Final Year Project Report

Full Unit - Interim Report

Playing Games and Solving Puzzles using AI

Dimitar Seraffimov

A report submitted in part fulfilment of the degree of

BSc in Computer Science with Software Engineering

Supervisor: Magnus Wahlstrom

Department of Computer Science
Royal Holloway, University of London

December 7, 2023

Declaration

This report has been prepared on the basis of my own work. Where other published and unpublished source materials have been used, these have been properly acknowledged.

Word Count: 6290

Student Name: Dimitar Seraffimov

Date of Submission: December 7, 2023

Signature: Dimitar Seraffimov

Table of Contents

Abstract

Sudoku is a logic-based, combinatorial number-placement game that became popular in the last decade. The Japanese puzzle became so sensational and influential that it managed to quickly spread around the world and capture the attention of millions. The simple structure and clear rules made it a burning passion for millions of people at some point in their life. Even though the game does not require strong mathematical skills, it is essential to have solid problem-solving and logical thinking to solve it in timely manner. Due to the puzzle's popularity, different creative variants emerged quickly, changing the size of the grid, or adding additional rules are typical examples of the new game features. As interesting they can get, the new modifications still did not manage to provide sufficient excitement in order to completely replace the original puzzle game.

Chapter 1: **Introduction**

The worldwide passion around the game, created gigantic appetite and desire for some sort of competitive aspect. Following the logic and structure behind the original game, it took a logical development direction - creating different difficulty levels and tracking the completion time, are the most popular benchmarks for players to compare and showcase their abilities. Drawing such international attention, naturally involved software developers to create different algorithms to find all possible solutions, one specific and the best / fastest solution.

There are many different human tactics and software approaches to solving the problem, but this project focuses on researching, developing, and implementing algorithmic solution to the original Sudoku puzzle constraints. Constraint Satisfaction [Apt03] is a leading approach for solving search problems and in this instance, it is implemented around the rules in the Sudoku game. The fundamental technique chosen to successfully solve the search problem is to incorporate efficient backtracking algorithm with strategic backjumping.

Following the completed proof of concept programs, it is obvious to me that thanks to its time complexity, the chosen methodology is a solid foundation for future improvements of the application and with some additional fine-tuning it can achieve a good final application. The planned extension of the project to android mobile application [Eis15] is an ideal stepping stone and gives enormous variety of possible functionalities to be researched and developed in the second term. Detailed documentation and explanation of the mentioned functionalities can be found in the Aims and Objectives chapter.

1.1 Sudoku Puzzle Constraints

A Sudoku puzzle can be characterized as a partly finished N×N grid. A general Sudoku puzzle has 9x9 rows and columns, and 3x3 minigrids which account to nine cells each. All together, the full grid has 81 cells in total.

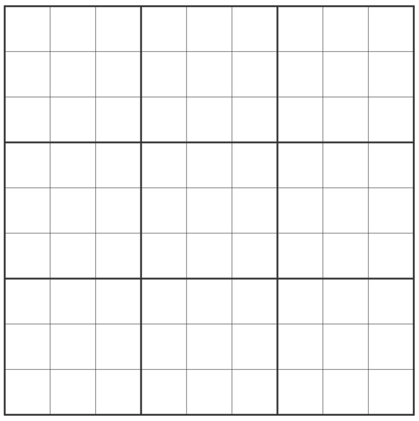

Figure 1.1: An empty Sudoku Grid

The simplest way to describe the game's constraints is that each digit shows up once in every unit(row, column, minigrid). The mathematical approach of the Sudoku configuration can be summed up to a N×N rows, columns, and grid divided into N-areas or minigrids, where each of the N-rows, columns, and minigrids have N-cells and each of the N-digits occurs precisely once in every row, column, and minigrid.

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

Figure 1.2: Partially completed Sudoku Grid

Despite the si mple rules of the game there exists no generalised method to determine the minimal number of pre-filled cells needed at the start of the game to guarantee a unique solution. This is the reason for the vast number (6.67×10^{21}) of potential distinct Sudoku puzzle configurations. To ensure a single, unique solution a Sudoku grid needs minimum 17 clues. This implies that a puzzle with less than 17 clues will inevitable have two or more solutions. However, it is not excluded for a valid Sudoku grid with 17 or more clues to have multiple solutions.

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

Figure 1.3: Sudoku puzzle with only 17 clues

1.2 Problem Statement

The primary challenge in developing a Sudoku solving game, lies in the integration of different additional functionalities with the end goal to create a user friendly and efficient application. The direction of this project is to create a beautiful and seamless user experience with the focus not only on solving and generating new puzzles with unique solutions but also exploring other functionalities in order to enhance the player's involvement.

The features which will present a challenge to the project in the second term are offering and categorising different difficulty levels to the player, giving hints for the best next step and keeping count of the provided, and last but not least - keeping local and online track of previous attempts. There are number of in-game features which will be challenging to visually and logically implement in the program. I have highlighted five of them which are essential to me in the *Aims and Objective* section of the report.

I completely understand that there is a significant difference between the *'FYP-Plan'* I have submitted in the beginning of the term and this Interim Report. I hope this will be seen as personal development and better professional understatement of my Final Year Project, and will not be penalised.

1.2.1 Algorithmic Complexity and Integration

Thus far, the developed Proof of Concept programs research and implement a constraint satisfactory [BPS99] backtracking algorithm which solves Sudoku puzzles and prints in the command-line interface (CLI) the starting grid and below - the solved one. The current backtracking algorithm [Civ13] is fine-tuned for efficiency and accuracy, ensuring that in a timely manner it can handle a wide range of puzzle complexity. At this point, the starting state of the Sudoku grids are defined as 2D lists, containing the specific number arrangement and empty cells(denoted by 0), inside each Proof of Concept program and are not being generated by a specific program.

One of the next challenges will be to develop an algorithm dedicated to creating puzzles with unique solution that not only follow the standard Sudoku rules but also categorises them into different difficulty levels. To generate a Sudoku puzzle the criteria for determining the level of difficulty needs to be considered. The first factor affecting the criteria is the number of empty cells in the initial puzzle - it is generally safe to assume that the more empty cells there are, the higher the level of difficulty. It is important not to forget that to ensure a unique solution the grid needs minimum of 17 clues (Sudoku Puzzle Constraints). The second influential factor in determining the puzzle's difficulty is the placement on the grid of the different initial numbers. From observation, simpler puzzles usually have their initial number clues evenly spaced apart, compared to the more difficult ones which have their clues clustered in small groups.

From a developer's standpoint, to correctly evaluate the difficulty level of a Sudoku puzzle, the effort and tools used by the program to solve the puzzle need to be analysed. This is an aspect on which if I decide to focus, it will shift the *Aims and Objective* of the whole project. The reason behind this, is that I will have to compare different solving techniques and some of them do not align with my primary ambition and goal of this project. Which simply put, is to research and implement more advanced backtracking algorithms and Artificial Intelligence techniques, as well as eventually to implement machine vision to allow the user to scan Sudoku puzzles using their phone's camera.

The techniques discussed and developed in the book *Programming Sudoku* [Lee06] primarily fall under the category of pattern-based solving strategies and brute-force elimination methods. The listed approaches are Column, Row, and Minigrid Elimination (CRME), Lone Rangers, Twins / Triplets in minigrids/columns/rows, and Brute-force elimination which are undeniably valuable for a human approach to solving Sudoku puzzles, as they help the player discover patterns and strategically eliminate possibilities. However, the project does not focus on comparing human-like methodologies and would inevitably divert my resources and time away from exploring more sophisticated algorithmic solutions. The approach I will implement is discussed in the Generating Sudoku Puzzles future Proof of Concept programs.

As already mentioned, an intriguing but challenging extension of the project is implementing machine vision that allows the user to scan Sudoku puzzles using the phone's camera and have them solved by the application. The described feature is really advanced and requires in-depth research and the development of new skills, particularly in image processing and pattern recognition. Furthermore, the described extension demands substantial research and understanding in image processing and pattern recognition technologies, which are distinct from the conventional logic-based Sudoku solving methods.

1.2.2 User Interface and Interaction Design

When discussing the User Interface and Interaction Design [RA03], the main challenge lies in crafting an interface that is not only visually appealing but also intuitive and efficient for the user. My focus is on mitigating and tackling the risk on focusing too much effort and time on the front end of the application. The methodology to achieve this, consists of minimalist design, single color scheme and easy to navigate single-page main menu. This approach is used with the intent to streamline the user interaction with the system, reducing unnecessary complexity and enhancing the overall user experience.

To ensure all essential application requirements are covered, I systematically followed User Centered Design (UCD) methodologies. This organised approach benefited me in gaining a better understanding and identifying the expectations and needs of the end-user. The techniques utilised to guide me in the development process are thoroughly discussed in the User Centered Design subsection of the report.

Chapter 2: **Aims and Objectives**

When describing the fundamental and possible extensions of the Aims and Objectives for this project, important to me is achieving a harmonic combination between seamless user experience and professionally implemented and documented back end software[FH+01]. This chapter of the report and it's sections are dedicated to present all back end and front end functionalities, as well as introduce a possible extension of the project suggested by the department.

My vision for the future application follows minimalist, yet effective principles and processes. The process of starting a new Sudoku game will present the user with an option to choose from different difficulty levels. When started, a dedicated program will generate a Sudoku puzzle with single unique solution and will present the game screen with different in-game functionalities. On leaving this screen, the program will automatically save the current status of the game and the user will be able to continue from the last state of the previous not finished attempt. Expanding further on this, the application will keep local and online track of the completed previous attempts, giving insight to the date of completion, previous difficulty level, completion time and count of validations used.

When describing the in-game user experience, it is essential to implement five important functionalities – validation, partial hints or complete puzzle solution, one step back and puzzle completion time. Each one of the aforementioned functionalities significantly enhances user engagement, by offering a chance to the user to assess their strategy and introducing a competitive element, encouraging them to improve their problem solving skills. All functionalities mentioned above and their objectives will be further discussed in the following sections of this chapter.

Apart from extending this project to a mobile application, I am planning on exploring and possibly implementing machine vision [BW12] to enter and automatically solve puzzles for the user. This special feature is not yet entirely researched by me but I will do my best to further explain it in the Extension: Machine Vision section. This development path for my project is incredibly exciting to me, even though it will require a lot of research and code development [WH14].

2.1 Algorithmic Approach and Puzzle Generation

The primary objective of the project's algorithmic approach is to ensure each generated Sudoku puzzle is solved with its unique solution. To achieve this, a backtracking algorithm is being developed to best satisfy the constraints in the Sudoku puzzles. This technique, which in its essence is a depth-first search, when applied to the Sudoku game's strictly defined configuration is highly effective in iterating through all possible solutions.

The algorithm developed inside *'backtracking_algo.py'* in the third Proof of Concept program is the backbone of the whole project, as it takes the responsibility for finding the unique solution to the Sudoku puzzles. Solving Sudoku problems involves a form of exhaustive search of all possible configurations. However, exploiting constraints to rule out certain possibilities for specific positions is the key method utilised to improve the solving time of the Sudoku puzzles. This approach takes advantage of the game's rules by eliminating possible options and simplifies the process of finding the correct solution. The fine-tuned algorithm for this project will be covered in detail in the Completed Proof of Concept Programs section of the report.

The project has yet not approached the puzzle generation and difficulty ranking problem of creating a Sudoku game. As already outlined in the Algorithmic Complexity and Integration subsection, there are two main factors that affect the difficulty of the project. To evaluate the difficulty level of the generated project, I will use a simple yet effective technique - the puzzles will be generated as fully completed and randomised Sudoku grids, after which a number of cells will be randomly removed to create a ready-to-solve puzzle. Key part of this process is ensuring that each puzzle has only one, unique solution - this will be achieved by leaving minimum of 17 clues in the standard 9x9 grid [Lee06]. The choice of how many cells to remove will depend entirely on the level of difficulty chosen by the user. To ensure a different puzzle is generated each time, the list of possible values is randomised. The application will have four levels of difficulty - Easy, Medium, Hard and Extreme.

Level	Empty Cells
1 (Easy)	40 to 45
2 (Medium)	46 to 49
3 (Hard)	50 to 53
4 (Extreme)	54 to 58

Table 2.1: Number of Empty Cells for Each Difficulty Level

Note: It is important to specify, that I will only be judging the level of difficulty based on the number of empty cells in the generated puzzle. I will not be considering the positioning of the empty cells as as a factor affecting the difficulty level, as well as how much algorithmic ”effort” is needed to solve it.

2.2 In-Game Functionalities

In the project specification provided by the Department of Computer Science there are multiple extensions discussed and particular emphasis is placed on enhancing the in-game user experience [Dep23]. I have identified and decided to implement five critical functionalities. Each one of them will be carefully designed and developed to not only enhance the user engagement but also to provide strategy assessment and competitive element to the application. The detailed objectives of these features are discussed in this section of the report.

1. Validation Option:
    - The validation option will allow the user to compare the current progress to the unique correct puzzle solution - it will indicate the existing (or lack of) mistakes. It will serve as a real-time assessment tool and will benefit the user in improving or comparing their problem-solving techniques.
2. Validation Count:
    - The application will also keep track of the validation count, this feature will track the number of validations performed for each specific puzzle - this provides additional competitive element. Keeping track of the performed validations will act as a reflective tool for the player and will keep good statistics of their progress based on their previous attempts.
3. Hints and Complete Solution:

- Providing one step forward hints, as well as adding the option to solve the whole problem are necessary functions and they will also be tracked by the system. These specific features will ensure that the game remains accessible and enjoyable for all players, regardless of their skill level. The system will also track the usage of these assistance tools.

4. One Step Back:
    - This features is as important as the above-mentioned functionalities, allowing the user to take one step back in the game. This functionality adds one more layer of user assistance and ensures that the player can enjoy and make the best use of the environment.
5. Completion Time Tracking:
    - Incredibly important feature to me, as this introduces a competitive aspect to the application. This feature will record the time taken by the player to solve each Sudoku puzzle. My goal with adding this functionality, is fostering a sense of achievement in the player's mind as they see their solving times decrease with more practice and experience.

The described in-game features will act as statistics and they will be displayed at completing each puzzle and will be stored in a list of previously completed games. The immediate feedback allows players to asses their performance in the specific game they have completed and offer insights into the problem-solving methods they have used. My idea with this "archival" feature is to not only add more depth to the game but also create a sense of continuous learning and achievement.

2.3 Graphical User Interface

The graphical user interface (GUI) of the application is focused on a minimalist approach, emphasizing functionality over complicated visual effects. The aim is to create a simple, easy-to-navigate design that focuses on the user interactions without unnecessary complexity. By prioritizing essential functionalities and user-friendly design elements, the interface will be tailored to provide an efficient and engaging experience for the player. This approach ensures intuitive and accessible application interface, catering to users' needs while maintaining a clean and straightforward aesthetic.

2.4 Extension: Machine Vision

Apart from the above discussed functionalities, probably the most challenging and interesting extension to me is using machine vision (the phone's camera) to allow the user to input a completely new puzzle to the system. I have a good idea and understanding of the functionalities which will become fundamental part of the final mobile application, but the implementation of machine vision will require detailed research and acquiring new set of skills.

My primary motivation and interest in incorporating Machine Vision into the Sudoku application is to implement additional level of user interaction and gain practical knowledge in advance image processing. The feature will allow the user to scan Sudoku puzzles using their phone's camera, which the application will interpret and input into the solving algorithm.

If implemented, this will also be the only way for the user to input custom Sudoku grids, increasing the user's overall gameplay experience.

By pursuing this possible extension will not only add additional functionality level for the user but will also add completely different level of sophistication and research to the development process. I will have to research sophisticated pattern recognition algorithms [BW12], capable of effectively distinguishing numbers and the Sudoku grid lines in various lightning conditions, angles and backgrounds. The goal will be to achieve high success rate in correctly scanning and interpreting puzzles, also comparing different approaches and algorithms to solve this problem [SUW18].

Before implementing this feature, the fundamentals of the game and in-game functionalities need to be completed and after that I can start to integrate the machine vision into the Sudoku application. It will represent a significant step in advancing the technical sophistication [WH14]of the application and the real-world use cases for the user. The implementation of this feature is not just about adding a new functionality but for me it is more about embedding a new layer of technological complexity that transforms completely how the users will interact with the application. The process of scanning a completely new and custom puzzle will involve the user scanning the puzzle, after that confirming the scanned puzzle and the application's algorithm solves automatically the confirmed puzzle.

To deepen my knowledge and enhance my understanding in this area, I am committed to thoroughly review and study the various sources that have been referenced in this section.

Chapter 3: **Theoretical Analysis**

In this chapter of the report, my focus is to showcase the researched theories and explain some parts of them. For better readability and understatement of the discussed topics, I have logically divided the chapter into Algorithms Analysis and User Interface Theory sections.

In the Algorithms Analysis section, a more theoretical approach is taken to explain the strategies for finding the best solution and puzzle generation. This includes brief theory explanation of the solution to the N-Queens puzzle(my first Proof of Concept Program), the backtracking algorithm's approach [MG17] and the challenge behind puzzle difficulty categorisation [Lee06].

In the User Interface Theory section, my focus shifts to the structured design principles and interaction models applied to shape the application's front end. This includes short explanation on the Unified Modeling Language (UML) diagram and the practical user stories, which outline the framework and key interactions inside the application. In the Software Engineering chapter I will go in detail about the methods and technology frameworks used.

3.1 Algorithms Analysis

This section gives up a closer look at the technical aspects that make the software effective and accurate. The algorithms are broken down with the goal explaining the theory behind them and understand their role in the application's functionalities.

3.1.1 N-Queens Problem

This is a classic constraint satisfactory problem, in which N queens are placed on an N×N chessboard in such a way that no two queens threaten each other[RVW06]. The developed program inside the "First Proof of Concept Program" finds all possible solutions for a given number of queens. The "*n_queens_solver.py*" program uses a recursive backtracking algorithm to find all possible solutions to the puzzle. This algorithm places queens on the board and iterates through the possible positions, backtracking when a conflict arises and therefore exploring all potential configurations.

Each solution is represented as a list of N tuples, where each tuple represents the position of a queen on the board (row, column). My solution is configurable to different NxN board sizes, it provides a clear demonstration of the recursive backtracking power in solving problems that require exploring multiple possibilities and making sequential decisions.

3.1.2 Backtracking Algorithm

Based on my research and understanding, the backtracking recursion algorithm is a systematic method to iterate through all possible configurations of a search space. It is a general algorithm which must be customized for each problem and set of constraint satisfactions [Civ13]. In the "Third Proof of Concept Programs" I explore the application of recursive backtracking to solve simple and complex Sudoku puzzles [RVW06]. The program's documentation discusses two recursion types: basic recursion and backtracking recursion. The basic recursion can be described as self-calling function that solves smaller segments of a larger problem until a base case is reached, and the backtracking recursion is a more systematic approach involving depth-first search of all potential configurations. Backtracking is the key to implementing exhaustive search programs correctly and efficiently [Lee06] [DHS10]. Based on my personal understatement and the simplest way for me to explain it, backtracking is a special approach to problem-solving that involves:

- depth-first search on an implicit graph of configurations;
- exploring all possible solutions to a problem;
- eliminating those solutions that fail to satisfy the constraints of the problem;
- "backtracking" to a previous step and trying alternative solutions.

I have identified three main categories of problems that can be solved by using backtracking algorithmic recursion:

- enumeration problem: the goal is to generate all possible solutions to a problem or count the total number of possible solutions to a problem;
- decision problem: the goal is to find one specific solution to a problem or prove that one exists;
- optimization problem: the goal is to find the best possible solution to a given problem.

In the case of this project, the focus will be to solve the given Sudoku puzzle by the one and only possible solution. Therefore, the backtracking algorithm will be used to solve the above described "decision problem".

3.2 User Interface Theory

In the development of the user interface, I have approached and utilised two key theoretical principles, I learned from and used in my last year's "CS2800: Software Engineering" module - UML (Unified Modeling Language) diagrams and detailed User Stories. During my research on the best approach to develop the interface, I have searched for inspiration and found motivation in the following book "Systems analysis and design: An object-oriented approach with UML" by A. Dennis [DWT15].

The UML diagram serves as a visual and schematic representation of the system's architecture, providing a structured framework that visualises various components and their interactions. This methodology, rooted in object-oriented theory, offers a clear and systematic approach to visualize and document the design of the system, ensuring coherence and clarity in the development process. On the other hand, the user stories offer a human-centric approach, focusing on the practical needs and interactions of the end-users. Originating from Agile methodologies [FH+01], user stories encapsulate the requirements and functionalities of the system from the user's perspective, guiding the development process towards creating features that are directly aligned with the user's needs and preferences.

Chapter 4: **Development Progress**

In this chapter of the report, I provide an overview of the technical implementation required for bringing the application to life. The following sections are intended to offer a clear understanding of the current stage of development and outline the planned future programs. I will be discussing the Programming Languages & Frameworks in the linked section.

4.1 Completed Proof of Concept Programs

This section includes a detailed explanation of the developed algorithms, focusing on the implementation of the "*n_queens_solver.py*" First Proof of Concept Program and Third Proof of Concept Program in "*backtracking_algo.py*". Additionally, the progress made on the front end side of the development, highlighting the advancements in the Sudoku GUI as part of the Second Proof of Concept Program.

4.1.1 First Proof of Concept Program

The First Proof of Concept Program focuses on the N-Queens problem, a classic constraint-satisfaction puzzle with the objective to place N number of queens on an NxN chessboard, such that no two queens threaten each other (the queen can be moved any number of unoccupied squares in a straight line vertically, horizontally, or diagonally [Tur88]). This PoC showcases the technical integration and application of a recursive backtracking algorithm to systematically find all valid solutions.

The program begins with setting the '*numQueens*' variable, which determines the number of queens and the size of the chessboard, set by default to 8. There are two key functions inside the program '*isSafe*' and '*placeQueen*'. The '*isSafe*' function verifies if it is possible to place a queen on a given row and column without any conflicts. It iterates through all previously placed queens to check for conflicts and incorporates a logical assessment of any potential threats from other queens (columns and diagonals):

```
# iterate through all previously placed queens to check for conflicts
for row in range(0, testRow):
    # check for another queen in the same column
    if testCol == currentSolution[row]:
        return False
    # check for another queen on the same diagonal
    if abs(testRow - row) == abs(testCol - currentSolution[row]):
        return False
return True  # if no conflicts, it's safe to place the queen
```

The '*placeQueen*' is a recursive function that attempts to place queens on the board, iterating over columns and rows, and backtracking when necessary. This function updates two lists: '*currentSolution*', which keeps track of the current state of the chessboard, and '*solutions*', which collects all the successful arrangements of queens found.

```
for col in range(numQueens):
    # try placing a queen in each column of the current row
```

```
        if not isSafe(row, col):  # check if it’s safe to place the queen
            continue  # if not, skip to the next column
        else:
            currentSolution[row] = col  # place the queen
            if row == numQueens - 1:  # if all queens are placed
                # add the solution to the list
                solutions.append(currentSolution.copy())
            else:
                placeQueen(row + 1)  # recursively try to place the next queen
```

The output provided by the program is the total number of solutions found, each represented as a list where the index signifies the row and the value at that index indicates the queen’s column position.

4.1.2 Second Proof of Concept Program

The focus in the Second Proof of Concept Program is entirely on developing the main Graphical User Interface (GUI) for the Sudoku game, structured around the user stories and guided by the UML diagram.

The interface provides a logical path for the player to start a new game, choose from four difficulty levels and customise the application’s theme. Additionally, it allows easy interaction with the Sudoku grid, allowing for number entry, cell highlighting, erasing mistakes and thereby enhancing the puzzle-solving experience.

Features added to the system, include a ’verify’ button for error checking, hints for assistance, and a mistake counter to help players track their progress and improve their problem-solving abilities. It also includes options to pause the game and monitor playtime for added flexibility. Upon puzzle completion, players receive feedback on their performance and completion time, with time bonuses or penalty for those who solve puzzles without using the ’verify’ button, encouraging skill improvement.

4.1.3 Third Proof of Concept Program

The backtracking algorithm in the center of this project, the program and documentation for it, are designed and created to answer the following question: ’*How can I use recursive backtracking to find the best possible solution to very challenging Sudoku puzzles?*’

The backtracking algorithm involves a depth-first search through all possible puzzle configurations, systematically eliminating solutions that fail to meet the Sudoku’s constraints and backtracking to the previous position, when such an ”elimination” is reached. However, exploiting constraints to rule out certain possibilities for specific positions, prunes the search to the point where that Sudoku puzzle can be solved even by hand. Backtracking is the key to implementing exhaustive search correctly and efficiently. Simply explained, backtracking is a special approach to problem-solving that involves:

- depth-first search on an implicit list of configurations;
- exploring all possible solutions to a problem;
- eliminating those solutions that fail to satisfy the constraints of the problem;
- ’backtracking’ to a previous step and trying alternative solutions.

In this specific case, the technical implementation of this program is carried out in Python version 3.10.4. An example Sudoku puzzle is defined at the beginning as a 2D list with zeros representing empty cells. The algorithm's effectiveness is demonstrated by its ability to find the unique solution to this and other test puzzles. The program's output includes printing the original puzzle and the solved grid, showcasing the algorithm's successful implementation.

It initialises dictionaries to keep track of empty cells and their count in each row and column, which in the Sudoku puzzle is crucial for optimizing the backtracking process. By identifying the most constrained parts of the grid, the algorithm can start with these areas, making the process more efficient.

```
# sorting empty cell positions to prioritize those
# in rows or columns with fewer empty cells
# helps in prioritizing cells that are in rows or
# columns with fewer empty cells

sorted_positions = sorted
    (position.keys(), key=lambda x: remaining[x[0]] + remaining[x[1] + 9])
```

The '*valid*' function plays a pivotal role in this process by ensuring that each number, placed on the grid follows the Sudoku constraints. The function conduct three critical checks - row, column and 3x3 minigrid check. In the case, when all checks pass without finding a conflict, the function returns '*True*', signifying a valid placement. The most interesting of the three is the 3x3 minigrid check:

```
# checks the 3x3 box, in which the number is inserted
box_x = pos[1] // 3 # gives the x position of the 3x3 box
box_y = pos[0] // 3 # gives the y position of the 3x3 box

# iterates through the rows of the box
for row in range(box_y * 3, box_y * 3 + 3):

    # iterates through the columns of the box
    for column in range(box_x * 3, box_x * 3 + 3):
        # checks if the current number inserted is already in the box
        if grid[row][column] == num and (row, column) != pos:
            return False
```

The program's core function '*solve_sudoku*' is the heart of the puzzle-solving process, fills the grid by using iteration, placing numbers in valid positions and backtracking when a dead end is reached. This approach ensures that every possible configuration is considered, leading to the discovery of the puzzle's solution. The function takes the grid and a list of sorted positions of empty cells '*sorted_positions*'. The sorting is based on the strategy to prioritize cells in rows or columns with fewer empty cells, optimizing the backtracking process. The function iterates through numbers 1 to 9 and checks if each of these numbers can be validly placed in the first empty cell (determined by '*sorted_positions*'). When a valid number is found, it is placed in the cell, and the function is recursively called for the rest of the empty cells. This function showcases the essence of backtracking, methodically exploring and retracting steps to find a valid solution to the Sudoku puzzle.

```
for num in range(1, 10):
    # checks if the current number is valid for the current position
    if valid(grid, num, (row, col)):
        grid[row][col] = num    # if the number is valid, insert it

        # recursively call the function to solve the grid
        if solve_sudoku(grid, sorted_positions[1:]):
            return True # the puzzle can be solved with this number

        # the puzzle can not be solved with this number, backtrack
        grid[row][col] = 0
```

The Third Proof of Concept Program represents a significant step in the development process, showing the integration of a complex algorithm into a functional code. It also proves my progress in applying theoretical computer science concepts to practical problem-solving scenarios. The implementation highlights the power of backtracking recursion in solving similar to the Sudoku puzzles, requiring a methodical and exhaustive exploration of possible solutions.

4.2 Planned Future Programs

Following the discussed Aims and Objectives, the following subsections are dedicated to share my future development plans. They are not in incredible detail as I did not really have time to deeply research each of the topics.

4.2.1 Generating Sudoku Puzzles

The next step in the development, focuses on automated puzzle generation and difficulty categorisation. The approach chosen is to first create a fully completed Sudoku puzzle, from which a specific number of cells will be randomly removed to form the puzzle grid [Lee06]. The method ensures that each puzzle has only one, therefore unique solution. The difficulty of these puzzles will be categorized into four levels – Easy, Medium, Hard, and Extreme – based on the number of cells left empty. This categorisation is critical for appealing to a broad spectrum of players, from beginners to advanced. Furthermore, using a randomization technique in the puzzle generation, my goal is to make each gameplay unique, thereby maintaining player's interest and engagement over time.

The development of this feature will involve extensive testing to achieve the desired balance between challenge and correct category allocation, in line with the aim for a user-friendly and engaging application.

4.2.2 Sudoku Puzzle Validation

To enhance the in-game experience, the plan is to integrate a comprehensive Sudoku Puzzle Validation feature. The feature will provide real-time feedback, enabling the user to effectively assess their progress and strategies. Important to these functionalities is a validation option that allows players to compare their solutions with the correct answer, highlighting any errors. This will serve as a vital learning tool to improve problem-solving skills and logical thinking. Additionally, the system will track the number of times validation is used for each puzzle, adding a competitive element and supporting the players in monitoring their progress. Incorporating hints and complete solution on demand are another key aspects of the discussed validation feature. This functionality is crucial for maintaining the game's accessibility and enjoyment across different skill levels, offering assistance when players are challenged.

Ensuring accuracy and efficiency in the implementation of these validation features, especially in providing real-time feedback and hints, is critical in maintaining a seamless and user-focused interface. The development of these features will involve extensive testing to achieve the desired end result, supporting the player.

4.2.3 Machine Vision

The possible integration of Machine Vision in the Sudoku final application is a substantial step towards enhancing its capabilities and user interaction. The feature is designed to allow users to input new puzzles using their phone's camera. The aim is to create an algorithm proficient in scanning, interpreting, and inputting Sudoku puzzles from user chosen sources under different conditions.

This extension requires in-depth research in advanced image processing and pattern recognition technologies. The development of this feature includes evaluating various pattern recognition algorithms to ensure accurate identification of numbers and grid lines in diverse lighting and angles. The primary challenge is to achieve reliable scanning and interpretation of puzzles, requiring a thorough approach to algorithm development and testing.

Chapter 5: **Software Engineering**

This chapter delves deeply into the programming languages and software methodologies used and followed in the project, explaining the reason behind their selection. The focus is on the explanation of the programming languages and frameworks used, the design methodologies adopted, and the documentation approach.

5.1 Programming Languages & Frameworks

Python (current version: 3.10.4) was chosen as the main back end language for its simplicity, readability, and wide support for scientific computing and algorithm development[MG17]. Python's extensive libraries and community support make it ideal for implementing complex algorithms like backtracking, which is the foundation of the Sudoku solver.

For the front end, I selected ReactNative as the primary JavaScript-based framework [Eis15], as I already have some experience in developing with it. React Native offers a detailed and fairly recent documentation for developing mobile applications, allowing for a single codebase to be deployed on both iOS and Android (even though in this project I will focus on creating only an Android application). Its compatibility and resemblance with JavaScript, efficiency in rendering, and vast component library make it a preferred to me choice for creating a responsive and user-friendly interface for the Sudoku application.

The 'Expo' is an open-source framework that is designed to aid in the development of cross-platform mobile applications using React Native [MF17]. It simplifies the process of building, deploying, and quickly iterating on iOS, Android, and web apps from the same codebase. 'Expo' provides a rich set of tools and services, including access to native functionalities such as the camera, which is ideal for the possible Machine Vision extension of the project. The framework is particularly beneficial for projects like this, as it allows for rapid development and easy updates, ensuring a smooth and consistent user experience across various platforms.

5.2 UML design & User Stories

The use of Unified Modeling Language (UML) and User Stories in the project serve as a navigational tool for me, as a developer [DWT15]. UML is utilized for its effectiveness in visually representing the system's architecture, providing a clear and structured overview of all components and their interactions. This is a great support for me in maintaining clarity and coherence throughout the development process.

User Stories, on the other hand, are used to keep the development process user-focused and not forget the features I find exciting. Trying to imagine the requirements from the user's perspective, guide my development process towards features that align closely with the user's expectations. Following, this ensures me that my final year project and application are theoretically and technologically complex, as well as user-friendly.

5.3 Sphinx

Initially, I kept the Proof of Concept documentation in simple '.txt' files due to their straightforward nature and functionalities. However, I recently discovered the Python's documentation tool "Sphinx" [Bra10], my plan is to transition to this more structured format in the future development phases. Based on my research, "Sphinx" offers a more organized and readable format for documentation, supporting various output formats and enabling better management of technical documentation. This transition aims to enhance the maintainability and accessibility of the project's documentation.

5.4 Future Risk Assessment

Thinking ahead about the problems which can arise, I have identified five real concerns to me at this stage of development. I will not go into much detail, as I have already discussed every single feature and the problems that arise are logically connected to them.

1. Complexity in Machine Vision:

   The possible implementation of Machine Vision in the application, could bring complexities in the accuracy of the processing and interpreting diverse puzzle images (consider different lighting, backgrounds and angles). These possible problems will require extensive testing and refinement, therefore I will not start research and implementation before I have integrated successfully all of the main features.

2. Performance Optimization:

   As more features are integrated, there's a risk of decreased performance. Ensuring that the application remains responsive and efficient, especially on lower-end devices, is crucial. This risk is also directly connected to the Machine Vision extension.

3. Algorithmic Limitations:

   The backtracking algorithm, while effective, might face scalability issues or efficiency challenges with extremely complex Sudoku puzzles. My humble assumption is that I will have to rewrite the current algorithm at least one more time so that it produces fast results.

4. User Interface Consistency:

   Maintaining a consistent and intuitive user interface as the application evolves, especially with the addition of the Machine Vision extension feature, will be essential to ensure a positive user experience and also satify the developer (me).

Bibliography

[Apt03] Krzysztof Apt. *Principles of constraint programming.* Cambridge university press, 2003.

[BPS99] Sally C Brailsford, Chris N Potts, and Barbara M Smith. "Constraint satisfaction problems: Algorithms and applications". In: *European journal of operational research* 119.3 (1999), pp. 557–581.

[Bra10] Georg Brandl. "Sphinx documentation". In: *URL http://sphinx-doc. org/sphinx. pdf* (2010).

[BW12] Bruce Batchelor and Frederick Waltz. *Intelligent machine vision: techniques, implementations and applications.* Springer Science & Business Media, 2012.

[Civ13] Pinar Civicioglu. "Backtracking search optimization algorithm for numerical optimization problems". In: *Applied Mathematics and computation* 219.15 (2013), pp. 8121–8144.

[Dep23] Department of Computer Science, Royal Holloway, University of London. *Full unit project list.* 2023. URL: `https://projects.cs.rhul.ac.uk/List2023.php?PROJECT-TYPE=Full`.

[DHS10] Sean Davis, Matthew Henderson, and Andrew Smith. "Modeling Sudoku Puzzles with Python". In: *In practice* 7 (2010), p. 3.

[DWT15] Alan Dennis, Barbara Wixom, and David Tegarden. *Systems analysis and design: An object-oriented approach with UML.* John wiley & sons, 2015.

[Eis15] Bonnie Eisenman. *Learning react native: Building native mobile apps with JavaScript.* " O'Reilly Media, Inc.", 2015.

[FH+01] Martin Fowler, Jim Highsmith, et al. "The agile manifesto". In: *Software development* 9.8 (2001), pp. 28–35.

[Lee06] Wei-Meng Lee. *Programming Sudoku.* Apress, 2006.

[MF17] Eric Masiello and Jacob Friedmann. *Mastering React Native.* Packt Publishing Ltd, 2017.

[MG17] Andreas C Muller and Sarah Guido. *Introduction to machine learning with Python.* O'Reilly, 2017.

[RA03] Andrew Rollings and Ernest Adams. *Andrew Rollings and Ernest Adams on game design.* New Riders, 2003.

[RVW06] Francesca Rossi, Peter Van Beek, and Toby Walsh. *Handbook of constraint programming.* Elsevier, 2006.

[SUW18] Carsten Steger, Markus Ulrich, and Christian Wiedemann. *Machine vision algorithms and applications.* John Wiley & Sons, 2018.

[Tur88] Alan M. Turing. "Chess". In: *Computer Chess Compendium.* Ed. by David Levy. New York, NY: Springer New York, 1988, pp. 14–17. ISBN: 978-1-4757-1968-0. DOI: `10.1007/978-1-4757-1968-0_2`. URL: `https://doi.org/10.1007/978-1-4757-1968-0_2`.

[WH14] Baptiste Wicht and Jean Hennebert. "Camera-based sudoku recognition with deep belief network". In: *2014 6th International Conference of Soft Computing and Pattern Recognition (SoCPaR).* IEEE. 2014, pp. 83–88.

Appendix A: **Appendix & Personal Diary**

NOTE:
The "references" in the Personal Diary are completely different from the ones in the Interim Report. The diary was regularly updated over the past term, any changes to it can be seen in the GitLab history.
Most of the "references" refer back to the plan, submited in October. I will have to change that.

/Sprint 1

- During the first sprint I had the important task to write my Project Plan and present it to my supervisor. The plan is ready, as well as I am ready in terms of fully understanding the project requirements, objectives and the development path that needs to be followed.
- Started reading the "Introduction to machine learning with Python" [1] which is part of the reading list for the "Machine Language" model I am taking. The book will give me the foundations need to understand the code I will be writing for my project and the knowledge needed in my "Machine Language" model.
- Finished reading the "Modeling Sudoku Puzzles with Python" [2] article in which the modeling of Sudoku puzzles in a variety of different mathematical domains was discussed and simplified.

References:

- [1] Muller, A.C. and Guido, S., 2017. Introduction to machine learning with Python. O'Reilly.

- [2] Davis, S., Henderson, M. and Smith, A., 2010. Modeling Sudoku Puzzles with Python. In practice, 7, p.3.

/Sprint 2

- During Sprint 2, I focused on exploring in-depth the foundational references in the project plan, intently spending time on understanding the algorithms and technologies that were explained and presented. From the discussions on the Backtracking search optimization algorithm [1] to the complex nuances of constraint satisfaction problems discussed by Brailsford [2], each one of the recources provided me with valuable knowledge into problem-solving methodologies and computational techniques.
- Specifically, the frameworks presented in the "Solving the set cover problem and the problem of exact cover by 3-sets" [3], the elegance of Knuth's Dancing Links [4] also gave me a comprehensive understanding of solving puzzle-type problems. I also familiarized myself with the dynamics of stochastic optimization methods by K. Marti [5] and I had a quick overview of the "Agile software development models TDD, FDD, DSDM, and Crystal Methods: A survey" where I was happy to learn more about the various agile software development models and confirmed my decision to utilize the Agile development in the project.
- Furthermore, I briefly delved into the challenges of Sudoku puzzle modeling and its computational ratings by studying the works of R. Pelánek [7], T. Yato and T. Seta [8]. The practical approach with which some essential algorithms were illustrated by R. Stephens [9] using Python equipped me with a few practical strategies on how to approach the next steps in the project - writing the Proof of Concept Programs. Out of curiocity, I had some time to review the Eisenman's take on React Native [11], and I was happy to find out that there are more than valuable insights on how to successfully set the foundational stage for the GUI.
- Taken the time to read and learn something new before the start of the development phase, I've also started crafting user stories and am in the initial phase of sketching a user flow dia-

gram, giving a clear direction to the visual aspect of the application. The following /Sprint 3 will bring forward the culmination of these preparatory steps into real work. My immediate goals include finalizing the user stories, completing the user flow diagrams, and initiating coding for the first Proof of Concept program. Concurrently, I will begin laying the groundwork for the GUI, ensuring it aligns seamlessly with the technologies that I will use and with the user stories.

References:

- The references used are from my Project Plan, I am not going to add them to this file as they are simply reused here.

I have some questions which I will address with my supervisor:
- Do I want the application to give hints for next steps?
- How do I test the application's algorithms and their implementation while playing the game? Do I have to do this? Do I have to display some information?

/Sprint 3

01/11/2023, 11:10am - meeting with my supervisor, what I have taken from the meeting: Do I want the application to give hints for next steps?
- Yes, it should not be that hard and it is an important feature.
How do I test the application's algorithms and their implementation while playing the game? Do I have to do this? Do I have to display some information?
- Test the implementation of those algorithms manually during the development process. Focus on the outcome and implementation (technologies used, time under dev etc.) in the final report.
- It is a good idea to send my presentation and my report to my Supervisor before submitting it for grading.
- Start with the implementation of the 8 Queens problem, continue with implementing the Backtracking algorithm.

- I worked on developing and documenting the first Proof of Concept (PoC) program, a classic computational problem. The program is written in Python and utilizes a recursive backtracking algorithm to find all possible solutions for placing N queens on an N×N chessboard without any two queens threatening each other. Each solution is represented as a list of N tuples, indicating the precise positions of the queens on the board.
- In the process, I created a comprehensive report detailing the program's functionality, usage, and configuration. Documented explanation of the algorithm can be found here [1]. The report serves as a user guide, providing clear instructions on how to run the script, set up the necessary configurations, and understand the output.
- Reflecting on today's work, I am satisfied with the progress made and the depth of analysis provided in the report. The documentation is detailed and well-structured, ensuring that any future users or developers can easily understand and utilize the program. I will continue with the development of the second PoC program, which will look into the backtracking algorithm's application in solving the Sudoku puzzle.

References: - [1] file "*doc_n_queens_solver.txt*" in the "PoC - N-Queens puzzle" folder.

/Sprint 4

- Started working on the second Proof of Concept (PoC) program, which will focus on the application's GUI. The initial plan to create a simple, intuitive, and user-friendly GUI will be followed so that the frontend will be mostly completed and will not require a lot of changes in a later stage of development. The user interface will be written entirely using the React Native framework, and the application will be tested on Android devices.

There is more information regarding the GUI in the "PoC - Program2" folder, more specifically in the "*doc_simple_GUI.txt*" file. Here is a list of what I will strive to achieve in the following week:

Home screen
- slide bar to choose from different difficulty levels - DONE
- button to choose different theme
- button to start a new game - DONE
- button to continue from previous game
- button to view the history of games, redirect to History - DONE

Game screen
- the grid of 9x9 cells and 1-9 numbers displayed as buttons under the grid - DONE
- button to validate the solution - for now the button will be inactive
- button to clear the selected cell - DONE
- button to clear the entire grid
- button to return to Home screen - DONE

About screen
- displays information about the application
- button to return to Home screen

Instructions screen
- displays instructions on how to solve a Sudoku game and how to interact with the application
- button to return to Home screen

- I also reconsidered the user stories and I have made the required changes. Changes were needed as the previous file was not structured properly and it was not focused only on the user stories for the application.

I fell quite ill on Friday, 11th November 2023, I was supposed to complete the described components in the previous week, but I was not able to do so. I will try to complete them in the following week.

I am still working on the GUI but decided it is time to implement the backtracking algorithm and therefore I will start working on the third Proof of Concept (PoC) program in a separate /thirdProgram branch.

The work over the next two weeks (20-26th and 27-30th November) will be considered as part of Sprint 4 and it will be focused on the following:
- Implementing and documenting the backtracking algorithm in the third PoC program.
- Completing the GUI pages for the second PoC program.
- Creating a presentation and sending it over for review to my supervisor - before the 24th November.

/Nov 21, 2023

After careful research on the backtracking algorithm I came up with implementation which unfortunately does not provide the desired results. I will have to do more research and try to find a solution to the problem. For now, I will keep this program as a stepping stone for the next - corrected implementation of the algorithm. The *first_backtracking_algo.py* is in the "PoC - Program3" folder and it is documented in the "*doc_first_backtracking_algo.txt*" file. I will continue working on the GUI for the second PoC program.

I also realised that the README.md file is not up to date and I will have to update it with the instructions on how to run each PoC program. I will do this in the following week.

/ Nov 23, 2023

As I mentioned on the 21st of November, I will keep the *first_backtracking_algo.py* as a stepping stone for the next - corrected implementation of the algorithm. Today, I believe to have found the solution to the problem and I have implemented it in the *backtracking_algo.txt* file in the "PoC - Program3" folder. I have thoroughly documented the algorithm in the "*doc_first_backtracking_algo.txt*" file. I will continue working on the GUI for the second PoC program.

Notes: In the following weeks, I will write a program to generate Sudoku puzzles inside the future application. The program will generate a random Sudoku board where all numbers are filled in and then it will remove some of them to create an unique puzzle. This will ensure that the puzzle always has a solution. BUT! Making sure that the generated puzzle has exactly one solution will be a bit more challenging as I must leave at least 17 numbers for a 9x9 Sudoku (Game theory).

/ Dec 07, 2023

I have completed a simple "demo" GUI for the future application. The GUI is written in React Native and it is tested on Android devices. The GUI is not fully functional, but it is a good starting point for the future development of the application. I have also updated the README.md file with instructions on how to run each PoC program.