checkIfCellValid* can be used independently if only a single cell has been altered.

The sudoku is constructed with one parameter which is a 2D array of integers. This will serve as the sudoku the user wishes to be solved. Like the 2D array *grid*, the 2D array of integers must also be 9 x 9. It will begin to iterate through both *grid* and the integers. If the integer is a 0 then its corresponding position in *grid* is initialised as an empty cell, and if it is a number then the cell is initialised with the number and made immutable. Once all numbers have been inserted then the grid is checked if it is valid using *checkIfGridValid*. If the puzzle was set up correctly then you may begin using the sudoku.

The next vital function of the sudoku is *insertNumber* which takes three integers as its parameters - a number being inserted, a y-position and an x-position. First it will perform checks to see if the number can be inserted: if the cell is mutable, if the number is between 1 - 9 or if the number is not the same as the existing number in the cell. If the number is 0 then it will use the *removeNumber* function which works similarly to *insertNumber* but instead of using cell's *setNumber* function to alter the number, it will just initialise it as a new empty cell. After passing the checks, it will call *unmarkCells* which will unmark cells previously marked. I initially had a problem of unmarking cells that were still invalid i.e. if there were three of the same number in a row and I altered one, all three would be unmarked when only the altered cell should have been unmarked. This was fixed by checking for multiple invalid cells in the column/row/square and if there were more than two then it will not unmark them. Following this step, the new number is inserted and *checkIfCellValid* is called and its boolean is returned.

As the cell numbers are within the object cell, it is inconvenient to call the cell to get the number. Instead, *getCellValues* returns a 2D array of integers containing all the numbers in the grid. I also have a *displaySudoku* function that will contextualise the cell numbers in a sudoku format in the console. *displaySudokuDebug* also prints into the console but instead of the cell number, it will print out the boolean value (as 1 or 0) of the variable I call for in the parameter. This is to help me check if the *-Valid* or *isEmpty* variables are working as intended.

5.3 Writing the Backtracking Algorithm

The solver program will initialise with two variables - *OGPuzzle* which will hold the sudoku provided as a parameter, and *solvedPuzzle* which will store the sudoku that has been successfully solved. *OGPuzzle* will be used when initially beginning the solver. I also had a third variable named *tempPuzzle* to theoretically hold another copy of the puzzle, but this was unnecessary due to the nature of the recursive function. I had completed two versions of the backtracking solver as I had problems with recursion in my first version but was able to fix it in the second version which I will be discussing below.

From my research, I found backtracking can be done as a recursive function. This begins with a base case statement that is needed in order to terminate the function. In this case, I will need to check if the sudoku is completed and if it is, then I can store the solved puzzle in its respective variable and return true. As I am returning a boolean with the recursive function, I will also need it to return false at the bottom of the function in case the algorithm is not able to solve the puzzle given.

The next step would be to call the function recursively. I initially did this by using a for loop to iterate through the cell numbers I am going to insert. I would then insert the number into the sudoku while also using it in an if statement. If the number is invalid, then it will not be inserted and the for loop will increment to try and insert the next number. Once it has inserted successfully then the x- and y-position will increment to the next cell and the *solvePuzzle* function will be called again, this time inserting into the next available cell. This is where I discovered the issue with my first attempt at the algorithm because if it reaches the number 9 and it still cannot be inserted then it will return false and should remove the number at its latest position then backtrack to a previous one to insert the next number. However in my attempt, once it failed at 9 then it would backtrack first then remove the number at the previous position, leaving the number 9 in a cell ahead which in turn meant the algorithm will always fail when it reaches that point. I believe the problems laid in where I was incrementing the position of the cell. Since it was only incrementing when a number was successfully inserted and in the same code block where it would recurse, it did not backtrack the way I intended. This was adjusted in my second attempt at writing the algorithm.

In my second attempt, I kept the base case statement the same as I felt that was going to work as intended. What I did change was where I got the cell's coordinates. I used another nested for loop to iterate through the grid and if the cell were empty, I would store the x- and y-positions then break out of the loop(s). Next, like my first attempt I use a for loop iterating the cell number I wish to insert. However, this time if the number cannot be inserted in a position then it is removed in that same position. This is to ensure the number is removed before backtracking into a different position. If the number is inserted, then the function is called, and the position will be updated in that instance of the function. This version of the algorithm proved successful and was even able to solve sudokus in less than 3 seconds, a time I would have been initially satisfied with if I were building my stochastic search algorithm. I was then able to move onto the next step which was implementing it on the app.

5.3.1 How I would have used a stochastic search algorithm

Stochastic optimisation methods involved many iterations of optimising in order for the algorithm to produce a successful solution. This meant I would have built a stochastic search algorithm along with all the components necessary to optimise it - a search-space as well as a fitness function. In hindsight, this method of solving is a lot more resource heavy than expected and the time it would take to optimise plus the time for it to solve may not produce a result that would make me choose it over the more simple backtracking method. However, without a proper attempt at the approach, I can hardly make a judgement on its capabilities.

6 Design

6.1 Change in Circumstance

I will be elaborating on my experience due to the lockdown in Chapter 8. For this reason, this chapter will discuss how I achieved my goals without the use of image recognition as well as how I intended to use it.

6.2 Designing the App

When designing the app, I decided to go for a simpler design that would be clear for the user and easy to navigate. I was able to achieve that with only a few activity screens: a main activity where the user will input the sudoku they require help with, a solving activity that will solve the sudoku however the user wants and a "howto" activity screen which instructs the user on how to use the app.

6.2.1 Main Screen

For the first activity page of the app, there will be a 9 x 9 grid and several buttons near the bottom of the app. These buttons are: the numbers the user would like to insert (1 - 9), two clear buttons (one to clear one cell and another to clear all cells), a button to solve the sudoku and a button that will take you to the "howto" activity screen. For my app, I provided a default sudoku that can be used to help the user understand how to use the app before attempting to use it for their own leisure.

Each cell on the grid has a *onClick* function attached to it and is linked to the number and clear buttons. If a number or clear button has been clicked (I have also made it so only one number or clear option can be clicked at a time), and a cell has been clicked after, then the function *changeCellNumber* is called. This function will change the cell's text to correspond with the chosen option e.g. if 9 is selected then it will insert 9 into the next cell clicked, or if "clear one" is selected then the cell will become empty. For the cells, number and clear buttons I chose to use *TextView* instead of *Button* because I found them easier to customise and visualise as well as being able to fit many of them into *TableRows* which was what I needed.

When the solve button is clicked, I need a way to convert the *TextView*'s the user has been filling into integers that will be inserted into my sudoku program. This was done by initially storing all the cells into an array and also initialising a 2D array of integers. I would then iterate through the cells and obtain its text value which would be the number. This value is parsed as an integer and stored in the 2D array (if the text value was empty then the integer stored will be 0) in its corresponding position. The 2D array is inserted into the sudoku program which is then inserted into the solver program. It will attempt to solve the sudoku and if successful, the app will move onto the solving activity screen along with the solved sudoku and user inputted sudoku. However, the sudokus cannot be easily transferred to the next activity as *sudoku* is a custom object. To get around this, I will have to only bring the integer values of both sudokus as 2D arrays and serialise them which will allow them to be bundled and transferred.

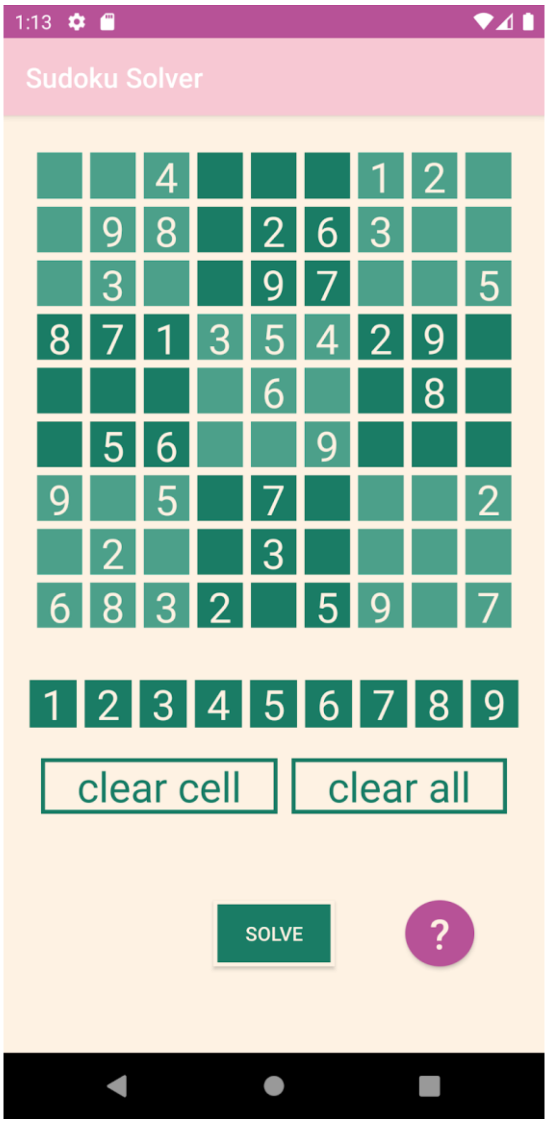


Figure 5. Main activity screen

6.2.2 Solving Screen

Like the main activity screen, there is a 9 x 9 grid but instead, there are buttons that allow the user to solve the sudoku. To begin, I retrieve the data from the previous activity. As the two sudokus were serialised, they are retrieved as only an object class so I parsed them back as integers when moving them into their respective 2D array. The 2D array containing the user inputted sudoku is then inserted into the grid by iterating through and inserting their values in their respective cell.

From my primary research, I chose to use three different ways to solve the sudoku. I felt this was sufficient for what I aimed for when building my app. The first solve button works similarly to the number buttons on the main activity except when a cell is chosen, its index in the *cells* array is used to find its relative position in the solved sudoku. As the *cells* array is one-dimensional and the solved sudoku is a two-dimensional array, I had to divide the index to get a x- and y-position (the modulo for the x-position and the remainder for the y-position). Once its relative position is found then the number in the solved sudoku is placed into the grid to reveal the correct cell to the user.

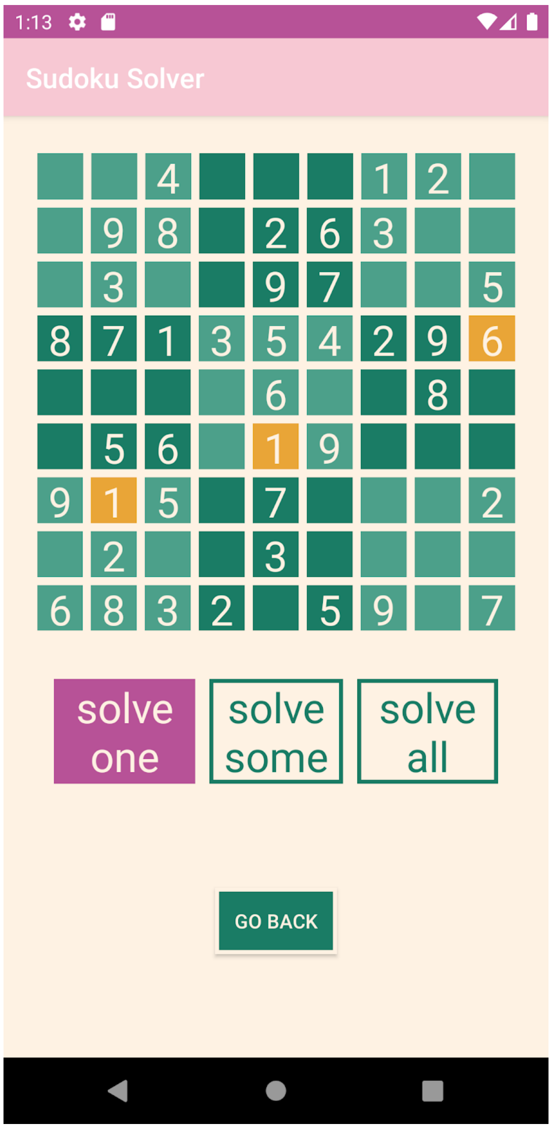


Figure 6. Solving activity screen in practise

The second button solves several randomly chosen cells at once. This is done by choosing a random integer up to 5 (as to not reveal too much at once) and a for loop will iterate that many times. In this for loop, a random cell on the grid is chosen and its number is revealed from the solved sudoku array. If the cell's number is already shown, then another random cell is chosen.

The final solve button reveals the rest of the sudoku for the user. This is done by simply iterating through the cells and if it is empty then the number in the relative position on the solved sudoku is inserted. There is also a button at the very bottom that will allow you to go back to the main activity screen.

6.2.3 How I would have implemented image recognition

I would have created my own image recognition system that utilised edge detection to find the sudoku grid in any photo. I would have also liked to have trained it against many digits of different styles so it can recognise numbers in any font. This system would have been implemented in the main activity screen as another way of inputting the numbers into the cell.

7 Implementation

7.1 White Box Testing

In order to improve on the functionalities of the app, I needed to make sure every part of the program was working as intended. Before implementation of the solver into the app, I conducted several tests on every function. Once I was satisfied with the results there, I performed more tests after implementing them on the app. This was also to check the performance of the app and make sure it was running smoothly. My findings can be found in Appendix D.

7.2 Black Box Testing

This approach to testing allows me to see how my app will be used in the hands of my target audience. For this testing, I asked for two users that were familiar with sudokus but had differing affinities to them. I then measured how often they use the app and what features they used to help themselves. The first user was someone who rarely plays sudoku and finds them difficult to complete, whereas the second user has completed many sudokus of varying difficulty. I felt that this difference in experience was important in understanding what exactly is necessary for my app.

7.2.1 First User

The first user will be given a sudoku of medium difficulty and 10 minutes to solve it. I did not want to make it too difficult to force them to use the app, but I also did not want the users to spend too long to solve it.

Solving method	Times used
Solve one	6
Solve some	2
Solve all	1

Figure 7. Frequencies of features used by the first user

The user stated they had no problems solving the first few numbers but then hit a wall early and decided to use “solve some” to give them clues on how to continue. There were a couple cells that could have been one of two numbers and they decided to use “solve one” on those too. The user then started solving a few more numbers on their own, hit another wall but when they attempted to solve the sudoku with the app, it was invalid. This meant that some of their own inputs were wrong and needed to be removed to try and solve the app again. They removed a few numbers and were able to get into the solving screen again. They used “solve one” a few more times to check the cells they had to remove. In the end, they did not complete the sudoku in time but used “solve all” to check the final result.

The user was pleased with the help the app was able to provide. However, they felt that having to input additional numbers into the solver constantly cut into their own time to solve the sudoku. Nevertheless, they felt the sudoku was not too difficult and only used the app when desperately needed or to confirm some of their own deductions. When asked if they would use the app again, they replied that they would definitely have the app in hand whenever they would play sudoku.

7.2.2 Second User

The second user will be given a harder sudoku than the first user but will still be given 10 minutes to solve it. This was to make sure the sudoku was not too easy for them to ensure that there was a possibility for the user to need the app.

Solving method	Times used
Solve one	3
Solve some	1
Solve all	0

Figure 8. Frequencies of features used by the second user

The user was able to complete the sudoku under the time limit but did use the app a few times. They said they used “solve some” after only inputting 5 numbers. After that, they only used “solve one” whenever they encountered numbers that could be in one of two cells in the same square and were unable to find any other numbers.

They said that the app seemed more targeted to players with less experience as it really helps when you are really unsure of a cell and just need confirmation. They said that it is really helpful as completing a sudoku by any means necessary can motivate people to play more. Even as someone that has completed many sudokus, they stated that while it is not very efficient to play a game of sudoku on the app itself, as it is missing many features that are on common sudoku apps, it is nice to be able to store the numbers in the grid and play it at another time, albeit inefficiently. They said that they may not use the app every time but would have it ready when attempting extremely hard sudokus.

7.3 Evaluation

The tests that I have conducted have been successful in the improvement of my app. I was able to make the necessary adjustments from the results of my white box testing which in turn made the black box testing a much better experience for my users. The results from my black box testing have also demonstrated the need of my application in the market. From the feedback received, I concluded that while the app works as intended, it is missing features that would have made the user experience much better. I will be taking into account the user evaluations and do additional tests with more users when further developing the app.

8 Evaluation

8.1 Evaluation of Project Aims

I believe that I was able to successfully achieve most of my project aims, detailed in Chapter 4.1, successfully. The amount of research I did was extensive, and I was able to examine different techniques within backtracking and stochastic optimisation, to conclusively compare and contrast what would be the ideal algorithm for my app. During the progression of my project, I was deprived of my resources at university and had spent less time than intended as I was responsible for taking care of my eight-year-old nephew during school time and beyond. This made it extremely difficult for me to manage my time against my proposed schedule in the original plan, which meant that I was forced to make some sacrifices during the project.

Firstly, I had intended to use a stochastic optimisation technique for the sudoku solver as it was more efficient than a brute force method. However, time was not on my side and I had decided to opt for the backtracking method which worked fine. Secondly, one of the major features of my app that I proposed in my original plan was to “implement the use of a camera to recognise a sudoku and solve it on the app.” This is something that I was keen on including and was a major part of the ambition of my project, but I did not have the resources to test this technology at home. It was a real shame to not have these elements included in my work, but I still think that the app is successful without it and these features can still be implemented in my further work. Thus, the aim to “design the most efficient sudoku solver - with the resources available” was still achieved.

As per my proposal, I said that I would conduct a survey of 20-30 people which I achieved and had learnt a lot from. Primary research is important to any project being conducted as it allows you to receive more project-specific answers, as well as fill any gaps that secondary research may have not been able to answer. It gave me the chance to have greater control of my needs within the project and what research I implement into the application. Moreover, the data I retrieved was unique to my project alone which allowed for a more meaningful contribution to existing studies on the subject. Through analysing the data, I was able to confidently use it to help me with the UX design when implementing it on an android-based application, so that it was an app that would actually appeal to its users as it is ultimately made for their benefit.

Beyond using the survey, I carried out further primary research by supervising tests on two differently skilled sudoku players and recorded their experiences. Overall, this was positive and they had given me insight into things that could be improved and added which I can certainly look at in the future. All of this was supported with a relevant and wide range of sources and appendices that can be referred to at the end of the report. However, it must be noted the data shown in Appendix C shows later results as more people had completed the survey after my analysis.

Throughout the project, I was able to achieve my project aims through insightful discussion and evaluation at every stage, to ensure that I was able to gain the most research before designing and implementing the app so that it could be as successful as possible, considering the resources, time available and circumstances due to the pandemic.

8.2 Evaluation of Project Requirements

In regard to the project requirements listed in Chapter 4.1.1, I believe I was able to successfully build the files necessary for the app to function. I discovered a major bug late in my development of the app which I was unable to find until I did more white box testing. This bug prevented some cells from being validated when they needed to be because they had already been marked by a previous cell. This was fixed by simply not checking if the cell was marked, which I initially put there in hopes to save time. Now the sudoku will check even marked cells and has not run into the same bug since. I was very satisfied with the final build of the sudoku as I was able to build all the components as well as allow almost every part of the sudoku to be accessed. This was important for future development because if I was to use another solving method, I would have the ability to access the sudoku down to its components.

The backtracking algorithm was another success which I did not find too difficult to design. As mentioned in Chapter 5.3, I had problems with iterating through the sudoku in my first attempt at writing the algorithm but was able to fix it in an efficient manner. While I was not able to design a stochastic algorithm, as intended at the beginning of my project, I was extremely satisfied with the results of the backtracking algorithm and was able to easily implement it into my app. Future development of the app may include the use of another solving algorithm and if I was to do that, then updating the app would be easy.

8.3 Evaluation of Application

Overall, I am satisfied with the app as a whole. While I was not able to add all the features I intended to at the beginning of the project, the app I was able to produce was still able to achieve its primary goal which was to help the user solve a sudoku, as proven in my black box testing. This was also my first attempt at creating an app and while it did take some time to learn, I found it easy to navigate around the IDE as Android Studio is based on IntelliJ, my chosen IDE for Java. I was not completely comfortable with some features like fragments, but I was able to achieve a working app without it. I will have to consider implementing those features in the future as an attempt to optimise the performance of the app.

The layout of the app is very pleasing with its clear and minimalist design making it easy to navigate while also having a lot of room for additional features in the future. The shapes used were quite basic as I did not try to experiment too much with the format, but this worked in my favour as I was able to make the layout simple in style.

In terms of the backend, I was not entirely happy with the layout of the cells. While it did get the job done in the end, having 81 identical *TextViews* as cells did not seem very efficient. It also made modifying them incredibly time-consuming as I could not find a way to do it all at once. In the future, I would like to find a way to optimise that part of the app as it may be the most resource consuming.

8.4 Further Work

As demonstrated through the user testing, the app remains successful despite missing out on major features that were originally proposed. However, this means that when I develop this work further, I will be able to create something that is more invaluable to its users.

The first thing I would do is to attempt to write the sudoku solver with the stochastic optimisation method to test if it would be able to solve the sudoku even faster. From my research, this should be the case but having both options available would allow me to test it myself and see if the change of method would be significant.

Secondly, I would like to add the camera aspect to the sudoku solver as mentioned as I think it would really amplify the demand for the app. Not only would this be a helpful option for players, the additional gimmick of seeing the numbers being recognised on the phone could make it more appealing for users. If I were to go ahead with this, I may have to conduct more research on the subject as this type of technology is fast-developing and I would want to ensure that the resources that I use are up-to-date.

At the end of my survey, I had asked if the users were looking for any features in particular and someone said “take a photo of the sudoku I’m playing and continue it on the app. Sometimes I won’t have time to finish it on my commute but would like to resume it at a later time.” I thought that would be a really nice addition to the app, but it would require me to do more research and give more thought to the UX.

With regards to my further work, I have a lot of pointers that I can work with to create the second edition which would offer more features and could potentially improve the current app by making it more user-friendly and faster. This would be completed alongside more user testing.

8.5 Self-Evaluation

Throughout this process, I have developed invaluable knowledge about search algorithms and how to implement a design into an android-based application. Although I was not able to achieve everything that I set out to do, I am happy with the outcome as it taught me the importance of time-management and how I can adapt my work in a positive way to suit my needs. The project was also somewhat collaborative with the number of participants and testers involved so it helped me develop my communication and deductive reasoning skills when analysing the results. Having failed to attempt this project last year, I am pleased with the outcome as I was still able to produce my application within the circumstances of the pandemic. I have thoroughly enjoyed this project and look forward to developing the application further in the hopes that I could publish it for public use.

9 Conclusion

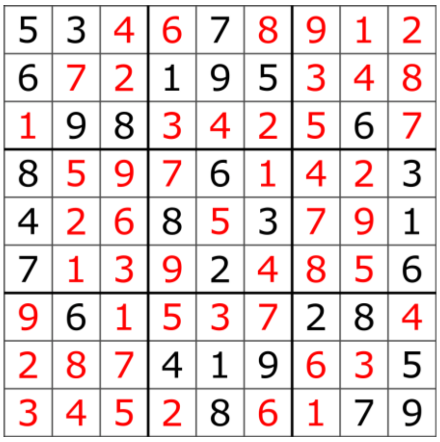
The rise of sudoku and other analog based games continues to rise as the global pandemic forces us to stay home (Williangham, 2020). With that in mind, the need for an application like the Sudoku Solver app may see a great demand during this time.

The two main aims of this project was to create a sudoku solver for paper-format users and implement a camera that could scan these sudokus. This project currently fulfills one of these aims very well and at a functioning level after being tested against multiple users. Due to the lack of resources and not being able to complete this aim, the project tries to satisfy this void through extensive research that is inclusive of a survey which has now been completed by 24 participants.

As the application currently stands, it is able to solve the problem that sudoku solvers experience when dealing with traditional paper sudokus. Some newspapers only supply the answers days later and this application provides a solution for those who are looking to complete the puzzle now. This application aids players of paper-format sudokus to be at an even better position than this application had intended by offering more than just solving the game, it allows users to customise how many hints they are looking for which mobile sudokus do not currently offer.

In conclusion, the application can be deemed a success as it is able to solve the problems recognised at the start of the project without fulfilling every single aim. For this reason, with the developments and improvements as mentioned in my further work, I can only see more potential in its demand when I am able to implement all features that were in the original framework and those that were commented on through the survey and testing.

By providing this access to sudoku players, it gives the opportunity for paper-format sudokus to thrive and possibly see a similar craze to its height in 2005. After all, there are 6,670,903,752,021,072,936,960 (~10^{21}) possible sudoku grids (Kiersz, 2019) and I am hopeful that all sudoku players will enjoy my application and open up their interest into paper-format sudokus.



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

Figure 9. Diagram of a solved sudoku

Bibliography

Barták, R., 1998. *Systematic Search Algorithms*. [online] ONLINE GUIDE TO CONSTRAINT PROGRAMMING. Available at: <https://ktiml.mff.cuni.cz/~bartak/constraints/backtrack.html> [Accessed 10 August 2020].

Cantú-Paz, E. and Goldberg, D., 2000. *Efficient Parallel Genetic Algorithms: Theory And Practice*. [online] Academia.edu. Available at: <https://www.academia.edu/9396103/Efficient_Parallel_Genetic_Algorithms_Theory_and_Practice> [Accessed 9 August 2020].

Chaturvedi, A., 2018. *Backtracking | Introduction - Geeksforgeeks*. [online] GeeksforGeeks. Available at: <https://www.geeksforgeeks.org/backtracking-introduction/> [Accessed 8 August 2020].

Daniel Anderson. 2011. *Sudoku Solver: Hint or Solve (version 5.0).* [Mobile App]. Available at: <https://apps.apple.com/us/app/sudoku-solver-hint-or-solve/id434333494> [Accessed 12 August 2020]

Dantzig, G. B., and J. H. Ramser. "The Truck Dispatching Problem." *Management Science*, vol. 6, no. 1, 1959, pp. 80–91. [book] JSTOR. Available at: <www.jstor.org/stable/2627477> [Accessed 10 August 2020]

Eberhart, R., Shi, Y. and Kennedy, J., 2001. *Swarm Intelligence*. [ebook] Amsterdam: Elsevier. Available at: <https://doc.lagout.org/science/Artificial%20Intelligence/Swarm%20Intelligence/Swarm%20intelligence%20-%20James%20Kennedy.pdf> [Accessed 10 August 2020].

GeeksforGeeks. 2020. *Searching Algorithms - Geeksforgeeks*. [online] Available at: <https://www.geeksforgeeks.org/searching-algorithms/> [Accessed 10 August 2020].

Goodfellow, I., Bengio, Y. and Courville, A., 2016. *Deep Learning*. Massachusetts: MIT Press.
Google. 2011. *Google Translate (version 6.11).* [Mobile App]. Available at: <https://apps.apple.com/gb/app/google-translate/id414706506> [Accessed 12 August 2020]

Gu, X., 2019. Google Translate’s instant camera translation gets an upgrade. [Blog] *Google The Keyword*, Available at: <https://blog.google/products/translate/google-translates-instant-camera-translation-gets-upgrade/> [Accessed 12 August 2020].

Guo, P., 2020. *6.3. The Sequential Search — Problem Solving With Algorithms And Data Structures*. [online] Runestone Academy. Available at: <https://runestone.academy/runestone/books/published/pythonds/SortSearch/TheSequentialSearch.html> [Accessed 10 August 2020].

Huang, T.S., 1996. *Computer Vision: Evolution and Promise.* [online] CERN School of Computing. no. 19 pp. 21-25. Available at: <http://cds.cern.ch/record/400313/files/p21.pdf.> [Accessed 9 August 2020].

Kiersz, A., 2019. *The Number Of Solutions You Could Fit Into Your Weekend Sudoku Is Mind-Boggling*. [online] Business Insider. Available at: <https://www.businessinsider.com/number-of-possible-sudoku-puzzle-grids-2019-4?r=US&IR=T> [Accessed 10 August 2020].

Kiersz, A., 2020. *The Number Of Solutions You Could Fit Into Your Weekend Sudoku Is Mind-Boggling*. [online] Business Insider. Available at: <https://www.businessinsider.com/number-of-possible-sudoku-puzzle-grids-2019-4?r=US&IR=T> [Accessed 10 August 2020].

Kirkpatrick, S., Gelatt, Jr., C. and Vecchi, M., 1983. *Optimization by Simulated Annealing. Science*, [online] 220(4598), pp. 671-679. Available at: <http://wexler.free.fr/library/files/kirkpatrick%20(1983)%20optimization%20by%20simulated%20annealing.pdf> [Accessed 10 August 2020].

Klette, R., 2014. *Concise Computer Vision: An Introduction Into Theory And Algorithms*. Berlin: Springer Science & Business Media.

Knuth, D., 2011. *Fundamental Algorithms*. Upper Saddle, N.J: Addison-Wesley.

Lee, B., 2020. *Level Up Your Brain Power With Sudoku*. [online] Lifehack. Available at: <https://www.lifehack.org/articles/lifestyle/level-up-your-brain-power-with-sudoku.html> [Accessed 10 August 2020].

Miller, B. and Ranum, D., 2011. *Problem Solving With Algorithms And Data Structures Using Python*. 2nd ed. Portland: Franklin, Beedle & Associates.

Moraglio, A. and Togelius, J., 2007. *Geometric Particle Swarm Optimization For Sudoku Puzzles*. [online] Julian.togelius.com. Available at: <http://julian.togelius.com/Moraglio2007Geometric.pdf> [Accessed 9 August 2020].

More, A., 2020. *Jigsaw Puzzle Market 2020 Global Industry Brief Analysis By Top Countries Data With Market Size Is Expected To See Growth Of 730 Million USD Till 2024*. [online] MarketWatch. Available at: <https://www.marketwatch.com/press-release/jigsaw-puzzle-market-2020-global-industry-brief-analysis-by-top-countries-data-with-market-size-is-expected-to-see-growth-of-730-million-usd-till-2024-2020-07-02> [Accessed 10 August 2020].

Munoz, L., 2020. *Sudoku Solver Application Research.* [online survey] Survey Monkey. Available at: <https://www.surveymonkey.co.uk/r/YSWSF5J> [Accessed 10 August 2020].

Perez, M. and Marwala, T., 2008. *STOCHASTIC OPTIMIZATION APPROACHES FOR SOLVING SUDOKU*. [online] Arxiv.org. Available at: <https://arxiv.org/ftp/arxiv/papers/0805/0805.0697.pdf> [Accessed 9 August 2020].

Poole, D. and Mackworth, A., 2018. *Artificial Intelligence*. 2nd ed. Cambridge: Cambridge University Press.

Schrijver, A., 2003. *Combinatorial Optimization: Polyhedra And Efficiency, Volume 1*. Berlin: Springer Science & Business Media.

Spall, J., 2005. *Introduction To Stochastic Search And Optimization*. Hoboken: Wiley.

Surveymonkey.com. 2020. *Surveymonkey: The World'S Most Popular Free Online Survey Tool*. [online] Available at: <https://www.surveymonkey.com/> [Accessed 9 August 2020].

Szeliski, R., 2010. *Computer Vision: Algorithms and Applications*. Berlin: Springer Science & Business Media.

The Economist. 2005. *Do You Sudoku?*. [online] The Economist Publishing Group. Available at: <https://www.economist.com/business/2005/05/19/do-you-sudoku> [Accessed 10 August 2020].

Tsang, E., 2014. *Foundations Of Constraint Satisfaction: The Classic Text*. Norderstedt: BoD – Books on Demand.

Turovsky, B., 2016a. Ten years of Google Translate. [Blog] *Google The Keyword*, Available at: <https://www.blog.google/products/translate/ten-years-of-google-translate/> [Accessed 12 August 2020].

Turovsky, B., 2016b. Found in translation: More accurate, fluent sentences in Google Translate. [Blog] *Google The Keyword*, Available at: <https://blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate/> [Accessed 12 August 2020].

Verenich, Ilya & Nguyen, Hoang & La Rosa, Marcello & Dumas, Marlon. 2017. White-Box Prediction of Process Performance Indicators via Flow Analysis - Scientific Figure on ResearchGate. [Diagram] *ResearchGate,* 10.1145/3084100.3084110. Available at: <https://www.researchgate.net/figure/Backtracking-algorithm-taken-from-1_fig2_316610194> [Accessed 12 August 2020]

Wayback Machine. 2007. *Chapter 19 Backtracking Algorithms*. [online] Available at: <https://web.archive.org/web/20070317015632/http://www.cse.ohio-state.edu/~gurari/course/cis680/cis680Ch19.html#QQ1-51-128> [Accessed 10 August 2020].

Willingham, A., 2020. *People Are Curbing Their Stay-At-Home Anxiety The Analog Way: With Puzzles*. [online] CNN. Available at: <https://edition.cnn.com/2020/03/21/world/puzzles-self-quarantine-coronavirus-psychology-wellness-trnd/index.html> [Accessed 10 August 2020].

List of Figures

Appendix A: Github

https://github.com/lendlm/Year-3-Software-Project

Appendix B: Blog

https://lmsudokusolver.blogspot.com/

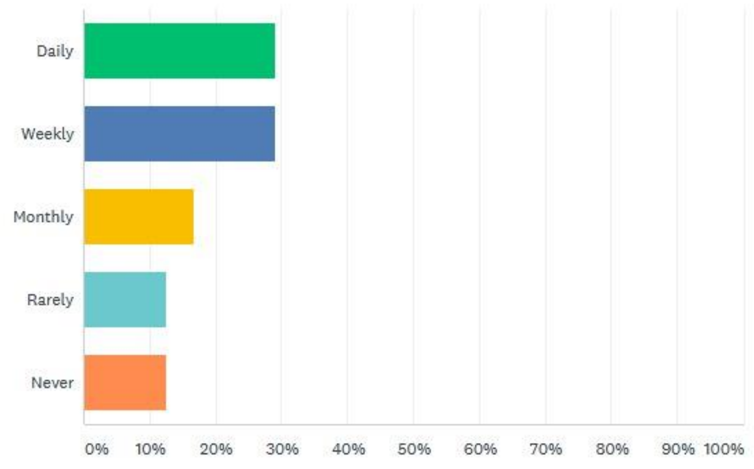
Appendix C: Survey Results

These are the latest results that were taken on August 15 2020 at 9:42am, and are different to the results discussed in the report which had analysed 21 responses, instead of the current 24 presented in the following data.

Q1

How often do you play Sudoku?

Answered: 24 Skipped: 0



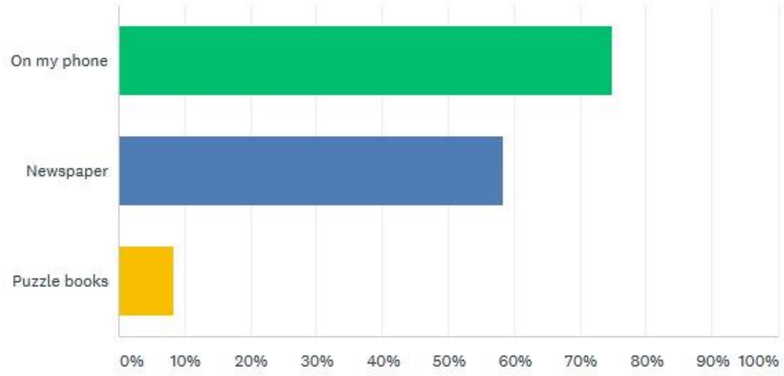
ANSWER CHOICES	RESPONSES	
Daily	29.17%	7
Weekly	29.17%	7
Monthly	16.67%	4
Rarely	12.50%	3
Never	12.50%	3
TOTAL		24

Q2

Customize Save as

What format do you play Sudoku's on?

Answered: 24 Skipped: 0



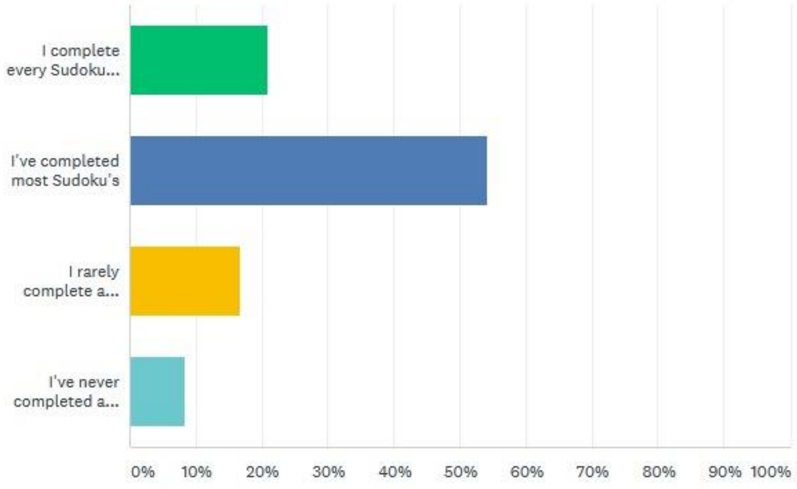
ANSWER CHOICES	RESPONSES	
On my phone	75.00%	18
Newspaper	58.33%	14
Puzzle books	8.33%	2
Total Respondents: 24		

Q3

Customize Save as

How difficult do you find Sudoku's?

Answered: 24 Skipped: 0



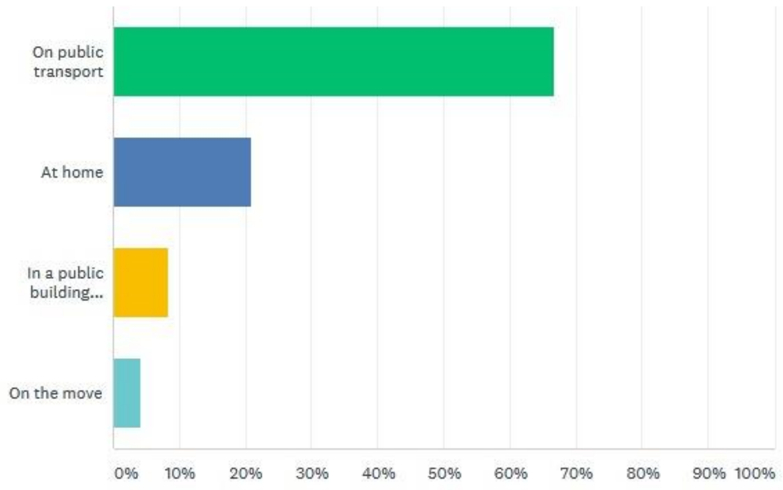
ANSWER CHOICES	RESPONSES	
I complete every Sudoku I've done	20.83%	5
I've completed most Sudoku's	54.17%	13
I rarely complete a Sudoku	16.67%	4
I've never completed a Sudoku	8.33%	2
TOTAL		**24**

Q4

Customize Save as

Where are you most likely to use the application?

Answered: 24 Skipped: 0



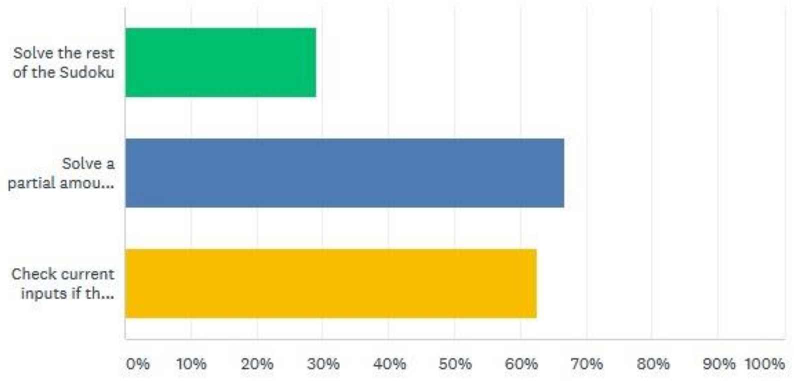
ANSWER CHOICES	RESPONSES	
On public transport	66.67%	16
At home	20.83%	5
In a public building (library, cafe, lecture hall, etc.)	8.33%	2
On the move	4.17%	1
TOTAL		24

Q5

Customize Save as

Which features would you most likely use?

Answered: 24 Skipped: 0



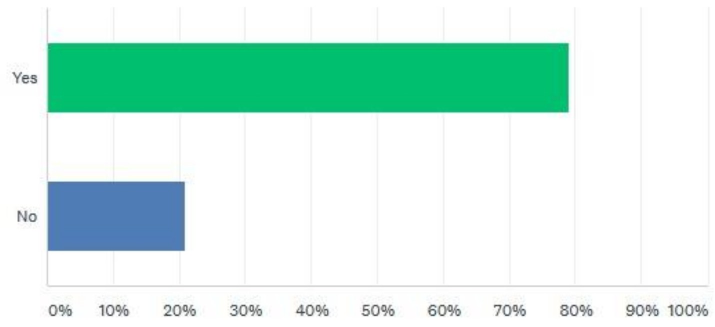
ANSWER CHOICES	RESPONSES	
Solve the rest of the Sudoku	29.17%	7
Solve a partial amount of the Sudoku (provide hints)	66.67%	16
Check current inputs if they are correct	62.50%	15
Total Respondents: 24		

Q6

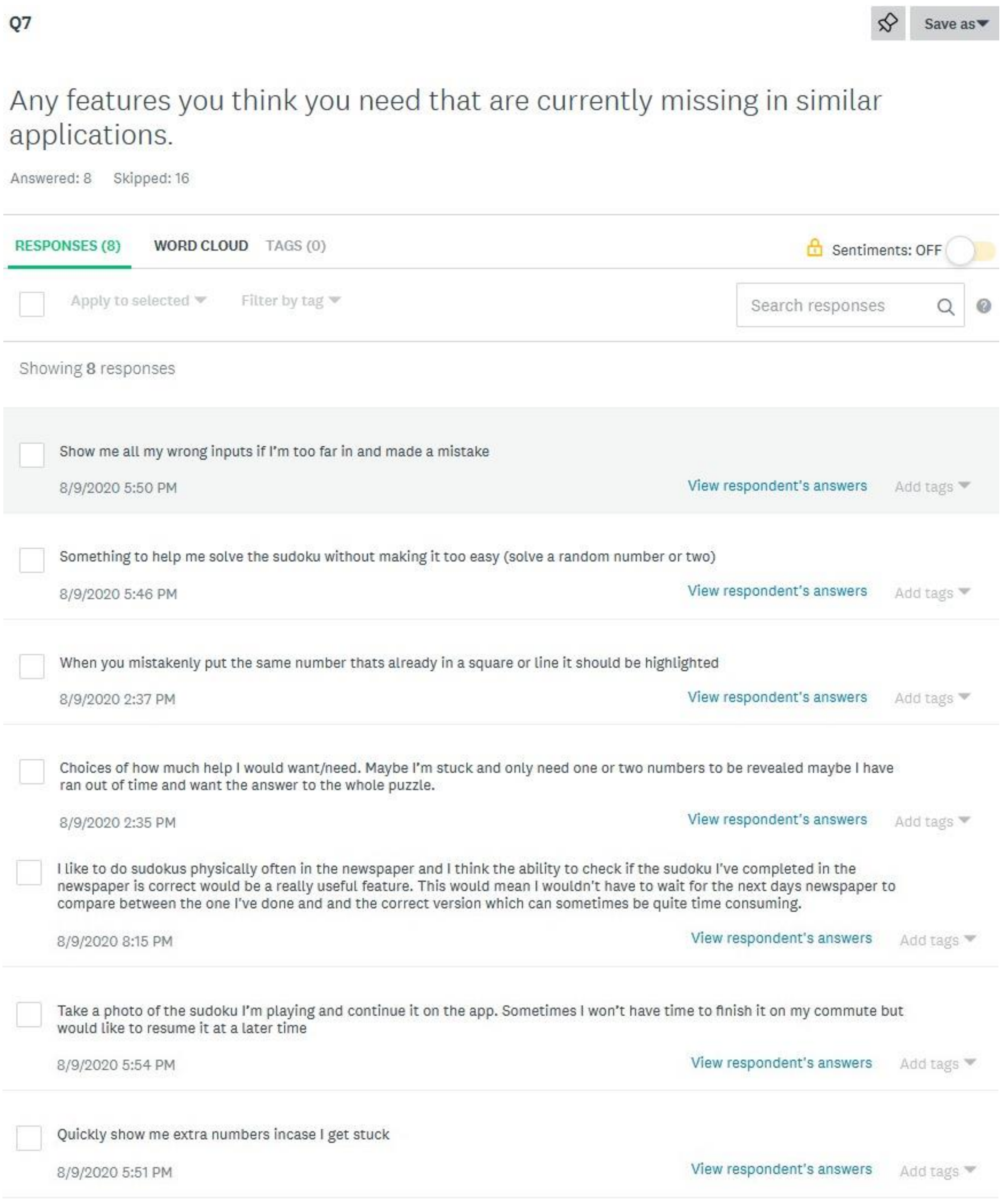
Customize Save as

Is this application something that would appeal to you?

Answered: 24 Skipped: 0



ANSWER CHOICES	RESPONSES	
Yes	79.17%	19
No	20.83%	5
TOTAL		24


Q7

Save as

Any features you think you need that are currently missing in similar applications.

Answered: 8 Skipped: 16

RESPONSES (8) WORD CLOUD TAGS (0) Sentiments: OFF

Apply to selected Filter by tag Search responses

Showing 8 responses

Show me all my wrong inputs if I'm too far in and made a mistake

8/9/2020 5:50 PM View respondent's answers Add tags

Something to help me solve the sudoku without making it too easy (solve a random number or two)

8/9/2020 5:46 PM View respondent's answers Add tags

When you mistakenly put the same number thats already in a square or line it should be highlighted

8/9/2020 2:37 PM View respondent's answers Add tags

Choices of how much help I would want/need. Maybe I'm stuck and only need one or two numbers to be revealed maybe I have ran out of time and want the answer to the whole puzzle.

8/9/2020 2:35 PM View respondent's answers Add tags

I like to do sudokus physically often in the newspaper and I think the ability to check if the sudoku I've completed in the newspaper is correct would be a really useful feature. This would mean I wouldn't have to wait for the next days newspaper to compare between the one I've done and and the correct version which can sometimes be quite time consuming.

8/9/2020 8:15 PM View respondent's answers Add tags

Take a photo of the sudoku I'm playing and continue it on the app. Sometimes I won't have time to finish it on my commute but would like to resume it at a later time

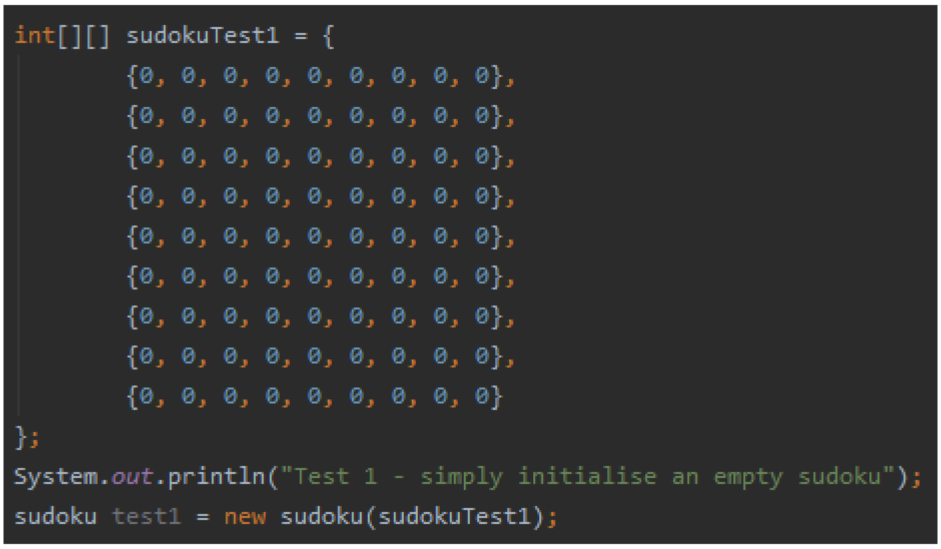
8/9/2020 5:54 PM View respondent's answers Add tags

Quickly show me extra numbers incase I get stuck

8/9/2020 5:51 PM View respondent's answers Add tags

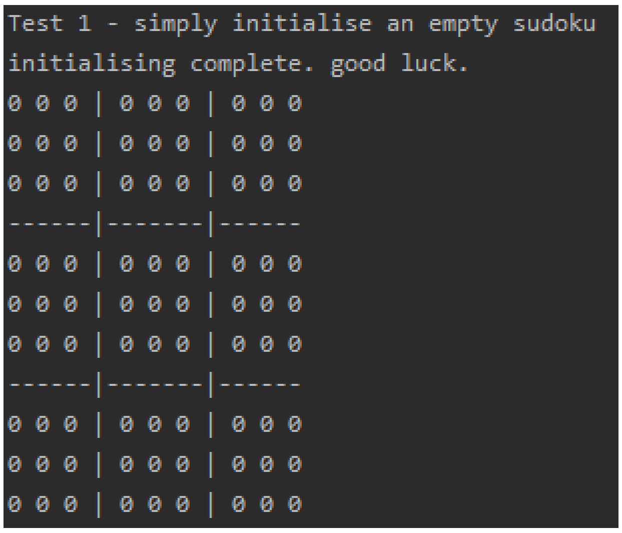
Appendix D: White Box Testing

Test 1



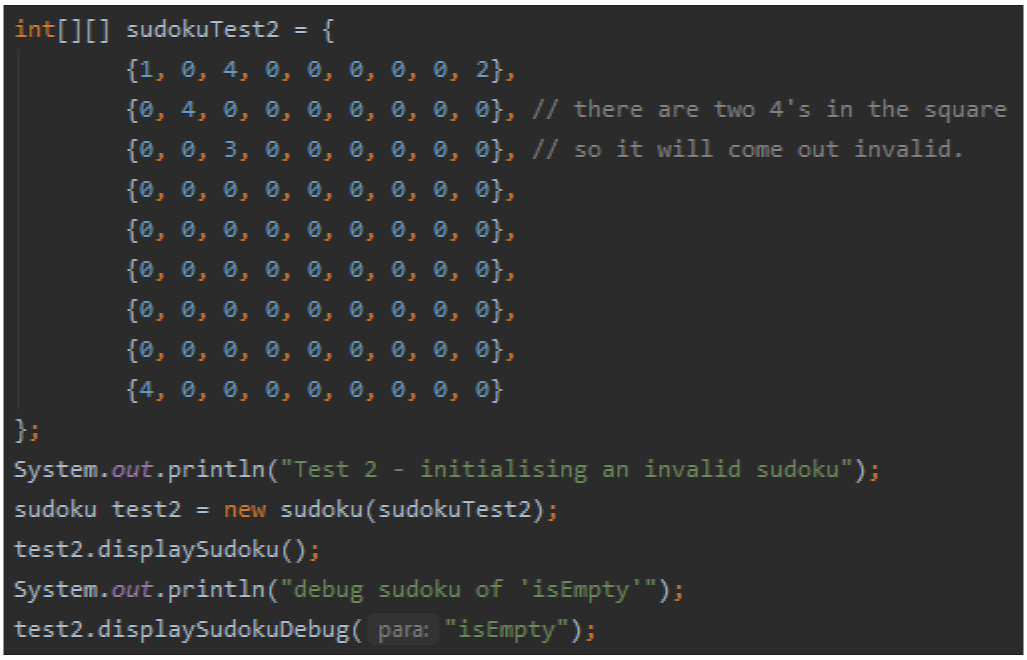
```
int[][] sudokuTest1 = {
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0}
};
System.out.println("Test 1 - simply initialise an empty sudoku");
sudoku test1 = new sudoku(sudokuTest1);
```

Results of Test 1



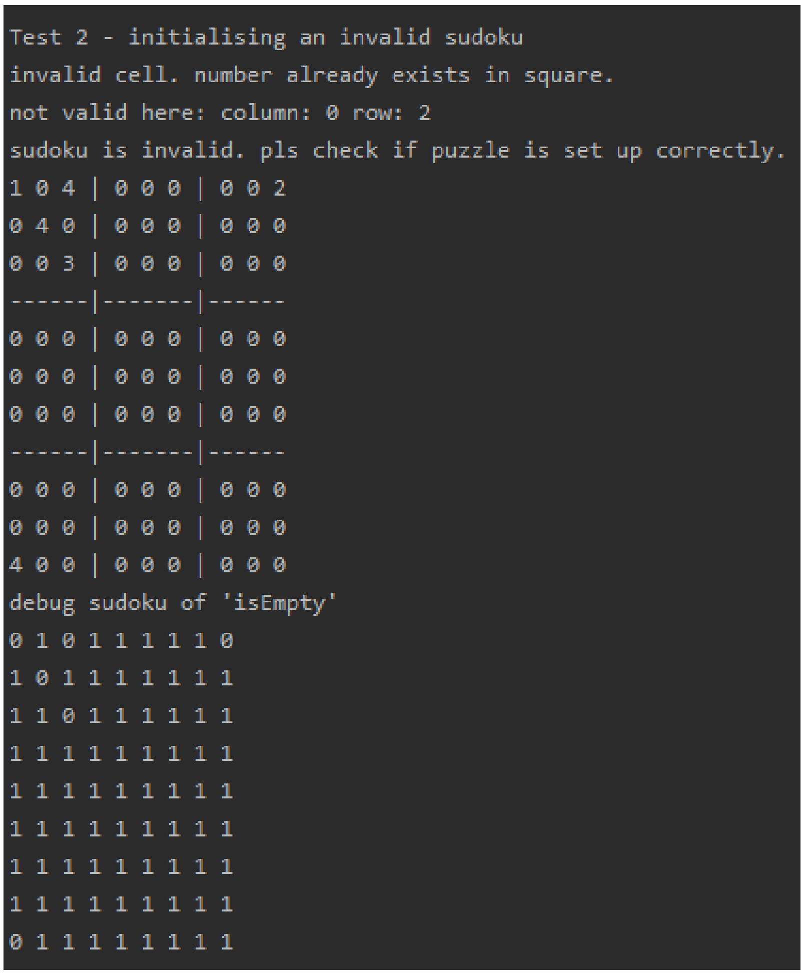
```
Test 1 - simply initialise an empty sudoku
initialising complete. good luck.
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
```

Test 2



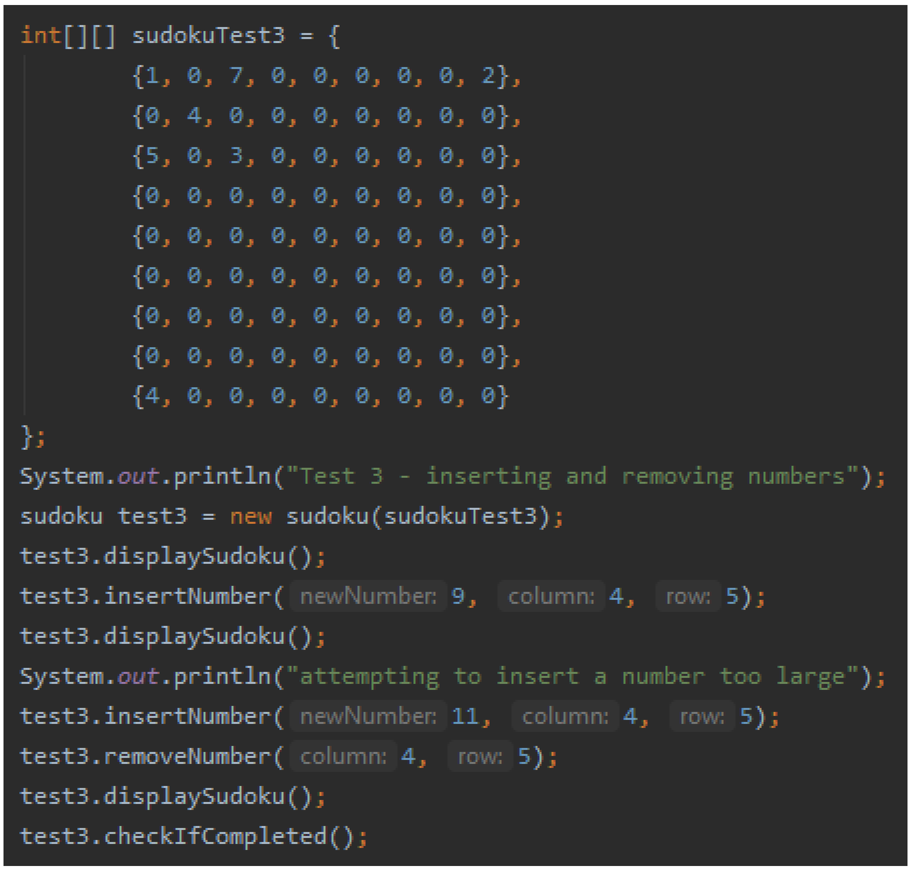
```java
int[][] sudokuTest2 = {
        {1, 0, 4, 0, 0, 0, 0, 0, 2},
        {0, 4, 0, 0, 0, 0, 0, 0, 0}, // there are two 4's in the square
        {0, 0, 3, 0, 0, 0, 0, 0, 0}, // so it will come out invalid.
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {4, 0, 0, 0, 0, 0, 0, 0, 0}
};
System.out.println("Test 2 - initialising an invalid sudoku");
sudoku test2 = new sudoku(sudokuTest2);
test2.displaySudoku();
System.out.println("debug sudoku of 'isEmpty'");
test2.displaySudokuDebug( para: "isEmpty");
```

Results of Test 2

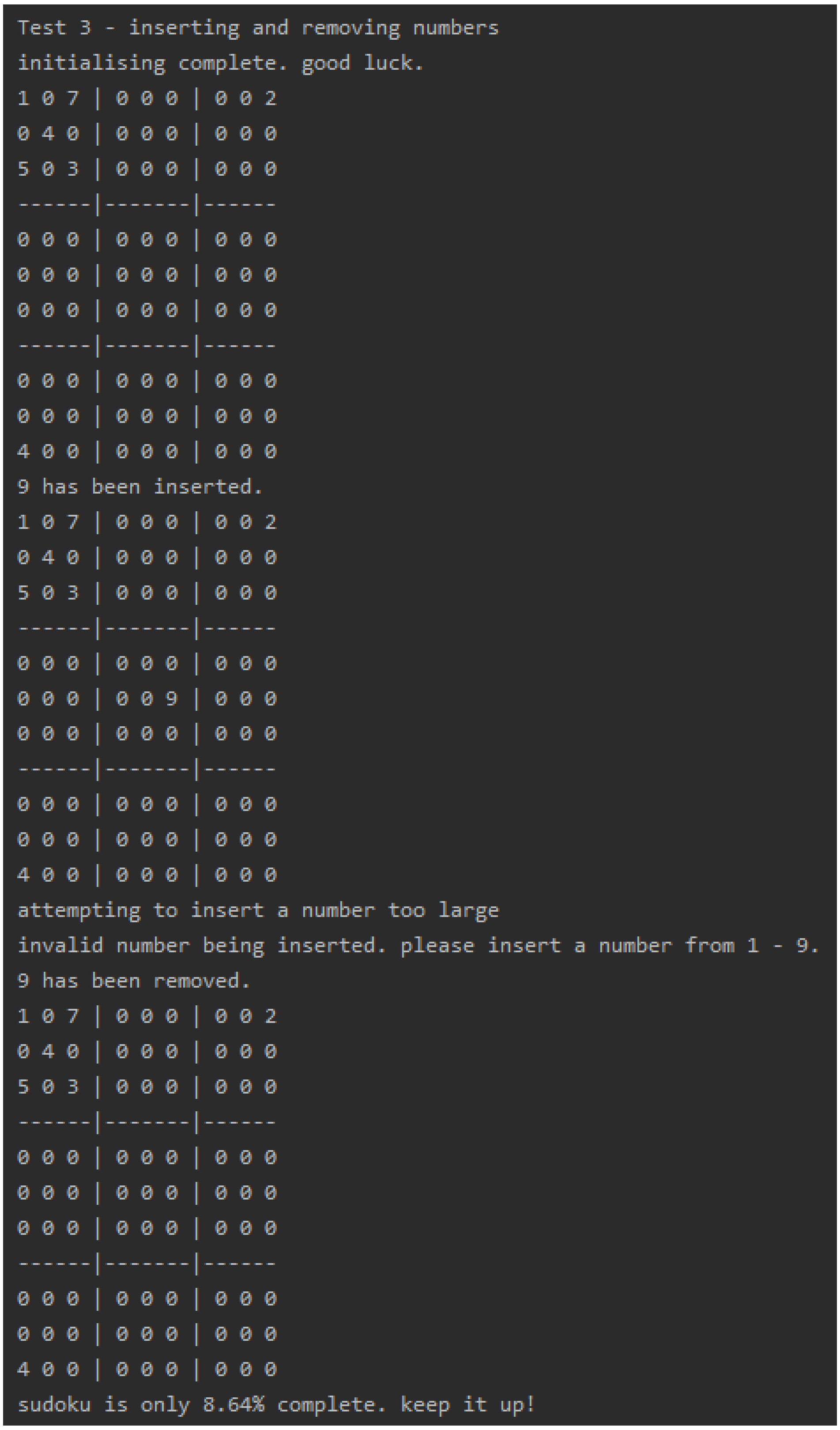


```
Test 2 - initialising an invalid sudoku
invalid cell. number already exists in square.
not valid here: column: 0 row: 2
sudoku is invalid. pls check if puzzle is set up correctly.
1 0 4 | 0 0 0 | 0 0 2
0 4 0 | 0 0 0 | 0 0 0
0 0 3 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
4 0 0 | 0 0 0 | 0 0 0
debug sudoku of 'isEmpty'
0 1 0 1 1 1 1 1 0
1 0 1 1 1 1 1 1 1
1 1 0 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1
```

Test 3

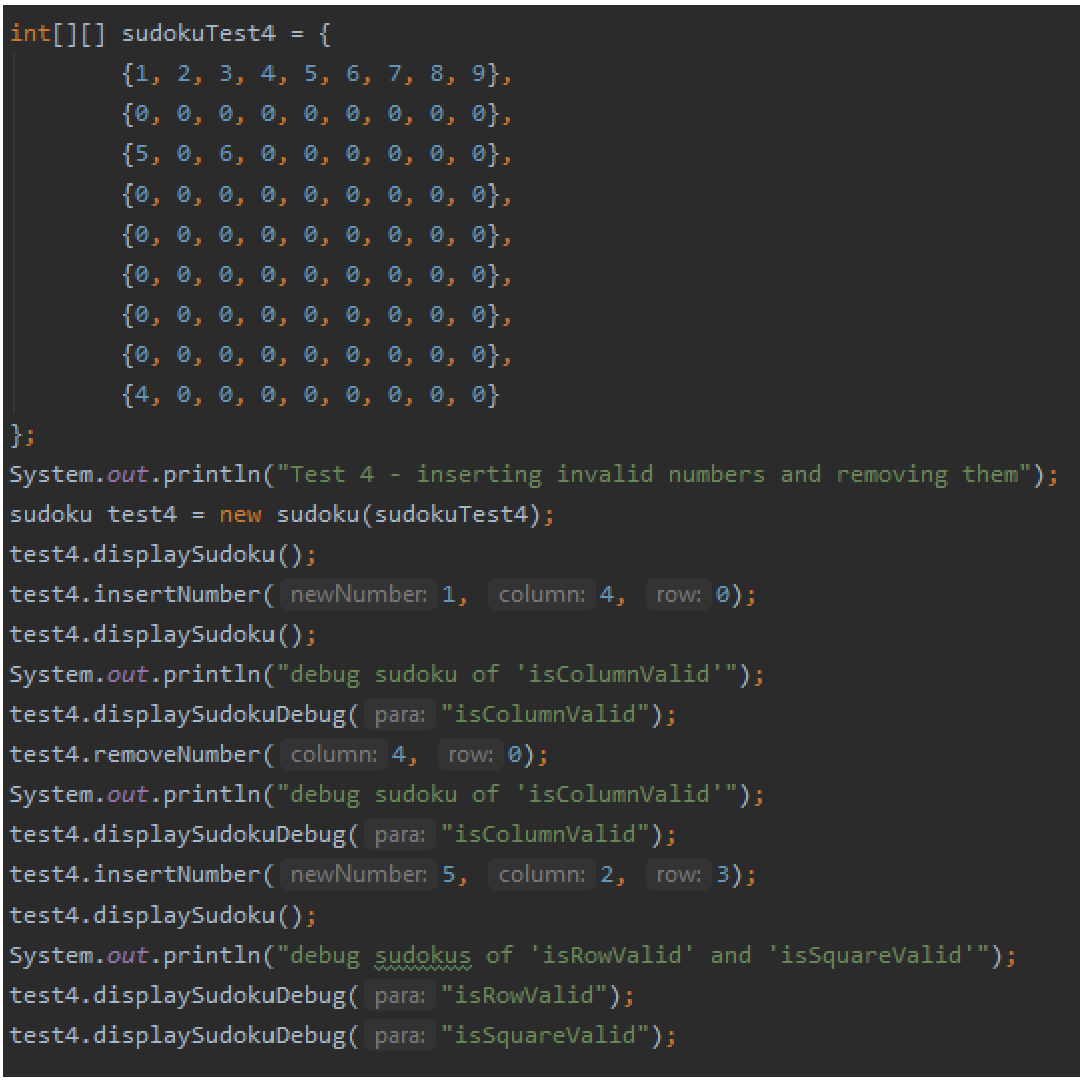

```
int[][] sudokuTest3 = {
        {1, 0, 7, 0, 0, 0, 0, 0, 2},
        {0, 4, 0, 0, 0, 0, 0, 0, 0},
        {5, 0, 3, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {4, 0, 0, 0, 0, 0, 0, 0, 0}
};
System.out.println("Test 3 - inserting and removing numbers");
sudoku test3 = new sudoku(sudokuTest3);
test3.displaySudoku();
test3.insertNumber( newNumber: 9, column: 4, row: 5);
test3.displaySudoku();
System.out.println("attempting to insert a number too large");
test3.insertNumber( newNumber: 11, column: 4, row: 5);
test3.removeNumber( column: 4, row: 5);
test3.displaySudoku();
test3.checkIfCompleted();
```

Results of Test 3



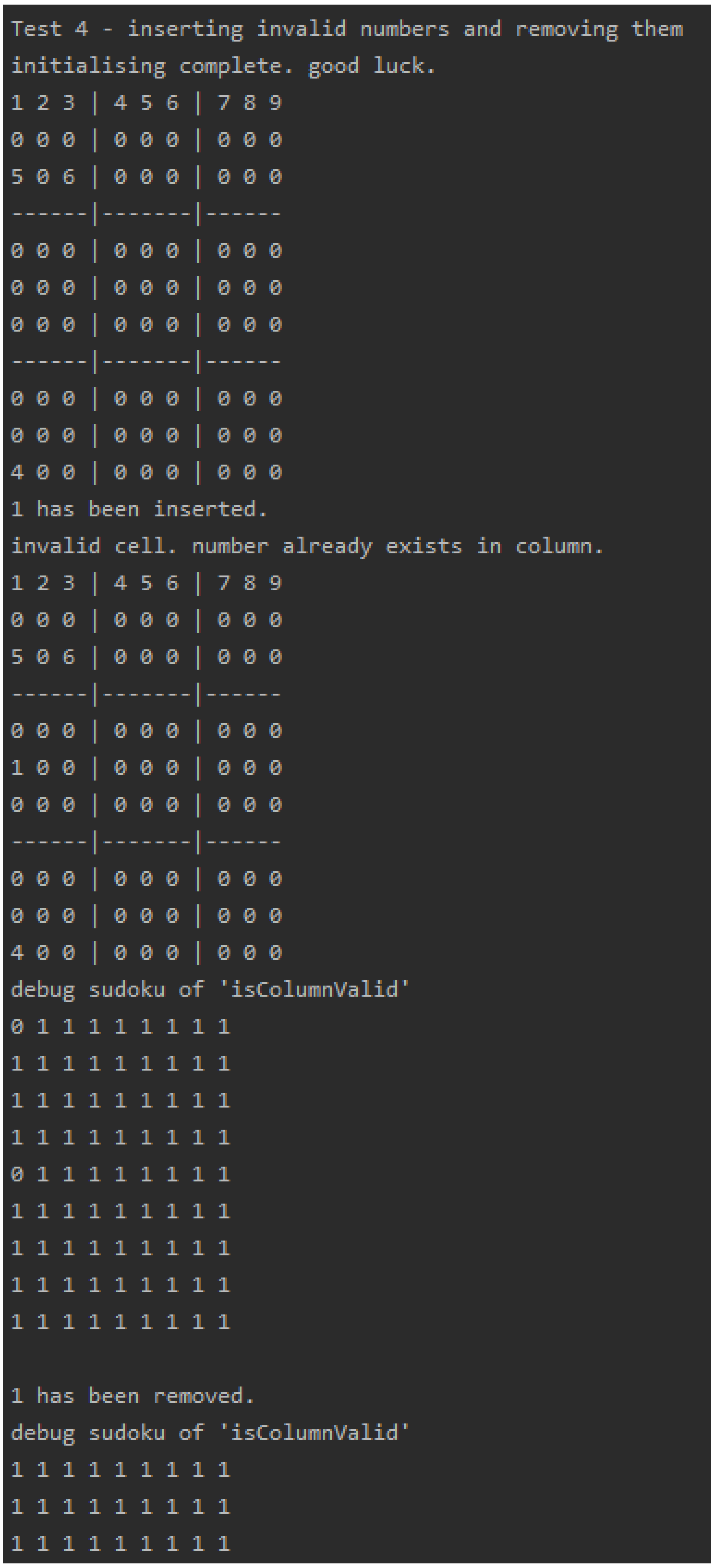
```
Test 3 - inserting and removing numbers
initialising complete. good luck.
1 0 7 | 0 0 0 | 0 0 2
0 4 0 | 0 0 0 | 0 0 0
5 0 3 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
4 0 0 | 0 0 0 | 0 0 0
9 has been inserted.
1 0 7 | 0 0 0 | 0 0 2
0 4 0 | 0 0 0 | 0 0 0
5 0 3 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 9 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
4 0 0 | 0 0 0 | 0 0 0
attempting to insert a number too large
invalid number being inserted. please insert a number from 1 - 9.
9 has been removed.
1 0 7 | 0 0 0 | 0 0 2
0 4 0 | 0 0 0 | 0 0 0
5 0 3 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
4 0 0 | 0 0 0 | 0 0 0
sudoku is only 8.64% complete. keep it up!
```

Test 4



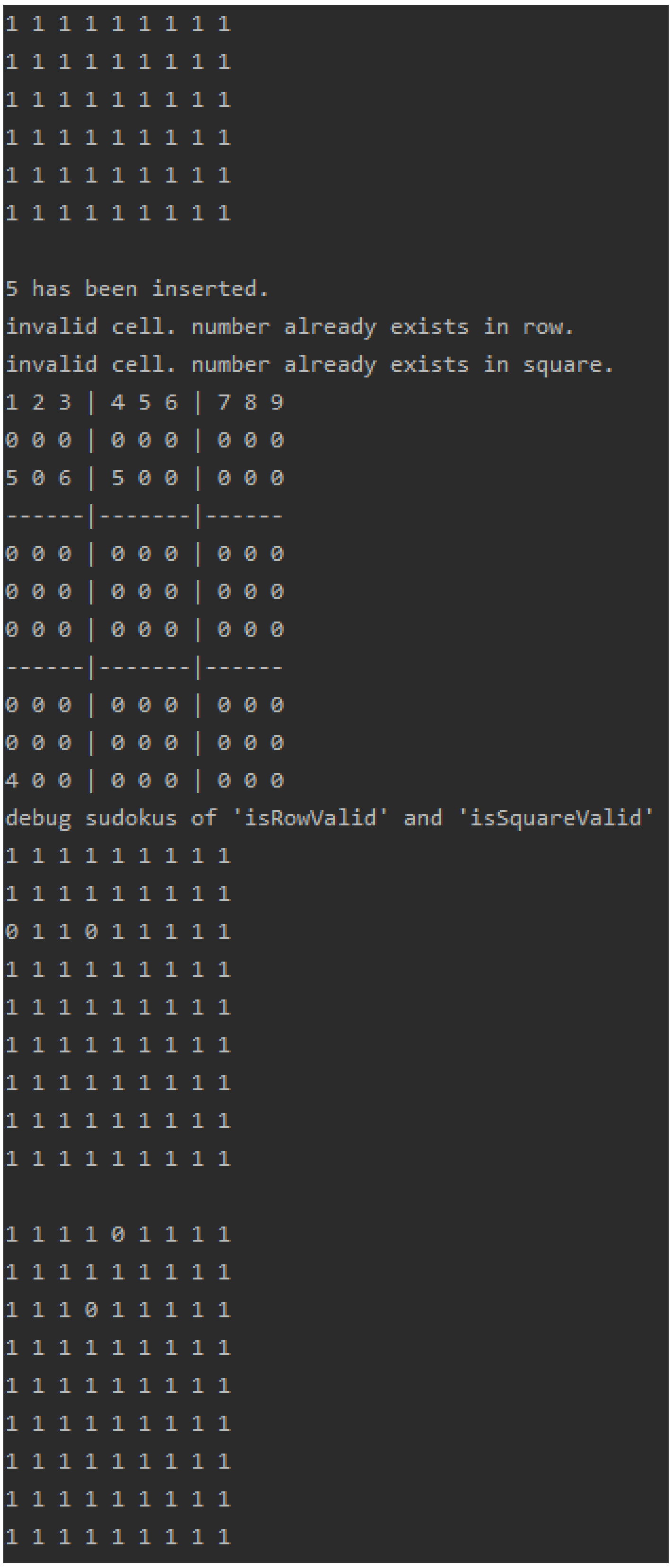
```java
int[][] sudokuTest4 = {
        {1, 2, 3, 4, 5, 6, 7, 8, 9},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {5, 0, 6, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {0, 0, 0, 0, 0, 0, 0, 0, 0},
        {4, 0, 0, 0, 0, 0, 0, 0, 0}
};
System.out.println("Test 4 - inserting invalid numbers and removing them");
sudoku test4 = new sudoku(sudokuTest4);
test4.displaySudoku();
test4.insertNumber( newNumber: 1,  column: 4,  row: 0);
test4.displaySudoku();
System.out.println("debug sudoku of 'isColumnValid'");
test4.displaySudokuDebug( para: "isColumnValid");
test4.removeNumber( column: 4,  row: 0);
System.out.println("debug sudoku of 'isColumnValid'");
test4.displaySudokuDebug( para: "isColumnValid");
test4.insertNumber( newNumber: 5,  column: 2,  row: 3);
test4.displaySudoku();
System.out.println("debug sudokus of 'isRowValid' and 'isSquareValid'");
test4.displaySudokuDebug( para: "isRowValid");
test4.displaySudokuDebug( para: "isSquareValid");
```

Results of Test 4



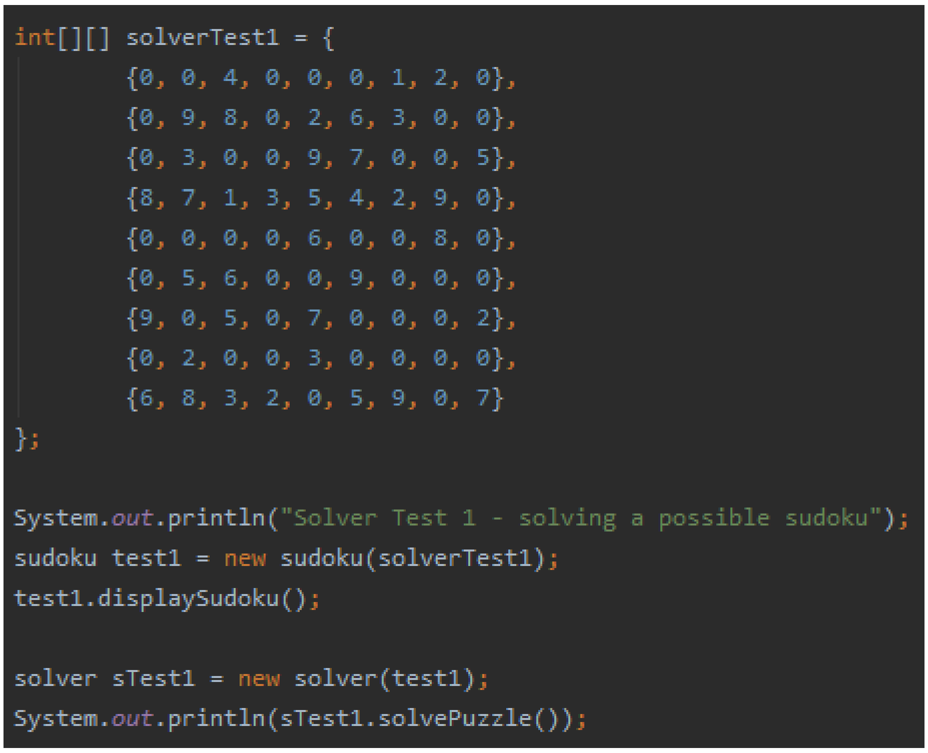
```
Test 4 - inserting invalid numbers and removing them
initialising complete. good luck.
1 2 3 | 4 5 6 | 7 8 9
0 0 0 | 0 0 0 | 0 0 0
5 0 6 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
4 0 0 | 0 0 0 | 0 0 0
1 has been inserted.
invalid cell. number already exists in column.
1 2 3 | 4 5 6 | 7 8 9
0 0 0 | 0 0 0 | 0 0 0
5 0 6 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
1 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
4 0 0 | 0 0 0 | 0 0 0
debug sudoku of 'isColumnValid'
0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
0 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1

1 has been removed.
debug sudoku of 'isColumnValid'
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
```

Results of Test 4 Continued



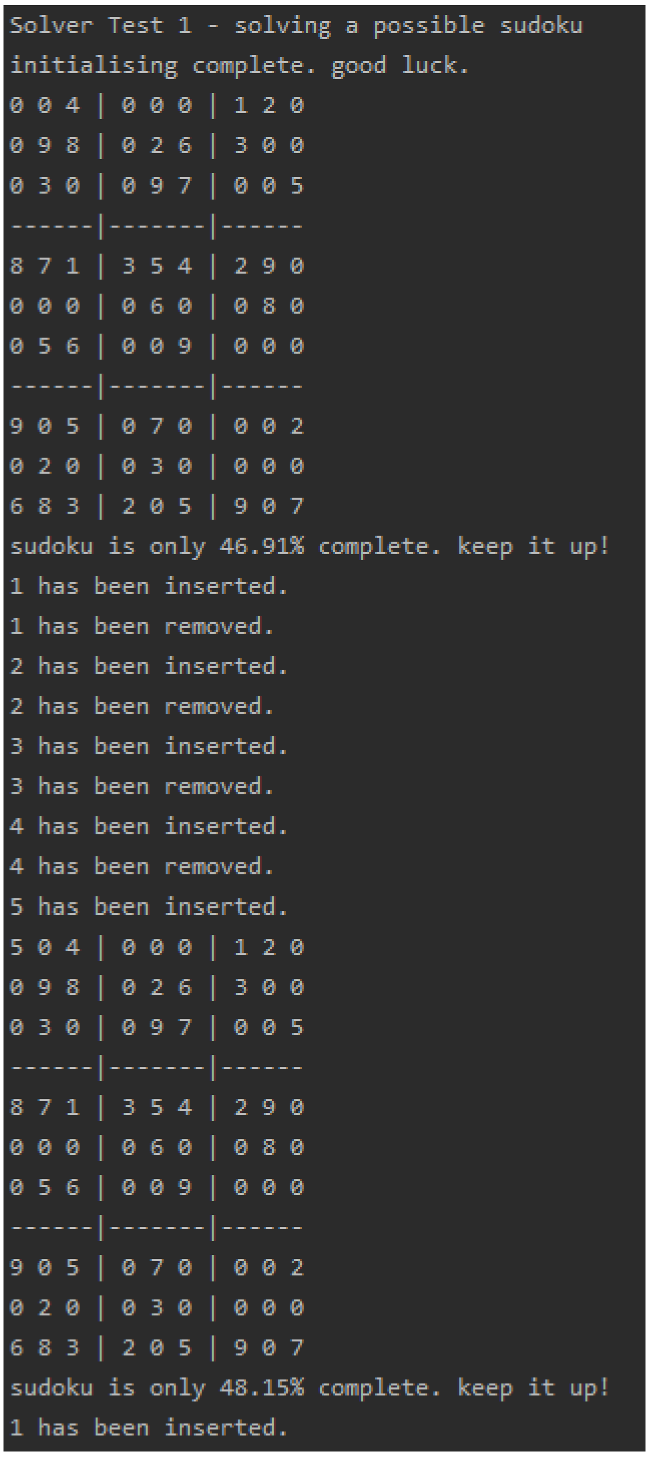
```
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1

5 has been inserted.
invalid cell. number already exists in row.
invalid cell. number already exists in square.
1 2 3 | 4 5 6 | 7 8 9
0 0 0 | 0 0 0 | 0 0 0
5 0 6 | 5 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
------|-------|------
0 0 0 | 0 0 0 | 0 0 0
0 0 0 | 0 0 0 | 0 0 0
4 0 0 | 0 0 0 | 0 0 0
debug sudokus of 'isRowValid' and 'isSquareValid'
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
0 1 1 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1

1 1 1 1 0 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 0 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1
```

Test 5



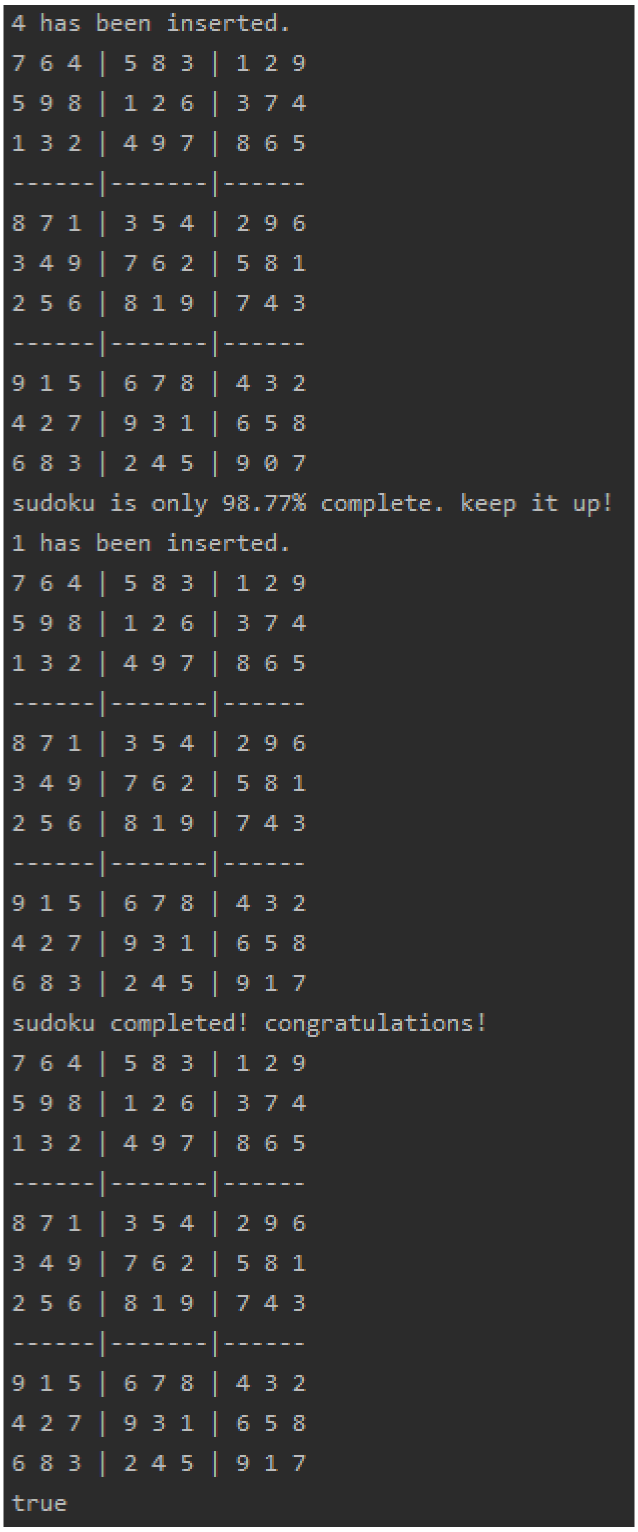
```
int[][] solverTest1 = {
        {0, 0, 4, 0, 0, 0, 1, 2, 0},
        {0, 9, 8, 0, 2, 6, 3, 0, 0},
        {0, 3, 0, 0, 9, 7, 0, 0, 5},
        {8, 7, 1, 3, 5, 4, 2, 9, 0},
        {0, 0, 0, 0, 6, 0, 0, 8, 0},
        {0, 5, 6, 0, 0, 9, 0, 0, 0},
        {9, 0, 5, 0, 7, 0, 0, 0, 2},
        {0, 2, 0, 0, 3, 0, 0, 0, 0},
        {6, 8, 3, 2, 0, 5, 9, 0, 7}
};

System.out.println("Solver Test 1 - solving a possible sudoku");
sudoku test1 = new sudoku(solverTest1);
test1.displaySudoku();

solver sTest1 = new solver(test1);
System.out.println(sTest1.solvePuzzle());
```

Results of Test 5 (start of algorithm)



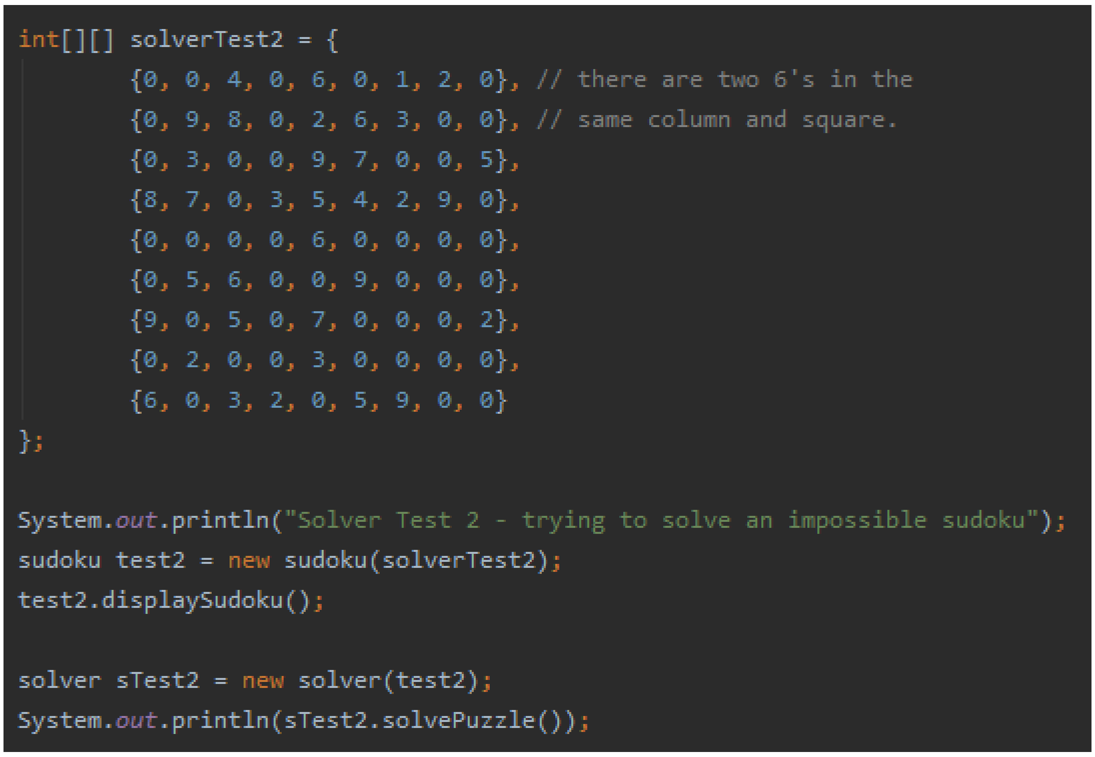
```
Solver Test 1 - solving a possible sudoku
initialising complete. good luck.
0 0 4 | 0 0 0 | 1 2 0
0 9 8 | 0 2 6 | 3 0 0
0 3 0 | 0 9 7 | 0 0 5
------|-------|------
8 7 1 | 3 5 4 | 2 9 0
0 0 0 | 0 6 0 | 0 8 0
0 5 6 | 0 0 9 | 0 0 0
------|-------|------
9 0 5 | 0 7 0 | 0 0 2
0 2 0 | 0 3 0 | 0 0 0
6 8 3 | 2 0 5 | 9 0 7
sudoku is only 46.91% complete. keep it up!
1 has been inserted.
1 has been removed.
2 has been inserted.
2 has been removed.
3 has been inserted.
3 has been removed.
4 has been inserted.
4 has been removed.
5 has been inserted.
5 0 4 | 0 0 0 | 1 2 0
0 9 8 | 0 2 6 | 3 0 0
0 3 0 | 0 9 7 | 0 0 5
------|-------|------
8 7 1 | 3 5 4 | 2 9 0
0 0 0 | 0 6 0 | 0 8 0
0 5 6 | 0 0 9 | 0 0 0
------|-------|------
9 0 5 | 0 7 0 | 0 0 2
0 2 0 | 0 3 0 | 0 0 0
6 8 3 | 2 0 5 | 9 0 7
sudoku is only 48.15% complete. keep it up!
1 has been inserted.
```

Results of Test 5 (end of algorithm)



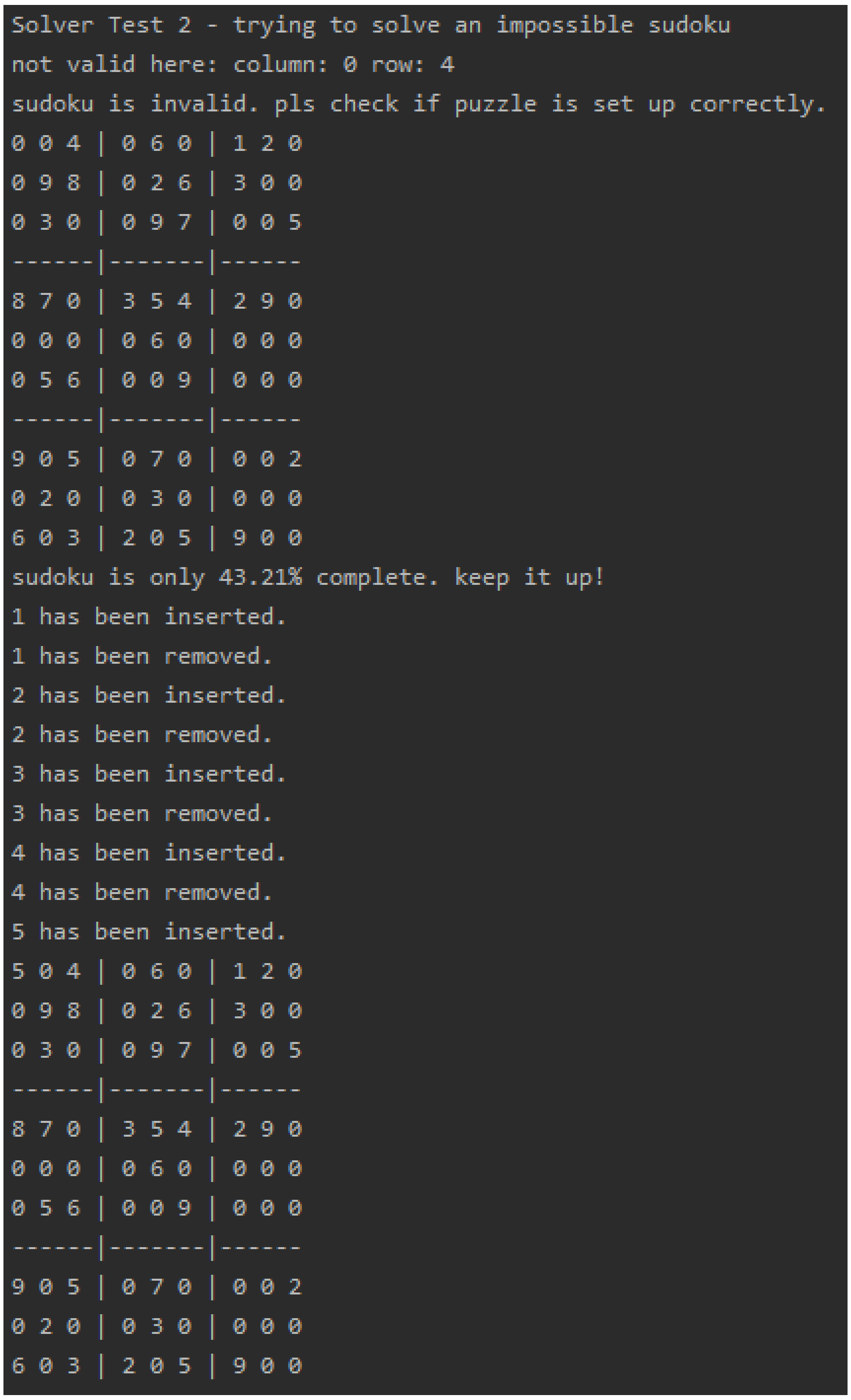
```
4 has been inserted.
7 6 4 | 5 8 3 | 1 2 9
5 9 8 | 1 2 6 | 3 7 4
1 3 2 | 4 9 7 | 8 6 5
------|-------|------
8 7 1 | 3 5 4 | 2 9 6
3 4 9 | 7 6 2 | 5 8 1
2 5 6 | 8 1 9 | 7 4 3
------|-------|------
9 1 5 | 6 7 8 | 4 3 2
4 2 7 | 9 3 1 | 6 5 8
6 8 3 | 2 4 5 | 9 0 7
sudoku is only 98.77% complete. keep it up!
1 has been inserted.
7 6 4 | 5 8 3 | 1 2 9
5 9 8 | 1 2 6 | 3 7 4
1 3 2 | 4 9 7 | 8 6 5
------|-------|------
8 7 1 | 3 5 4 | 2 9 6
3 4 9 | 7 6 2 | 5 8 1
2 5 6 | 8 1 9 | 7 4 3
------|-------|------
9 1 5 | 6 7 8 | 4 3 2
4 2 7 | 9 3 1 | 6 5 8
6 8 3 | 2 4 5 | 9 1 7
sudoku completed! congratulations!
7 6 4 | 5 8 3 | 1 2 9
5 9 8 | 1 2 6 | 3 7 4
1 3 2 | 4 9 7 | 8 6 5
------|-------|------
8 7 1 | 3 5 4 | 2 9 6
3 4 9 | 7 6 2 | 5 8 1
2 5 6 | 8 1 9 | 7 4 3
------|-------|------
9 1 5 | 6 7 8 | 4 3 2
4 2 7 | 9 3 1 | 6 5 8
6 8 3 | 2 4 5 | 9 1 7
true
```

Test 6



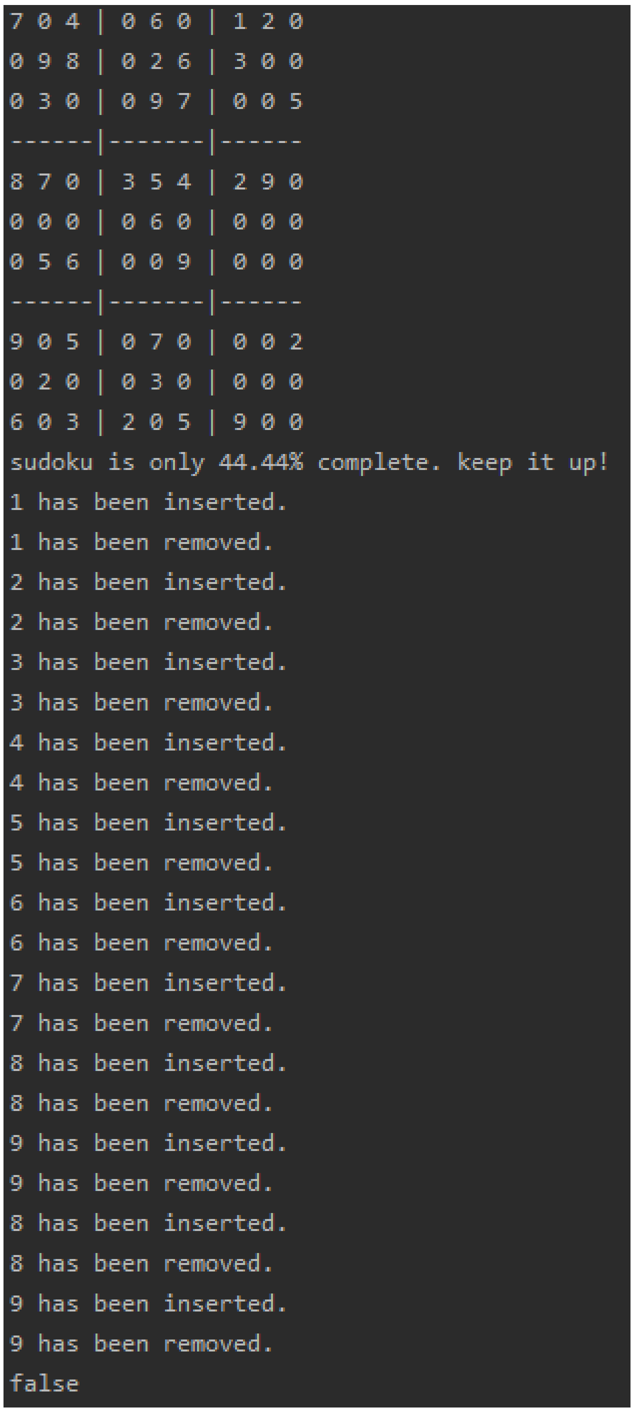
```java
int[][] solverTest2 = {
        {0, 0, 4, 0, 6, 0, 1, 2, 0}, // there are two 6's in the
        {0, 9, 8, 0, 2, 6, 3, 0, 0}, // same column and square.
        {0, 3, 0, 0, 9, 7, 0, 0, 5},
        {8, 7, 0, 3, 5, 4, 2, 9, 0},
        {0, 0, 0, 0, 6, 0, 0, 0, 0},
        {0, 5, 6, 0, 0, 9, 0, 0, 0},
        {9, 0, 5, 0, 7, 0, 0, 0, 2},
        {0, 2, 0, 0, 3, 0, 0, 0, 0},
        {6, 0, 3, 2, 0, 5, 9, 0, 0}
};

System.out.println("Solver Test 2 - trying to solve an impossible sudoku");
sudoku test2 = new sudoku(solverTest2);
test2.displaySudoku();

solver sTest2 = new solver(test2);
System.out.println(sTest2.solvePuzzle());
```

Results of Test 6 (start of algorithm)



```
Solver Test 2 - trying to solve an impossible sudoku
not valid here: column: 0 row: 4
sudoku is invalid. pls check if puzzle is set up correctly.
0 0 4 | 0 6 0 | 1 2 0
0 9 8 | 0 2 6 | 3 0 0
0 3 0 | 0 9 7 | 0 0 5
------|-------|------
8 7 0 | 3 5 4 | 2 9 0
0 0 0 | 0 6 0 | 0 0 0
0 5 6 | 0 0 9 | 0 0 0
------|-------|------
9 0 5 | 0 7 0 | 0 0 2
0 2 0 | 0 3 0 | 0 0 0
6 0 3 | 2 0 5 | 9 0 0
sudoku is only 43.21% complete. keep it up!
1 has been inserted.
1 has been removed.
2 has been inserted.
2 has been removed.
3 has been inserted.
3 has been removed.
4 has been inserted.
4 has been removed.
5 has been inserted.
5 0 4 | 0 6 0 | 1 2 0
0 9 8 | 0 2 6 | 3 0 0
0 3 0 | 0 9 7 | 0 0 5
------|-------|------
8 7 0 | 3 5 4 | 2 9 0
0 0 0 | 0 6 0 | 0 0 0
0 5 6 | 0 0 9 | 0 0 0
------|-------|------
9 0 5 | 0 7 0 | 0 0 2
0 2 0 | 0 3 0 | 0 0 0
6 0 3 | 2 0 5 | 9 0 0
```

Results of Test 6 (end of algorithm)



```
7 0 4 | 0 6 0 | 1 2 0
0 9 8 | 0 2 6 | 3 0 0
0 3 0 | 0 9 7 | 0 0 5
------|-------|------
8 7 0 | 3 5 4 | 2 9 0
0 0 0 | 0 6 0 | 0 0 0
0 5 6 | 0 0 9 | 0 0 0
------|-------|------
9 0 5 | 0 7 0 | 0 0 2
0 2 0 | 0 3 0 | 0 0 0
6 0 3 | 2 0 5 | 9 0 0
sudoku is only 44.44% complete. keep it up!
1 has been inserted.
1 has been removed.
2 has been inserted.
2 has been removed.
3 has been inserted.
3 has been removed.
4 has been inserted.
4 has been removed.
5 has been inserted.
5 has been removed.
6 has been inserted.
6 has been removed.
7 has been inserted.
7 has been removed.
8 has been inserted.
8 has been removed.
9 has been inserted.
9 has been removed.
8 has been inserted.
8 has been removed.
9 has been inserted.
9 has been removed.
false
```