A Multiplayer Sudoku App

Scott Adamson

Fourth Year Project Report

School of Informatics

University of Edinburgh

2023

Abstract

This project explores the development and implementation of a multiplayer Sudoku game, SudokuSlam, that transforms the traditionally solitary puzzle solving experience into a competitive race between two players. The game, built for Android with the Unity game engine, allows participants to engage in real-time battles to complete Sudoku puzzles, fostering a dynamic and interactive environment that redefines the classic gaming experience. A key feature of this app is the incorporation of an Elo rating system, which effectively tracks and ranks players' performance, facilitating fair and exciting matches between individuals with similar skill levels. Moreover, the game's robust grid generation algorithm is capable of producing nearly 2 billion unique and valid Sudoku grids, ensuring an endless variety of puzzles that challenge players' logic and strategy. The dissertation examines the design principles, technical implementation, and user experience of this innovative multiplayer Sudoku game, demonstrating the potential to revolutionize the way we approach and enjoy this popular puzzle genre.

Word count: 10515

Acknowledgements

I would first like to thank my supervisor, Nigel Topham, for proposing and overseeing this project, as well as my family and friends for the endless support they have given me throughout.

Table of Contents

Chapter 1

Introduction

Sudoku is a hugely popular logic-based puzzle that has long captivated puzzle-solvers and has become one of the all time classic conundrums. The puzzle's rise to popularity began in a French magazine in the late 19th century [33] and since, the game has seen many iterations and refinements over the years. [21] Today, the classic rules of 9x9 Sudoku, first given its name in Japan - meaning "the digits must remain single" [58, 30] - is said to have at times sold around 4.5 million Sudoku puzzle books a year in Japan alone. [43]

The goal of this project is to design and build a mobile app for Multiplayer Sudoku. This report provides incite into my take on this problem which outlines an Android solution for creating a diverse Sudoku puzzle set and mapping the typically one-player Sudoku experience to a two-player game. This game intends to closely stick to the design constraints of a well-formed Sudoku and puzzle set while introducing some innovative features to the online Sudoku space.

My solution was developed with the Unity game engine [24] and its design has been informed by the existing background material in the Sudoku research space as well as the current players serving the puzzle solving community. Before I discuss how this implementation worked in practise, I will outline the rationale behind some of the concerns which I addressed early on in the development stages, as well as detail a requirements specification of the app. The evaluation will then review how well the project meets its objectives, which include generating a sufficiently large and diverse set of valid Sudoku's and implementing a reliable matchmaking system. The report will conclude by summarising its achievements as well as discussing the potential en-

UN PROBLÈME PAR JOUR

5129. — CARRÉ MAGIQUE DE 9 A COMPARTIMENTS ÉGAUX

Composé par M. le commandant Cocoz

Compléter le carré ci-dessous en employant les 81 premiers nombres de manière que les neuf carrés composés de neuf nombres donnent chacun, comme total, 369. De plus, chacune des deux grandes diagonales, chaque horizontale et chaque verticale du carré complet devra donner 369.

17	20	5	75	67	79	51	36	39
70	73			42			8	11
45		33	2		26			64
19		16	11		29	63		78
47	32			81			10	22
75		72	13		1	35		50
59			27		15	37		34
6	18			38			77	62
31	43	46	80	56	68	12	24	9

Figure 1.1: One of the first ever printed Sudoku variations

hancements and future applications of the game.

Chapter 2

Background

This background chapter intends to give an introduction to the concept of Sudoku puzzles and discuss the landscape of the literature which supports. I will discuss the design constraints for Sudoku puzzles, as well as the various works on generating, solving and grading these puzzles. I will conclude by introducing some of the existing works in the field of Sudoku game development and discuss how these can be used to inform our design.

2.1 Sudoku Background

A Sudoku puzzle consists of a 9x9 grid of cells, divided into groups of rows, columns, and blocks (smaller 3x3 sub-grids) as shown in figure 2.1. [41] Each cell is filled with a symbol ranging from 1 to 9, and the puzzle is created by removing some of the cells' symbols. The remaining non-empty cells are known as givens. [59] The objective of the puzzle is to fill in the missing grid squares while satisfying the constraint that each number must only appear once in each row, column, and block. The simplicity of these rules, and yet its extremely challenging nature, is perhaps the reason this tricky combinatorial optimization problem has become so popular around the world [56] and has given rise to many efforts to fully understand the game of Sudoku, from both a psychological and mathematical standpoint. [48, 55]

2.1.1 Grading

Sudoku has the ability to cater to all skill ranges through its various difficulty ratings, which help categorise puzzles and allows players to measure the difficulty of the chal-

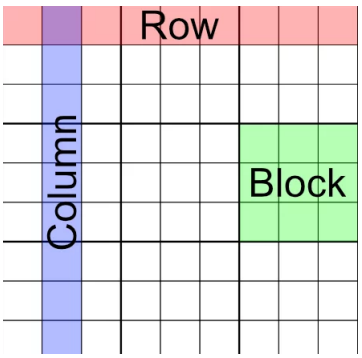


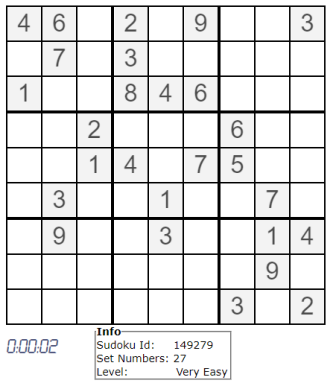
Figure 2.1: Sudoku grid components [41]

lenges they're facing. Andrew Stuart describes the grading of a Sudoku as "the greatest concern of the puzzle maker." [57] Stuart's statement is armed with good reason, "If too many people disagree with your grades then you are clearly going to lose your audience," he says. From beginner to expert level, these ratings allow players to play puzzles suited to their solving ability, which makes Sudoku an extremely accessible puzzle which can be fun for everyone.

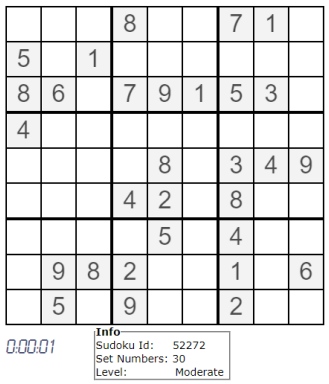
These difficulty levels are typically labelled using some variation of very-easy to very-hard puzzle ratings and while there are no standardised rules in place for determining a puzzles difficulty [49], there are a few commonly used techniques.

The more naive approach to puzzle grading is to count the number of clues given in the puzzle. While this method provides a fast way to arrange Sudoku's in terms of difficulty, with a positive correlation between difficulty and number of givens certainly existing [51, 28], this is not a foolproof way of ensuring the integrity of the grade [56] as there are many exceptions to this rule. For example, consider the Sudoku's in Figure 2.2. 2.2a shows a Sudoku graded "very easy" with only 27 givens, whereas 2.2b shows a 'moderate' Sudoku, two difficulty levels higher, with 31 given cells. [32]

The most commonly employed method of grading Sudoku's is by measuring the difficulty of the solving techniques required to solve the grid. Sudoku puzzles can be solved using a variety of different solving techniques, some more advanced than others. By allocating a difficulty rating to a set of common solving techniques, we can create an effective measure of a puzzles difficulty based upon the techniques required to solve it. [49, 48] Basic techniques, such as naked singles [8] and hidden singles [4], are more commonly employed in easier puzzles. In contrast, more advanced tech-



(a) A 'very easy' Sudoku with 27 filled cells



(b) A 'moderate' Sudoku with 31 filled cells.

Figure 2.2: A very easy Sudoku with less givens than a moderate Sudoku [32]

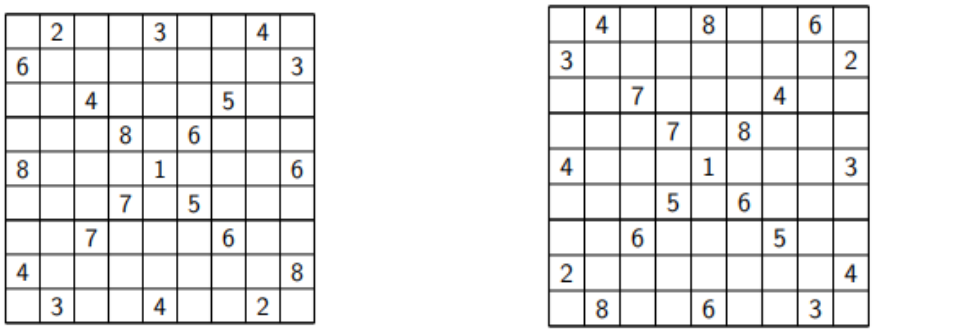
niques, such as X-Wing [29], Swordfish [23], and single's chains [14], are more likely to be necessary for solving harder puzzles. [13]

In a multiplayer context, understanding how grading systems apply to traditional Sudoku formats and potentially translate into our desired multiplayer context can help ensure fair matchmaking and a well-balanced game environment.

2.1.2 Symmetry

The symmetry of a Sudoku puzzle is considered to be an important feature by most, with some publishers, such as Japan's Nikoli, even stipulating that a Sudoku *must* be symmetrical in order for it to be regarded as valid. [9] However, it is generally accepted that this is purely for aesthetic purposes, a preferred convention of Sudoku makers which achieves a certain level of visual attractiveness of the puzzle. This has been an important factor in Sudoku amassing such a huge audience, Nikoli claims.

Having symmetrical Sudoku's also means that we can perform various rotations and reflections on a grid and still maintain a symmetrical, well-formed Sudoku. Consider Figure 2.3, which shows a valid Sudoku grid before and after a 90 degree rotation. The total number of possible symmetries which can be applied to a single puzzle to create a new 'distinct' grid, was determined to be 8. [53] This could prove useful, as it would allow us to generate a larger number of grids from a single base puzzle. [20]



(a) A symmetrical Sudoku (b) The same Sudoku rotated 90 degrees

Figure 2.3: A valid Sudoku before and after 90 degree rotation [53]

2.1.3 Uniqueness of Sudoku Solutions

A key element of a well-formed Sudoku is its unique solution. [36, 1] In order for a Sudoku puzzle to not present any ambiguous logical crossroads, which would significantly hamper the puzzle-solving experience, a single solution is required to guarantee that one logical and coherent line of reasoning can deduce the puzzles' answer.

The uniqueness property of a Sudoku is vital because it ensures that the puzzle is solvable using pure logic and deductive reasoning, without the need for guesswork or trial and error. If a Sudoku puzzle has multiple solutions, the solver would be unable to determine which solution is correct based on the information provided by the initial grid, and the process of solving the puzzle would become frustrating and less enjoyable. In order to guarantee the uniqueness of a Sudoku puzzle, creators and generators must carefully monitor the Sudoku's number of solutions when creating the puzzle grid.

2.1.4 Note Taking

Note taking in Sudoku, shown in Figure 2.4, often referred to as ”pencil marks” or ”candidates,” is where solvers jot down the possible numbers for each grid square in the corner of the cell. This technique helps players identify patterns, eliminate possibilities, and make logical deductions more efficiently. As the complexity of Sudoku puzzles varies, so does the need for note-taking. For more straightforward puzzles, experienced solvers may not require extensive note-taking. However, when tackling more challenging puzzles, even seasoned players rely on this technique to navigate intricate logical paths and reach the unique solution. Some solving techniques even

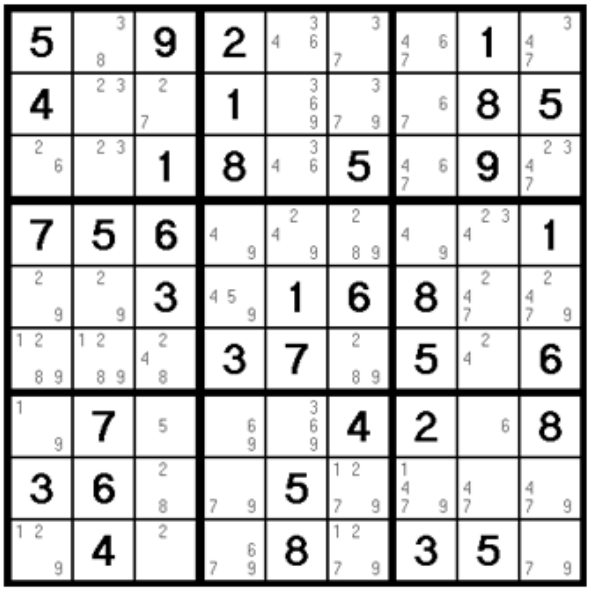


Figure 2.4: A Sudoku grid with notes [47]

rely upon note-taking. [10] This feature is employed by many of the most popular online Sudoku applications [11, 17, 25], meaning this is likely expected functionality from a Sudoku solving audience. By incorporating note-taking techniques, players can streamline their solving process, reduce errors, and ultimately, enjoy a more satisfying and engaging experience with Sudoku puzzles.

2.2 Valid Sudoku Generation

Puzzle generation is an import aspect of Sudoku as it ensures players have access to a diverse set of challenging problems. Although there are many ways to generate a valid Sudoku grid, the most popular of techniques operate with a similar approach. This involves continually modifying a complete puzzle until it possesses some desired property, as defined by the desired difficulty of the puzzle as well as the validity constraints of a well-formed Sudoku. The solution described and employed by the popular SudokuOfTheDay website is a perfect example of this concept in practice. [5] It goes as follows:

1. First we start by generating fully complete 9x9 grid that matches design constraints of a Sudoku.

2. Pairs of numbers are removed in such a way that some form of symmetry is

maintained until we have removed 15-20 pairs, the commonly expected minimum number of blank cells to solve.

3. Once we reach the minimum threshold for a Sudoku we continue to remove pairs of cells, testing for existence of the properties we desire of our puzzle after each iteration

4. Once these conditions are met, we have created our Sudoku puzzle.

If we consider a general Sudoku grid with N empty cells. For each empty cell, there are at most 9 possible options (numbers from 1 to 9). The number of potential solutions can be computed with 9^N. This means that a Sudoku with only 30 givens would have a colossal potential solution set of 9^{51}, or 3.24×10^{47}, grids. As each Sudoku puzzle with some cells removed has a huge number of potential solutions, we develop the need to automate the Sudoku solving process so as we can generate a sufficiently large data-set of valid Sudoku's.

In practice this becomes an issue when removing clues from a fully-complete Sudoku grid as there may arise a situation where more than one of the many possible solutions are valid answers to the puzzle. For example, removing cells at random from the valid Sudoku solution grid shown in Figure 3.3a results in a puzzle grid with 4 valid solutions, as shown in 3.3b. This means that taking this approach makes it is necessary to validate each grid with the uniqueness property after each cell removal. In the absence of an efficient Sudoku solver, it would make it impossible to say whether any given Sudoku has a unique solution within any beneficial amount of time. This has been the motivation for many different approaches to solving the game of Sudoku computationally, which has only strengthened by Sudoku being considered an NP-Complete problem [54]. This means if an efficient algorithm is found for any NP-complete problem, it could be used to solve all other problems in the NP class efficiently, fueling the desire for a polynomial time solution to this problem.

Some of the most notable Sudoku solving techniques include the widely used backtracking approach; genetic algorithms; and Donald Knuths algorithm X. Each of these methods require a small adaptation in order to validate solution uniqueness.

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

(a) A valid sudoku grid before cell removal

7	2	123456789	1	123456789	123456789	4	123456789	3
123456789	123456789	3	123456789	123456789	5	123456789	7	123456789
123456789	123456789	1	123456789	7	123456789	123456789	123456789	6
123456789	123456789	2	123456789	4	8	1	123456789	5
1	123456789	6	5	123456789	123456789	123456789	4	123456789
123456789	4	123456789	123456789	1	123456789	123456789	123456789	9
123456789	3	123456789	123456789	123456789	123456789	7	123456789	2
8	123456789	123456789	7	123456789	2	123456789	123456789	4
123456789	5	7	123456789	3	123456789	123456789	1	123456789

(b) Grid after random cell removal

Figure 2.5: A valid complete Sudoku grid with cells removed at random with 4 possible solutions [19]

2.2.1 Backtracking

The recursive backtracking algorithm, visualised in Figure 2.6 is considered the simplest of our efficient Sudoku solving contenders [34], avoiding a certain level of computational and implementational complexity which may arise in other approaches. [37] This method continually visits every empty cell in the grid and places any of the symbols which satisfy the current constraint parameters of the cell. If the algorithm arrives at a cell with no available symbols to choose from, it will return to the previous cell and select another option. The exhaustive nature of this method means that the brute force algorithm is effective in providing a logical solution to any puzzle - given enough time, as its time complexity is exponential. [44]

2.2.2 Algorithm X and Dancing Links

The ”Dancing Links” of Donald Knuth [45] describes the use of his Algorithm X for solving exact cover problems and is a popular and effective method for sophisticating the heavy-footed backtracking approach. This algorithm gets its name from its ”node dancing” techniques to remove and restore links between nodes in a doubly linked list, resulting in an extremely efficient method which can be used to explore all of the possible solutions to a Sudoku. [42]

Sudoku grids are converted into a sparse matrix, with each row representing a possible placement of a number in a cell, and each column representing a constraint. The Dancing Links algorithm then systematically searches through possible combinations using depth-first search and backtracking to find a solution that satisfies all constraints,

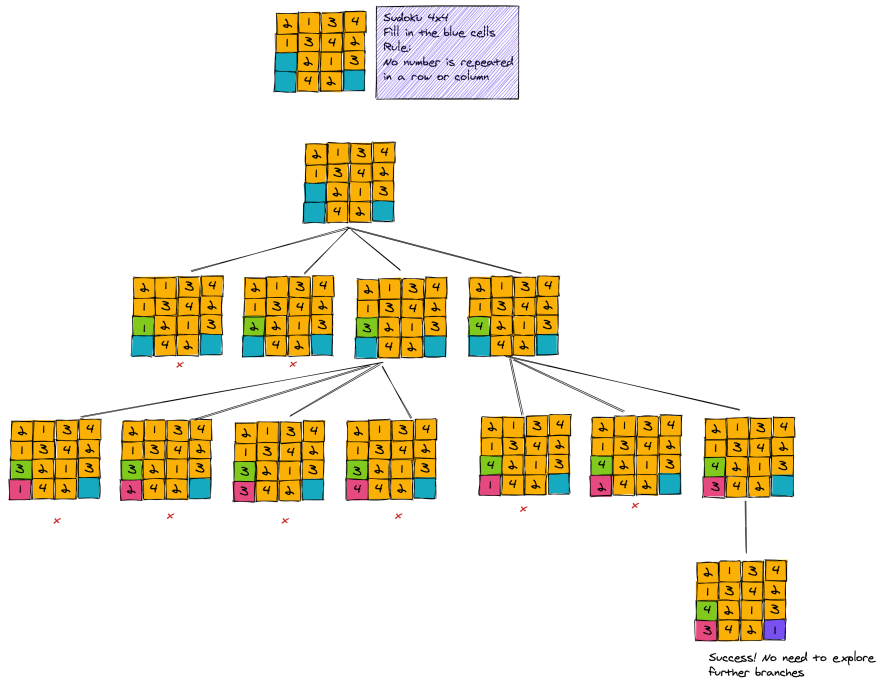


Figure 2.6: The backtracking algorithm in action

effectively solving the puzzle. This is one of the most efficient and powerful methods for solving Sudoku puzzles.

2.2.3 Performance Analysis

With there being so many ways of determining a Sudokus solution efficiently, its helpful to analyse the performance of some common methods, which has been done in the works of Chatterjee, Paladhi and Chakraborty. [34] This study compares the performance of 5 different Sudoku solving methodologies across 30 random Sudoku puzzles of varying difficulty. The research concludes that the brute-force approach performs extremely well, outperformed only by a graph-referencing algorithm which also employs backtracking-based methodology, but improves upon this by pruning some unsuccessful search branches within the search space, similarly to the Dancing Links algorithm, which is not measured within this study. Harmony search algorithms performed the worst by far, particularly with the more difficult puzzles. The longest time taken to solve any of the Sudoku's for any of the top 3 performing algorithms was 1.1 seconds and the key takeaway from this analysis is that there are a number of viable solutions for implementing an efficient Sudoku solving algorithm.

2.2.4 Challenges in Implementing Sudoku Solving Algorithms

Despite the undeniable viability of these solutions, all of which capable of producing run-time-worthy solutions, implementing these approaches presents a set of challenges which warrants exploration of other avenues.

One of the primary challenges in implementing Sudoku solving algorithms is balancing scalability and efficiency. Some algorithms work well for smaller grids but struggle to perform optimally on larger, more complex puzzles. Ensuring that an algorithm scales well and maintains reasonable performance for a variety of puzzle difficulties can be challenging. On top of this, some Sudoku solving algorithms can be quite complex, making them difficult to implement, understand, and maintain. Many Sudoku solving methods, such as probabilistic algorithms, such as Genetic Algorithms and Simulated Annealing, also do not guarantee that they will find a solution in every case, as their success depends on the quality of the initial population, the selection of the parameters, and the random nature of the search. In some cases, these algorithms might get stuck in local optima and fail to reach the global optimum or a valid solution. [61] These issues will need to be taken into consideration when deciding how best to generate the dataset for this project.

2.2.5 Transformations

As previously hinted at, an inventive and effective alternative to generate a large and diverse puzzle set is to perform various transformations upon pre-existing valid Sudoku grids. This method is first described in the works of Russel and Jarvis [55] who outline all of the possible transformations which may be performed on a Sudoku grid while preserving its validity in order to determine the number of essentially different Sudoku's. The paper concludes that only $5,472,730,538$, or less than 0.000000000082% [40], of the initial puzzle set are essentially different from one another, meaning almost all of the possible valid Sudokus can be produced as a transformation of some other valid Sudoku grid.

By applying transformations, such as swapping rows and columns within blocks, we can produce a large set of valid Sudoku puzzles in an extremely time efficient manner and at the same time preserve the difficulty of the original puzzle. [52] Using transformations such as these, we can produce an extremely large set of puzzles:

1. Permuting the three stacks. There are 3! (6) ways to permute the three vertical 9x3 stacks.

2. For each band, there are 3! (6) ways to permute the rows within it. Since there are three bands, we have $6^3 = 216$ possible permutations.

3. Permuting the three columns within a stack: For each stack, there are 3! (6) ways to permute the columns within it. Since there are three stacks, we have $6^3 = 216$ possible permutations.

By combining these methods, we can generate a large number of unique puzzles based on a single seed puzzle. Specifically, the total number of possible permutations using the transformations above would be $6 \times 216 \times 216 = 279,936$. To go a step further, if we were to have $N = 10$ number of puzzles for each of 4 levels of Sudoku difficulty levels, we would have an immense $10 \times 2457216 \times 4 = 11,197,440$ puzzles. One final transformation we can apply which drastically increases the possible number of Sudoku grids is permutation of the symbols in the grid. In the permutation process, each number in the grid is directly swapped with another number from a shuffled list of symbols. The same pair of numbers being consistently swapped throughout the entire grid maintains the structure and validity of the Sudoku puzzle while creating a new, distinct variation of the original seed grid. There are 9! (362,880) ways to permute the symbols (numbers) in the grid, which skyrockets the total possible number of puzzles. Further permutations suggested in these works which could further increase the potential puzzle set include reflecting the grid along the horizontal or vertical axis and performing rotations of the grid. These transformations would maintain the puzzle's structure and solvability, while offering new orientations and appearances. By incorporating these additional permutations, the number of distinct puzzles generated from the seed puzzles would grow, further enhancing the variety of challenges available to players and ensuring a continuously fresh and engaging Sudoku experience.

Although the total number of distinct puzzles which may be generated from initial seed puzzles is still significantly lower than the $6.67x10^{21}$ total number of possible Sudokus calculated by Felgenhauer and Jarvis [40], a puzzle set f distinct Sudoku puzzles in the hundreds of millions and beyond represents a diverse puzzle set capable of providing a vast array of challenges across many difficulty levels. This immense number of transformations ensures an engaging and challenging experience for players

while addressing a number of the challenges presented with other Sudoku generating routes.

2.3 Chess

Online Chess has seen a meteoric rise in popularity in recent years, regularly breaking its own huge daily online player records, with the number-one online chess-hub, Chess.com, recently reaching over 10 million daily users. [35] Though their gameplay and techniques differ, Sudoku and chess share a common ground as intellectual board games that test players' problem-solving ability and encourage mental agility. The success of Chess as an established multiplayer board game which has translated well into digital format offers valuable insights into the design of an engaging multiplayer experience. Some of the defining features of today's online chess experience contributing to the widespread appeal of the game include a robust rating system known as the Elo system, as well as Chess.com's wide array of personalisation options. Better understanding the causal elements of Chess's success could help pinpoint features which could be adapted to inform the development of a captivating Sudoku experience.

2.3.1 The Elo Rating System

Named after its inventor, Arpad Elo, the Elo rating system [38] is a commonly used method to determine the relative skill of players in two-player games, such as chess. It has become a key feature in competitive and online chess, giving players an opportunity to play games with those of a similar ability as well as providing a measurable benchmark against which to improve, encouraging a culture of constant learning and improvement. In 1970, the World Chess Federation (FIDE) officially adopted the Elo system [3], and it has remained standard for chess rankings worldwide since. The system assigns players with an Elo rating, a numerical value representing their skill level. The difference in rating points between two players predicts the likely outcome of their match. In general, a higher-rated player is predicted to do better versus a lower-rated player. The outcome of a match-up between two elo-rated players results in each players rating changing. Winners receive rating points, while losers lose them. The number of points gained/lost is determined by the difference in player rating between the players and the outcome of the match. This system provides a great degree of accessibility and promotes a competitive culture, making it worthy of keeping in mind during the

development process of a multiplayer Sudoku game.

2.3.2 Personalisation

Billion dollar management consultancy Mckinsey & Company states that personalisation has never been more pivotal in user engagement and that ”consumers now view personalisation as the default standard for engagement.” [31] Popular online chess platform, Chess.com, offers a wide range of customisation options which caters to the diverse needs and preferences of its large user-base. The features offered by Chess.com, such as custom board appearances, time controls and sound effects could help the development of a multiplayer Sudoku app deliver an engaging user-experience better equipped to captivate a wider audience and foster a more enjoyable gaming experience.

2.4 Existing Applications of Multiplayer Sudoku

Sudoku assumes many forms and performs well in a range of mediums, such as paper format, app games and online websites. While Sudoku is typically a single-player format game, there are some existing players already developing in the multiplayer Sudoku field. Several platforms focus on competitive multiplayer formats where players can challenge one another to complete puzzles in the shortest time or with the highest accuracy. [18, 6]

In addition to competitive gaming, several platforms offer collaborative Sudoku experiences, where users can work together to solve puzzles. [22] A common theme throughout a lot of multiplayer Sudoku games and even classic Sudoku games is that some platforms allow users to create and share Sudoku puzzles, providing an additional layer of engagement within the multiplayer experience. [16, 15]

Chapter 3

Design

In this section, I will outline some of the main design decisions I made during the early stages of developing and implementing our online Sudoku game.

3.1 Unity and C#

The Unity game engine is a well-established choice in the game development industry and provides an interface upon which we can build a scalable, interactive multiplayer Sudoku solution. Unity provides a user-friendly interface with a plethora of assets within its asset store to support the necessary networking functionality of the game. This engine is a cross-platform development environment paying credence to the scalability of the project and its potential ability to reach a more diverse audience. Along with its vast community of developers, Unity makes for an all-round good decision to enhance the development process.

Unity makes use of C# which offers developers a powerful scripting language for building interactive games. I had no prior experience with C#, luckily however, it is known for being simple to pick up. [27]

3.2 Mobile Platform

The project proposal of this Sudoku game leaves the choice of operating system (Android or iOS) open-ended. I made the decision to develop solely with Android in mind as it is generally considered more open and flexible for developers [46] as well as having a significantly larger user base. [60] It is worth keeping in mind that despite

developing for android, the choice of Unity as our engine makes adapting our solution to iOS a much easier process, making for a more scalable solution.

3.3 Photon

The Unity asset store provides access to the widely-used third-party plugin 'Photon 2' [12] which offers reliable management of the networking functionalities of the game. Photon is built to handle a large number of concurrent players and can handle a growing player-base. Unity offers a free tier plan that supports up to 20 concurrent users (CCU) per app. This limit is generally sufficient for initial testing and small-scale projects, but paid tiers can support thousand of players and is available at any point in the future.

The integration process includes installing the PUN 2 package, setting up a Photon account, configuring PUN 2 within Unity using the app ID and creating a Photon network manager in the Unity scene. The Photon servers also allow us to keep track of information about lobbies as well as custom room data in the servers memory. The server can also handle matchmaking and filtering meaning this could be useful in the implementation of a rating system. Overall, its usability, flexibility, scalable low-latency networking solution makes for a sound choice to handle the key networking functionalities of this project.

3.3.1 Using Photon's Custom Room Properties

One of the key features of Photon is its Custom Room Properties, which can be utilized to store and share game-specific data, such as player scores, nicknames, and puzzle grids. [2] Custom Room Properties are key-value pairs associated with a game room. These properties are synchronized across all connected clients, ensuring that every player in the room has access to the same data. We can define which properties are available and how they can be modified, allowing for fine-grained control over the shared data. By storing shared data here we have the power to implement more interactive features, such as displaying player progress bars, nicknames and Elos on the UI.

3.4 Player Ratings

As previously discussed, the Elo-rating system is a widely adopted solution among competitive games due to its efficiency and simplicity. In order to develop an engaging and competitive user experience with a balanced matchmaking system, adopting the Elo system seems an obvious choice. This system is based on the idea that the difference between two players' ratings can be used to calculate an approximate win probability for either player in a given match. The Elo algorithm calculates the expected outcome for each player using the following formula, where $R_b - R_a$ is the difference in player rating:

$$E_a = \frac{1}{1+10^{\frac{R_b - R_a}{400}}}$$

After the completion of each match-up, both players elo-ratings are modified based on the expected and actual outcome of the game. The new-elo algorithm takes into account a constant factor, K, representing the maximum possible rating change after a match. A moderately high K-factor allows initial player ratings to quickly adjust to accurately reflect their true potential, which is how FIDE justify their maximum K-factor, reserved for new and young players, of 40. [39]

$$NewRating = OldRating + K \times (ActualOutcome - ExpectedOutcome)$$

The actual outcome of a match is 1 for a win, 0 for a loss. I have chosen to use a relatively high K factor, 32, which allows for larger rating adjustments after each game. This means that players will adjust to their true skill level faster [50], which can be especially helpful when starting a new gaming community or when players are still learning the game. A more comprehensive K factor should be considered to be variable as users play more games, but this is a potential application for the future and not in the purview of this work.

I made the decision early on to store player Elo ratings locally using Unity's PlayerPrefs for the ease of access and simplicity it offered during the development process. However, I acknowledge that this decision effectively limits the implementation of some potential multiplayer features, such as global leaderboard functionality. While this feature was not a core requirement for the current scope of the project, it remains a valuable potential future enhancement. To enable leaderboard functionality in a future iteration, a server-based solution can be employed to store and manage Elo ratings

centrally, allowing for performance tracking across the entire user base which can be made visible to all users and aid in creating a community feel to the app.

3.5 Sudoku Generation

Initially, the primary goal of this project was to design an algorithm capable of efficiently generating valid Sudoku puzzles at run-time. The benefits of a complete solution like this are a huge range of potential puzzles. In the early stages of implementation it became clear that this methodology comes with the need for very efficient design in order to create a solution capable of guaranteeing puzzle validity for our a real-time Sudoku app. Upon further research, a discussion of the uniqueness of any given Sudoku puzzles from SudokuOfTheDay [5] made it clear the potential of leveraging existing valid puzzles and performing transformations on the grids. This approach allows us to generate a large number of unique puzzles of varying difficulty and complexity without sacrificing computational efficiency. This innovation proved to be a practical solution to overcoming the design challenges faced in the implementation of a solution counting algorithm.

Transformations such as these do not change the underlying logical structure of the puzzle. Therefore, the transformed puzzles can be solved using the same logical steps as the original puzzles, preserving their difficulty levels, meaning there is no need to implement a complex difficulty rating system.

The possibility of generating the puzzle set prior to run-time was also considered, which would help deal with the computational load during game play. However, this presented a host of data management concerns and did not seem capable of ensuring the potential puzzle set be diverse enough. The transformation approach offered several advantages over this solution, such as efficiency as well as a significant advantage in storage requirements.

3.5.1 Data Collection

In order to collect a base puzzle set of Sudokus to perform transformations upon, a diverse set of puzzle grids and their solutions was gathered from an online Sudoku generator. [32] The puzzles were originally ranked very-easy, easy, moderate and

difficult. I have chosen to refer to the 'difficult' grading as hard for the rest of this report. To ensure the quality and accuracy of these puzzles, each one was individually validated by hand using a separate reliable online Sudoku solver. [19] This manual validation process not only confirmed the validity of the grid, but also validated the uniqueness of the puzzle's solution, as well as its difficulty. By guaranteeing our seed puzzle set possesses the desirable qualities of a well-formed Sudoku, we extend these properties onto all future transformations of these grids. This data-set is stored as static lists of 2D integer arrays for each difficulty level, and you can access them using the classname SudokuGrids followed by the difficulty name and the index of the grid you want. For example, to access the first easy grid, you would use SudokuGrids.Easy[0]. Since the dataset is relatively small, it does not significantly impact the memory usage or performance of our program storing it in this manner. This is yet another benefit of performing transformations on a small set of puzzles to generate our Sudoku's. Consequently, this curated seed puzzle set serves as a robust foundation for our multiplayer Sudoku app, ensuring an engaging and challenging experience for participants while negating the possibility of ambiguous or invalid Sudoku scenarios.

3.6 Matchmaking

The matchmaking system is a crucial component of our multiplayer Sudoku game. Our goal is to create a balanced competitive environment where players compete against opponents of similar skill levels at Sudoku puzzles of their ability, ensuring fair and competitive gameplay. To achieve this, we can utilize Photon's lobby filtering features to match players based on their Elo ratings. The matchmaking protocol will go as follows:

1. Players enter the matchmaking queue
2. The Photon matchmaking system should pair players of similar abilities
3. Once an appropriate opponent is found, the players are placed in the same game lobby and assigned a puzzle difficulty level based on their Elo ratings.

The difficulty levels I have assigned to Elo ratings are as follows:

I have chosen these Elo ranges to cater to players with varying skill levels, ensuring that they face appropriate challenges as they progress as solvers. As players improve

Elo Range	Difficulty
0-1199	Very Easy
1200-1399	Easy
1400-1599	Moderate
1600 and above	Hard

Figure 3.1: Mapping of Sudoku difficulty rating's and Elo boundaries

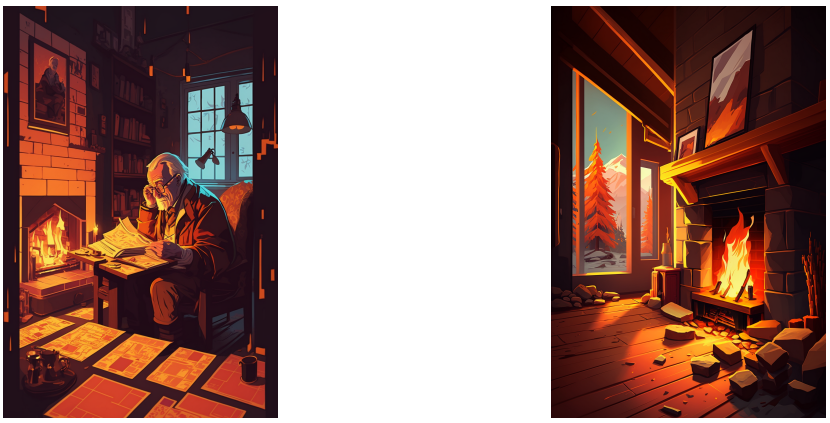
Figure 3.2: SudokuSlam Logo

and increase their Elo ratings, they will encounter more difficult puzzles and opponents of similar skill levels. This design choice fosters a competitive environment that encourages skill development and long-term engagement with the game. In the case that the two users match-up against each other are in different grade boundaries, the average of the two player Elo's should be taken and this will determine the puzzle to be provided. This ensures that the puzzle is neither too easy for the higher-rated player nor too challenging for the lower-rated player.

3.7 Branding

Branding plays a key role in the success of any game and plays a huge role in establishing a game identity. I have decided to brand the game as "SudokuSlam. This combination of words emphasizes that our game offers a fresh twist on the classic puzzle game, positioning it as an engaging and innovative option for Sudoku enthusiasts and casual gamers alike. Moreover, the name "SudokuSlam" is easy to remember, pronounce, and spell, which further increases its appeal and marketability.

The background elements contribute greatly to the atmosphere of an app and I wanted to create an interesting UI while still appealing to the roots of puzzle solving, which is typically a peaceful experience. To do this, I made use of Midjourney [7], a powerful AI image generation tool, which helped me develop the background images shown in Figure 3.3, creating a warm relaxed atmosphere to our app while still remaining eye-catching. The home screen wallpaper was generated with the prompt



(a) Home Scene Wallpaper (b) In-Game Wallpaper

Figure 3.3: Wallpapers created with Midjourney

”wideangled portrait wallpaper with vibrant warm colours depicting a man in his cigar room playing a sudoku in his newspaper by the fireside in the style of firewatch video game art –ar 2:3”. The game scene wallpaper was generated with ”wideangled portrait wallaper with vibrant warm colours looking directly at a fireplace in the style of firewatch video game art –ar 2:3”. The logo shown in 3.2 was designed in Photoshop.

Finalising a brand identity for our game aids in creating a consistent UI which enhances the overall user experience and usability of the app.

3.8 Requirements

This section outlines the key measures of a successful implementation of a multiplayer Sudoku app. By implementing the features and functionalities described in this documentation, the multiplayer Sudoku application will provide an engaging, interactive, and challenging experience for users of all skill levels.

3.8.1 Multiplayer Networking and Elo-Rating System

The app must support seamless two-player networking functionalities, which would be enhanced with the addition of matchmaking functionalities aligned with the design of the Arnad’s Elo system. The rating system must save and update player ratings locally in PlayerPrefs.

3.8.2 Elo Rating System

The application must implement an Elo rating system to track users' skill levels and progress over time. This system should: This system should:

- Assign an initial rating to new users
- Update ratings based on users' performance in multiplayer games
- Allow users to view their own ratings and the ratings of others
- Facilitate matchmaking between users of similar skill levels for balanced gameplay

3.8.3 Puzzle Generation

The application must be able to generate a diverse puzzle set of Sudoku's from a small number of seed cells, providing users with a variety of difficulties and unique challenges. The puzzle set should include puzzles ranging in difficulty from easy to expert.

3.8.4 Sudoku Grid Display

The application must display a clear and user-friendly grid for each puzzle. The grid should be responsive and easy to interact with, allowing users to input numbers and notes. It must also visually differentiate between user-inputted numbers, pre-populated numbers, and notes. Both players in the game should be solving the same Sudoku puzzle.

3.8.5 Note Taking

The app should allow users to take notes by writing small numbers in the corner of grid cells. Users should be able to easily add, edit, and remove notes during gameplay. This feature should:

- Allow users to input and edit small notes or potential number candidates within grid cells
- Differentiate visually between notes and actual number inputs
- Support quick and easy toggling between note-taking and number input modes

3.8.6 Hints

The application must offer a hints feature to assist users during gameplay. This is to help promote a challenging but fun environment. Users should be able to receive hints in a manner that's fair to both users, meaning they should be limited in use .

3.8.7 User Interface

The user interface must consist of an intuitive design which is implemented responsively to support a wide array of devices and is conducive to development of the other functionalities of the game.

- Straightforward navigation between different sections of the application
- Users should be able to create and modify a nickname.
- Clear instructions and tutorials for new users
- Responsive design, compatible with various devices and screen sizes
- Customisable settings and preferences, such as color schemes and font sizes

3.8.8 Additional Requirements

Some other features which do not take priority in the development of this app but would overrall lead to a better solution include:

1. Customisation: The app should offer some customisation options, such as a light and a dark mode.
2. Accessibility: The app should offer accessibility features, such as adjustable font sizes and color schemes, to cater to a broader range of users.
3. Puzzle Sharing: The app could allow users to share generated puzzles with friends or on social media platforms.
4. Tutorials: The app could include tutorials for new players explaining Sudoku or teaching different solving techniques.
5. Leaderboards: The app may display local leaderboards based on the Elo rating system, allowing players to track their progress and compete with others.

Chapter 4

Implementation

In this chapter I will discuss the key components of the multiplayer Sudoku solution I have designed.

4.1 Overview

Unity is fundamentally structured around the concept of scenes and game objects. Scenes serve as individual levels or sections within a game, each containing a collection of game objects that make up the various interactive elements, characters, and environment. To add functionality and interactivity to these game objects, we can attach C# scripts that define behaviors, actions, and events. These scenes are the building blocks of our app and leads us to the home screen as the natural starting point to begin implementing the key functionality of our game.

4.2 Home Scene

The home scene designed in Unity for the SudokuSlam game provides a simple and user-friendly interface. The branding dominates the screen, establishing a strong identity for the game and creating a memorable first impression. Users are greeted by a nickname input form beneath the logo, where they may easily add their desired nickname, allowing for a more personalisation and promoting a sense of community among players. The design also includes a large play button, which not only prompts players to begin playing but also acts as a link to the Launcher script. This script, in turn, is in charge of controlling networking features and utilising Photon's matchmaking facilities to pair players based on their Elo ratings produced by our EloManager.

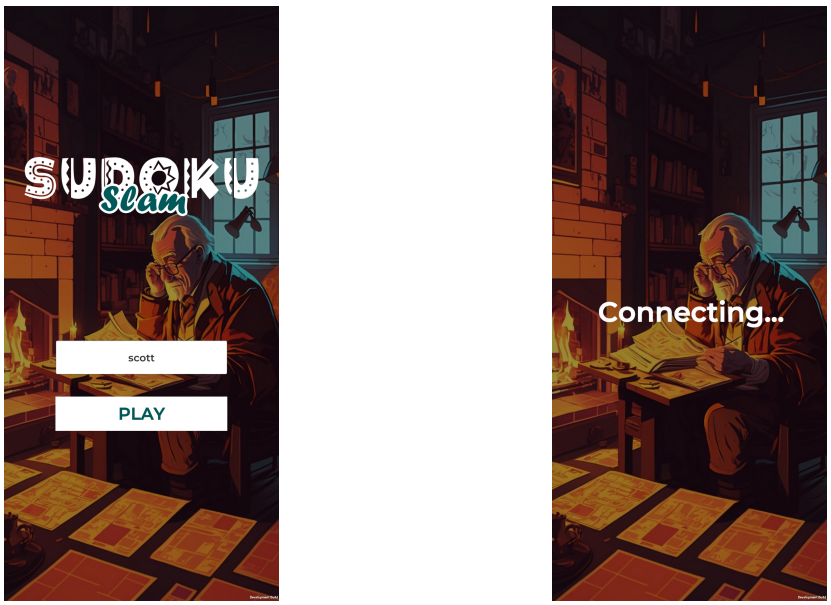


Figure 4.1: Home Scene and Connecting Overlay

This promotes balanced and competitive matches, expanding Sudoku enthusiasts' entire gaming experience. The connecting screen will overlay the rest of the home screen while users are waiting on Photon to find an opponent.

4.2.1 Launcher

The Launcher script serves as a central hub for handling matchmaking and room management. It is responsible for setting a maximum of two players per room, connecting the application to the Photon Cloud Network, setting custom room properties and sets automatic synchronization of scenes between the master client and other clients within the same room.

Upon connecting, the script attempts to join a random room by filtering based on player Elo ratings, ensuring balanced matchmaking. If no suitable room is found, the script creates a new room with the specified maximum player limit. The Launcher script also loads the game scene when the room is full, which requires setting some necessary room properties, such as the Sudoku puzzle, as well as how many empty cells it has and the nicknames of each of the players. Additionally, it allows users to update their nicknames, saving the updated name to PlayerPrefs, which is a simple data storage system in Unity for saving and retrieving user preferences and game progress locally.

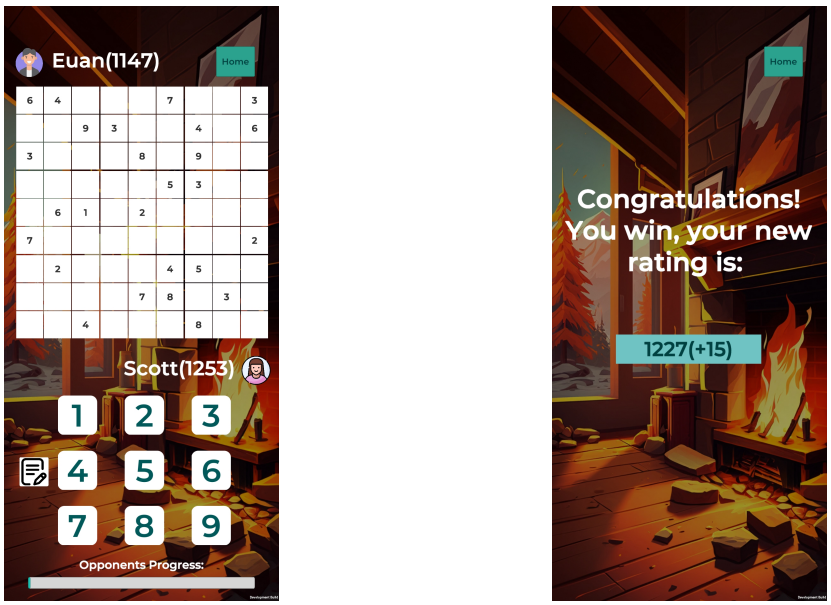


Figure 4.2: Game Scene and Game End Overlay

4.3 Game Scene

The final design of the game scene and game end sequence can be seen in Figure 4.2.

4.3.1 Board Representation and Display

In order to display the Sudoku to the user, I have implemented a SudokuGrid script, which is designed to create and display a 9x9 Sudoku grid on the screen. This script makes use of various private variables which enables easy adjustments of the grid's appearance, which can be accessed in Unity's Inspector Panel with the [SerializeField] attribute. Exposing variables such as the number of rows and columns allows for potential new future applications of the game, with varying grid sizes.

To display the grid on the screen, the SudokuGrid script creates 81 grid_square GameObjects for each cell in the Sudoku grid and positions them appropriately within the center of the screen. It takes into account the square offset, block offset, and scale to calculate the position of each square relative to the starting position. The RectTransform component of each square is then updated with the calculated position, ensuring that the squares are displayed correctly on the screen as a 9x9 grid. Each grid square is equipped with a primary text object as well as a note text object for display of the numbers.

If the user running the script is the primary user, we create a SudokuStorage class which allows it to generate a Sudoku puzzle. The puzzle and solution grids are stored in the rooms custom properties and the primary users grid squares are assigned their respective numbers. If the user running the script is not the primary user, i.e. the second player to join the game, the grid squares are spawned and the grid cell values are fetched from the rooms custom properties. I encountered a scenario of race conditions during the initial implementation stages, as the second users grid script would try and fetch the grid data before they were updated on the server. To resolve this issue, I implemented a co-routine, which attempts to fetch the grid data from the room properties. If the server has not yet updated, the script waits for 0.5s to allow other tasks to complete and attempts to fetch the data again. This approach eliminates the race conditions, leading to a consistent gaming experience for both users.

4.3.2 Sudoku Generation

In order generate a large puzzle through transformations I have designed a SudokuStorage class responsible for performing a random set of transformations upon existing seed grids at the specified difficulty.

The class begins by selecting a random puzzle and solution grid pair from the specified difficulty list (Very easy, Easy, Medium, or Hard) using the Generate method. The selected grids are stored as 9x9 integer arrays, which are convenient, easy-access data structures for Sudoku puzzles. I also considered a list of integers for this purpose, but a 2D array effectively captures the inherent two-dimensional nature of a Sudoku grid, allowing for more intuitive manipulation of the grids. In contrast, a list of integers or characters would necessitate additional calculations for row and column access, resulting in reduced code readability and increased computational complexity. Transformations are then performed on the chosen grid via the PermuteGrids method which starts by determining a random number of permutations to perform (between 10 and 20). It then iterates through each permutation and selects one of the four transformation operations: swapping rows within a block, swapping columns within a block, swapping horizontal blocks, or swapping vertical blocks. The maximum number of distinct puzzles that can be generated is determined by the possible permutations for each operation:

- There are 3! (6) ways to permute rows within a 3x9 block, considering that there

are 3 rows and we swap 2 distinct rows.

- Similarly, there are 3! (6) ways to permute columns within a 9x3 block.
- There are 3! (6) ways to swap two distinct horizontal 3x9 blocks.
- There are 3! (6) ways to swap two distinct vertical 9x3 blocks.
- There are 9! (362,880) ways to permute the symbols (numbers) in the grid.

The total number of permutations for each operation is multiplied to calculate the maximum number of unique puzzles that can be generated from a single seed grid:

$$6 \times 6 \times 6 \times 6 \times 362,880 = 46,945,920 \text{ unique puzzles}$$

This is an upper bound estimate, and the actual number of distinct puzzles may be somewhat lower, as some combinations of permutations could result in the same final grid. However, the implementation ensures that a vast number of distinct and valid Sudoku puzzles can be generated from a single seed grid. When considering 4 different difficulty levels with 10 seed puzzles for each level, there is a potential to generate a significant number of distinct Sudoku puzzles. The total number of unique puzzles that can potentially be generated is:

$$10 \times 4 \times 46,945,920 = 1,877,836,800 \text{ unique puzzles}$$

This represents a large and diverse puzzle set which services the needs of our app.

4.3.3 Number Pad

The choice to include a numpad in the game scene for user input serves to restrict input to only valid numbers while helping create a simple intuitive user interface. The interaction between the numpad and the Sudoku grid is facilitated by event listeners, which allow grid squares to respond to numpad button clicks and update their state accordingly. When a player selects a square and clicks a number button on the numpad, the selected square is updated with the value from the numpad button if the input is correct. Otherwise the grid squares and numpad button turn red for a short time to indicate incorrect input to the user. This process streamlines the input process and improves the overall user experience by making it seamless and efficient.

4.3.4 Notes

The notes feature in SudokuSlam is designed to provide players with a convenient way to add or remove notes on individual grid squares. These notes represent possible numbers that might fill the squares, assisting players in solving the puzzle. The primary components of the hints feature include the 'NotesToggle' button which enables and disables note-taking mode. When enabled and with a grid square selected, the user can add nodes to a desired grid square. Each grid square contains a 'notes text' object which is revealed when a user adds a note, and hidden when the cell's correct value has been entered.

In order to achieve this across all 81 grid squares, the implementation of simultaneous management of multiple grid squares was necessary. This was achieved by attaching an instance of the same grid square script to each of the 81 squares, allowing them to maintain their own state and handle the notes feature individually. Event-driven programming techniques were employed to ensure that each grid square responded to relevant user interactions, such as clicks on the notes button or selecting a grid square. The use of event listeners and event handlers in the grid square script facilitated the communication between the UI elements and the script logic, while maintaining the independence of each square's state.

4.3.5 Profile Pictures and Nicknames

In order to enhance the user experience and increase customization options, the game currently includes four random profile pictures for players to choose from. These profile pictures are automatically assigned at random when the user starts the game. The initial implementation of this encountered a problem related to differences in file path handling between platforms, making it difficult to access the images consistently. By employing Unity's Resources.LoadAll() method, we have resolved this issue, ensuring that profile pictures are seamlessly loaded and displayed across various devices and platforms. As a future improvement, the intention is to allow users to upload their custom profile pictures, further personalizing their in-game identity and adding a unique touch to their gaming experience.

Next to the profile pictures, we display the players' nicknames alongside their Elo ratings in a nickname(elo) format, similar to Chess.com, to further the apps' sense of

familiarity and competition between the users. The NicknameDisplay script is responsible for updating and displaying these user properties. The script utilizes Photon's event system, specifically the OnPlayerEnteredRoom and OnPlayerLeftRoom callbacks, to update the nickname display whenever a player joins or leaves the room. The UpdateNicknameDisplay function is called in these callbacks, as well as in the Start method to ensure the display is updated when the game starts. The UpdateNicknameDisplay function iterates through the PhotonNetwork.PlayerList, retrieves each player's nickname and Elo rating, and formats the text accordingly. The nickname text objects are attached using serialized fields, which allow us to easily assign references corresponding to objects from the scene.

4.3.6 Progress Bar

In the SudokuSlam game scene, I have implemented a progress bar to visually represent each player's progress in solving the puzzle. This progress bar feature enhances the user experience by providing an intuitive way to track the competition between players.

To achieve this, I have attached a custom script to a Slider object in the Unity scene. The Slider object serves as the visual representation of the progress bar, which fills up as the players fill in the empty cells of the Sudoku grid. The script is responsible for updating the Slider's value in real-time, reflecting each player's progress throughout the game. Our implementation listens for changes in the user scores, which are stored as custom room properties in the Photon networking framework. The script utilizes Photon's event system, specifically the OnRoomPropertiesUpdate callback, to detect updates in the room properties, such as changes in the players' scores. This combats potential race conditions experience when declaring the initial values of the bar at the start of the game. This method checks if the player is connected to the PhotonNetwork, then iterates through the custom room properties to find the opponent's score. After finding the opponent's score, it sets the progress bar's max value to the number of empty cells in the game and the current value to the opponent's score, thus displaying the opponent's progress relative to the total game progress. The method is called at the start of the game and whenever room properties are updated, ensuring that the progress bar remains up-to-date with the latest score information.

When a player's score changes, the script retrieves the updated score from the room properties and sets the Slider's value accordingly. This dynamic progress bar allows players to easily gauge their progress in real-time and adds an engaging element to the Sudoku gameplay experience.

4.3.7 Home Button

The game scene UI features a home button which takes the user back to the home scene. The LeaveLobby script listens for the onClick event on this button and leaves the Photon lobby and loads the game scene upon press.

4.3.8 End of Game Listener

The ”EndGameListener” script is attached to an empty game object in the game scene and monitors the custom room properties in a Photon-based multiplayer game to detect when the game is over. When the game over property is set, it triggers the end of the game sequence, which includes calculating and updating the new Elo ratings for the players involved in the match.

To do this, the script first registers for Photon network events when enabled, and unregisters when disabled. It listens for any changes in room properties, specifically for the ”game_over” property. If this property is detected, the script initiates the end of the game sequence.

The end of game sequence activates the gameEndSequence GameObject, which indicates the game is over. Additionally, the script calls the EloManager's UpdateEloAfterMatch() function to calculate and update the players' Elo ratings based on the match outcome. This new elo rating is displayed on the gameEndSequence, as well as the difference between the new and old ratings.

4.4 Elo Manager

The EloManager class is responsible for managing player Elo ratings. The manager ensures Elo's are saved locally on the user's device using Unity's PlayerPrefs system. This allows the ratings to persist between game instances and track users skill levels and progress over time. The ratings are fetched from the users local storage using

their playerId unique identifier as a key. The manager also implements the formulas shown in the design section within the UpdateEloAfterMatch() method in order to update player Elos between match-ups. The manager sets a default elo of 1000, meaning players will start their elo progress here and adjust to their true rating over time. References are made to this script both in the home and game scene in order to handle elo functionalities across the app.

Chapter 5

Evaluation

I performed a variety of tests in order to evaluate the performance of the SudokuSlam implementation and I will discuss the results within this chapter.

The performance and user experience of the app was assessed using both a Samsung Galaxy S22 and from within the Unity simulator. By implementing the game on the cutting-edge Galaxy S22, the research capitalized on the device's advanced hardware and software capabilities to ensure seamless gameplay and interaction. Additionally, the Unity simulator provided a controlled and customisable environment, allowing for the thorough examination of diverse gameplay scenarios and user behaviors. Ultimately, the combination of real-world testing on the Galaxy S22 and the flexibility of the Unity simulator laid a solid foundation ensuring the reliability and relevance of the results presented in this evaluation.

5.1 Sudoku Generation

A key factor underpinning the success of our application was the fast generation of a diverse set of valid puzzles at a range of difficulty levels. The use of puzzle transformations allowed us to overcome the early issues experienced with implementing alternative methods. Traditional generation algorithms often rely on a trial-and-error approach, which can lead to increased computation time and resource usage. In contrast, the transformation-based method successfully employed in SudokuSlam leverages a pre-existing set of base grids and applies a series of deterministic transformations to generate new puzzles. After conducting a large number trials, it was observed that the algorithm performed efficiently, with an average generation time of approximately

15 milliseconds. This is very much comparable with some of the other methods we have looked at in this report. This approach ensures a more predictable and efficient generation process, reducing the overall computational overhead and offering a more efficient, flexible, and scalable solution.

5.1.1 Grid Validity

Each generated Sudoku grid was assessed by hand to ensure that it adhered to the standard rules of Sudoku, including the requirement that each row, column, and sub-grid contains the numbers 1-9 without repetition. Each grid was entered into a third-party solver for verification, which assessed the validity of the Sudoku as well as the number of possible solutions. [19] This analysis confirmed the integrity of the generated puzzles, indicating that the transformation process consistently produced valid Sudoku grids.

5.1.2 Difficulty

A random subset of generated Sudoku puzzles were evaluated across a range of online grading algorithms. The difficulty ratings were largely in alignment with these new gradings. This analysis confirmed that the transformation process can offer a balanced distribution of difficulty levels, catering to a wide spectrum of user abilities and preferences.

5.1.3 Diversity

I have discussed throughout this report the importance of a diverse puzzle set and described this as an important key feature of this project. A diverse puzzle set ensures that users are continually challenged with fresh and unique Sudoku grids, reducing the likelihood of repetitive gameplay and fostering a sense of excitement and discovery as players encounter new challenges. The project's success in implementing the transformation algorithms is evident in the vast upper bound potential puzzle set of nearly 2 billion puzzles. This enormous figure highlights the effectiveness of the transformation process implemented in the SudokuSlam game at generating a multitude of distinct Sudoku grids, offering users a virtually inexhaustible source of engaging gameplay.

5.2 Matchmaking

Simulated matches were conducted between both virtual and real players with different ELO ratings to examine how the rating system responded to various outcomes. The matchmaking algorithm was also assessed by analyzing the pairing of players with similar ELO ratings, ensuring that the system provided fair and balanced matches.

5.3 User Interface

5.3.1 Response Time

It was important that the user interface was responsive and was capable of presenting users with a Sudoku puzzle fast. In order to evaluate this, the time taken for two users to connect to a lobby and have the game grid loaded was measured and further tested through usability testing. This metric proved crucial in understanding the effectiveness of the matchmaking and synchronization processes in providing a seamless user experience. Ultimately this metric has the final say in the success of the networking requirements of the app.

The average time for two users to connect to a lobby and have the grid loaded for both users was found to be 3.9 seconds. The grid loads significantly faster for the primary user as they are responsible for generating the puzzle, whereas the secondary player has to wait on the grid being uploaded to the rooms custom properties and fetch it. This swift connection and game grid loading time highlights the application's effective design and its ability to create a smooth and uninterrupted gaming experience. In this regard, the performance of the multiplayer Sudoku application indicates the successful implementation of efficient networking and resource management techniques, ensuring that users can quickly engage in competitive Sudoku gameplay without unnecessary delays.

5.3.2 Note-taking

To evaluate the effectiveness of the implemented hints feature, a combination of manual testing and debugging techniques were employed. This involved interacting with the grid squares in the Unity editor, simulating various user scenarios, and examining the behavior of the feature. Debugging tools such as Debug.Log were used to monitor

the internal state of each grid square and verify the correct functioning of the notes feature. Through this iterative testing and refinement process, any issues or edge cases were identified and addressed, ensuring that the hints feature worked seamlessly and consistently for all 81 grid squares.

5.3.3 Cognitive Walkthrough

By simulating how a first-time user would navigate and interact with the app, the cognitive walkthrough helped identify potential bottlenecks and areas of confusion in the user interface. This test focused on assessing the intuitiveness of the app's design and layout, ensuring that users can easily understand and engage with the game. Don Norman, expert cognitive scientist, claims in his 1989 book that any error made by a user when interacting with the product is the fault of the design, and not the user. This idea was used to create a simple but functional UI which complemented the functionalities of the app.

5.4 Comparison to Existing Solutions

In this section, we will compare our multiplayer Sudoku app to existing solutions in the market. While there are several online Sudoku games available, our app stands out due to its unique implementation of an Elo rating system. This attempts to mimic the recent success of Chess.com and promote a more competitive community around Sudoku. In contrast to other Sudoku apps, which often lack a comprehensive rating system or use simpler metrics such as win-loss ratios or average completion times [18], our Elo-based approach offers a more sophisticated and accurate representation of player skill.

SudokuSlam app also includes a game mode that is similar to popular implementations found in other Sudoku apps. This mode offers a familiar and enjoyable experience for players who have previously engaged with other Sudoku platforms. Incorporating this tried-and-tested mode, our app appeals to a broader audience, making it a more versatile and accessible solution.

Our Sudoku generating methodology, as previously discussed, is capable of generating up to nearly two billion unique Sudoku puzzles. This is comparative with other

online Sudoku games and highlights how our project has been successful in creating an an engaging, scalable game with a diverse puzzle set. [26]

As previously discussed, note-taking is a key feature of online Suduoku games and is featured in many of SudokuSlam's competitors. Successfully implementing this feature within our implementation allows SudokuSlam to compete with existing implementations in the space while also developing its own unique features.

Chapter 6

Conclusion

6.1 Limitations and Future Improvements

This section outlines the limitations of the multiplayer Sudoku application, identifying areas where the app falls short of its potential and acknowledging constraints that may impact its performance or user experience.

6.1.1 Local Storage of Elo Ratings

The application does not utilize a database solution for storing user Elo ratings; instead, it relies on local storage. This approach introduces several drawbacks:

- **Lack of Leaderboards:** Due to the absence of a centralized database, implementing leaderboards based on Elo ratings is not feasible. Leaderboards are an effective way to encourage competition and foster user engagement, and their absence may result in a less compelling experience for players.
- **Limited Social Applications:** Local storage of Elo ratings restricts the potential for social features such as comparing ratings with friends or sharing progress on social media platforms. This limitation may reduce the app's appeal to users who enjoy comparing their progress and skill levels with others.
- **Data Persistence:** Storing Elo ratings locally raises concerns about data persistence, as users may lose their ratings and progress if they switch devices or uninstall the app. This can lead to user frustration and discourage long-term engagement with the application.

6.1.2 Limited Game Types

The final iteration of the Sudoku Slam solution includes just one game type for the purposes of this project. Varied game types underpins the success of a variety of competitors in the space and expanding the game-types offered in the future would make for a more well-rounded solution.

6.1.3 Hints

One of the features we sought to implement was the inclusion of hints to aid players in solving challenges and to enhance their understanding of the game mechanics. However, due to various constraints elsewhere we were unable to successfully implement the hint system in the final version of the game. This feature could significantly improve the player experience in future iterations, and should be a high priority should this project be continue further, as it is a common staple in the Sudoku community.

6.1.4 Accessibility

Unfortunately the game does not implement any accessibility features, such as text to speech or colour blind settings. Should the app become ready for a production build in the future, this should be an important fixture. Tutorials may also be a beneficial future improvement in order to encourage new Sudoku solvers.

6.2 Project Accomplishments

The project achieved many of its design goals and can be summarised like so:

- Thorough background research into Sudoku, puzzle generation, Chess and existing Multiplayer applications in the field
- Design and implementing of a lightweight Sudoku puzzle generation methodology which allows for a vast and diverse puzzle set which meets the standards of well-formed Sudoku's. The set also caters to a range of difficulty levels.
- Implementation of the Elo rating system and successful integration with Unity matchmaking features, resulting in a balanced, fair, competitive environment.

- Creation of a responsive, aesthetic user interface which complements the features of our Sudoku game and can successfully allow users to compete over the same Sudoku grid. The UI also displays some more personal features, such as player nicknames, elos and puzzle progress.
- Offer customisation options in the form of nicknames
- Design of a scalable Multiplayer solution which is built upon solid foundations supporting future release and development

6.3 Closing Remarks

In conclusion, this report detailed the development of SudokuSlam, an Android app that introduces a multiplayer experience to the widely popular game of Sudoku. Through careful consideration of design elements, user experience, and technical implementation, the project successfully met several of its objectives and criteria.

Sudoku, despite its immense popularity in a single-player format, has seen a relatively limited exploration multiplayer contexts. By developing SudokuSlam, this project has demonstrated its potential to bring together Sudoku enthusiasts in a shared, interactive environment, allowing them to engage in both competitive and collaborative modes. The app's design, built on well-established principles, is left open for many areas of future development.

Bibliography

[1] Can sudoku have multiple solutions? https://masteringsudoku.com/can-sudoku-have-multiple-solutions/.

[2] Custom room properties pun documentation. https://doc-api.photonengine.com/en/pun/v2/class$_{p}hoton_{11}r ealtime_{11}r oom.html$.

[3] Elo rating system - chess terms. https://www.chess.com/terms/elo-rating-chess.

[4] Hidden singles. https://wordsup.co.uk/sudoku/hidden-single.

[5] How do we create sudoku? https://www.sudokuoftheday.com/creation.

[6] Live sudoku. https://www.livesudoku.com/.

[7] Midjourney. https://www.midjourney.com/.

[8] Naked singles. https://wordsup.co.uk/sudoku/naked-single.

[9] Nikoli sudoku. https://www.nikoli.co.jp/en/puzzles/sudoku/.

[10] Notes in sudoku. https://sudoku.com/sudoku-rules/notes-in-sudoku/.

[11] Ny times sudoku. https://www.nytimes.com/puzzles/sudoku/.

[12] Photon 2. https://assetstore.unity.com/packages/tools/network/pun-2-free-119922.

[13] Puzzle difficulty ratings. https://www.sudokuoftheday.com/difficulty.

[14] Single's chains. https://www.sudokuwiki.org/Singles$_{C}hains$.

[15] Sudoku. https://www.sudoku.name/.

[16] Sudoku exchange. https://sudokuexchange.com/.

[17] Sudoku online. https://www.sudokuonline.io/.

[18] Sudoku race. https://sudokurace.io/.

[19] Sudoku solver by andrew stuart. https://www.sudokuwiki.org/sudoku.htm.

[20] Sudoku symmetries. https://www.mathpages.com/home/kmath661/kmath661.htm.

[21] Sudoku variants worth checking out. https://masteringsudoku.com/sudoku-variants/.

[22] Sudoku zenkai. https://sudokuzenkai.imaginationoverflow.com/.

[23] Swordfish strategy, sudokuwiki. https://www.sudokuwiki.org/Sword$_{F}ish_{S}trategy$.

[24] Unity. https://unity.com/.

[25] Washington times daily sudoku. https://www.washingtontimes.com/puzzles/sudoku/.

[26] Web sudoku. https://www.websudoku.com/.

[27] What is c programming? a beginner's guide. https://www.pluralsight.com/blog/software-development/everything-you-need-to-know-about-c-.

[28] What makes sudoku easy, medium, or hard? here's the science behind sudoku levels. https://www.sudokulovers.com/what-makes-Sudoku-easy-medium-or-hard.

[29] X-wing strategy, sudoku wiki. https://www.sudokuwiki.org/X$_{W}ing_{S}trategy$.

[30] You may be good at sudoku, but do you know what the word literally means? https://www.dictionary.com/e/sudoku-game-meaning/.

[31] The value of getting personalization right-or wrong-is multiplying, Nov 2021.

[32] Michael Becher. Sudoku generator and solver. http://sudoku.becher-sundstroem.de/.

[33] Christian Boyer. Supplément de l'article "les ancêtres français du sudoku". https://web.archive.org/web/20061210103525/http://cboyer.club.fr/multimagie/SupplAncetresS 2006.

[34] Sankhadeep Chatterjee, Saubhik Paladhi, and Raktim Chakraborty. A comparative study on the performance characteristicsof sudoku solving algorithms. *IOSR Journals (IOSR Journal of Computer Engineering)*, 1(16):69–77.

[35] Chess.com. Chess is booming! and our servers are struggling. https://www.chess.com/blog/CHESScom/chess-is-booming-and-our-servers-are-struggling.

[36] Jean-Paul Delahaye. The science behind sudoku. *Scientific American*, 294(6):80–87, 2006.

[37] Johan Ekström and Kristofer Pitkäjärvi. The backtracking algorithm and different representations for solving sudoku puzzles, 2014.

[38] Arpad E Elo. The proposed uscf rating system. its development, theory, and applications. *Chess Life*, 22(8):242–247, 1967.

[39] FIDE International Chess Federation. Chess ratings. https://ratings.fide.com/calculator$_r td.phtml$.

[40] Bertram Felgenhauer and Frazer Jarvis. Enumerating possible sudoku grids. *Preprint available at http://www. afjarvis. staff. shef. ac. uk/sudoku/sudoku. pdf*, 2005.

[41] Puzzle Genuis. Sudoku from scratch 2. https://puzzlegenius.org/sudoku-from-scratch-part-two.

[42] Mattias Harrysson and Hjalmar Laestander. Solving sudoku efficiently with dancing links, 2014.

[43] Independent.ie. There's no escape... from sudoku. https://www.independent.ie/opinion/analysis/theres-no-escape-from-sudoku-25963746.html, 2005.

[44] Jon Kleinberg and Eva Tardos. Algorithm design addison wesley. *Boston, MA*, 2005.

[45] Donald E. Knuth. Dancing links. *arXiv preprint cs/0011047*, 2000.

[46] Sasha Langholz. The benefits and challenges of ios vs android app development, Nov 2022.

[47] Learn-Sudoku.com. A note on pencil marks. https://www.learn-sudoku.com/pencil-marks.html.

[48] NY Louis Lee, Geoffrey P Goodwin, and Philip N Johnson-Laird. The psychological puzzle of sudoku. *Thinking & Reasoning*, 14(4):342–364, 2008.

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

[50] Paul Lodder. The use of the k-factor in estimating individual ability.

[51] Harold Nolte. Sudoku glossary of terms.

[52] Building Up Puzzles. Sudoku: Bagging a difficulty metric & building up puzzles. 2008.

[53] Gordon Royle. Combinatorial concepts with sudoku i: Symmetry, 2006.

[54] Kristen(Zidan) Huang Ruby(Ziheng) Ru.

[55] Ed Russell and Frazer Jarvis. Mathematics of sudoku ii. *Mathematical Spectrum*, 39(2):54–58, 2006.

[56] Ivan Semeniuk. Stuck on you: just why is that little box of squares and numbers so horribly addictive? *New Scientist*, 188(2531-2532):45–48, 2005.

[57] Andrew C Stuart. Sudoku creation and grading. *Mathematica*, 39(6):126–142, 2007.

[58] Sudoku.com. The history of sudoku. https://sudoku.com/how-to-play/the-history-of-sudoku/.

[59] SudokuPrimer. Sudoku glossary of terms. https://sudokuprimer.com/glossary.php.

[60] Petroc Taylor. Global mobile os market share 2022, Feb 2023.

[61] Jozef Zurada et al. Optimization problems and genetic algorithms. *Review of Business Information Systems (RBIS)*, 14(3), 2010.