Bringing Sudoku to the Gaming Industry

Jason Chin

June 17, 2008

Abstract

The rise in popularity of Sudoku puzzles over the past few years has seen the puzzle software and publishing markets saturated with Sudoku games and puzzles. Many publishers have tried to make a ‘quick profit’ by producing software that does not offer an experience greater than what a pen and newspaper can offer. Software produced are often clones of the same game, offering a very limited range of features and are generally aimed at a very ‘casual’ audience.

The aim of this project is to offer an experience that extends beyond the limitations of a typical Sudoku puzzle, to allow players to stretch their puzzle solving abilities, for them to compete against an AI or other Sudoku players and to offer a rich feature set that would give a strong reason for a Sudoku player not to go back to the pen and newspaper Sudoku experience.

Acknowledgements

I would thank Margaret Cunningham who supervised this project and provided invaluable guidance and support throughout. I would also like to thank my second marker, Dr. Alessio Lomuscio, for his help and support.

I would like to say thanks to the people who participated in the testing of the project, the members from Games Society from Imperial College, the students from BETHS Grammar School and their parents. They provided invaluable help and feedback for evaluating the project.

I would like to thank Microsoft for creating such a well built framework, and everyone who provided me with image and audio resources to use for the project. I would also like to thank CSG from the Department of Computing at Imperial College for providing me their Xbox 360 to work on.

Lastly I would like to thank my sister, for consistently peskering me to add her somewhere into the report.

Contents

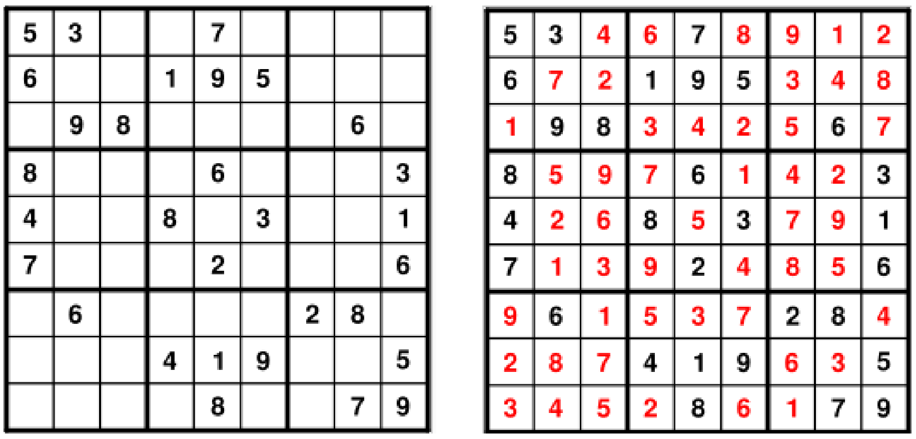
Chapter 1

Introduction

Sudoku is a logic puzzle based on symbol placement and arrangement involving various constraints. The objective of a typical Sudoku puzzle is to fill a partially completed 9×9 grid in such a way that each column, row and 3×3 'box' must contain the digits 1-9. Like many logic puzzles, the challenge in Sudoku is finding the single unique solution for the puzzle using logic based techniques and without guessing from a given partially filled grid.

The puzzle originated in the late seventies[1] and has since garnered international popularity, often printed daily in various newspapers and magazines.

The completion of a puzzle requires good logically deduced reasoning skills and the ability to quickly scan and analyse the problems involved. Solving a Sudoku grid is also known to be an NP-complete problem[8]. A more detailed explanation on the rules is given on section 2.1



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

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

Figure 1.1: A Sudoku puzzle and its solution

1.1 Motivation

In recent years, the popularity of Sudoku has rapidly grown and it now contributes to a significant amount of the puzzle industry. The main reasons for this are due to its easy approachable nature but challenging aspect. Since this growth, there has been a high increase in demand for computer software that will create and solve puzzles, most notably for the media and the gaming industries.

Initial puzzles of Sudoku were often hand-crafted and always drawn up in a symmetrical pattern[2], many also included witticisms. Once its popularity increased, computer generated puzzles were often used, although they are often devoid of symmetry or wit.

Grading a Sudoku puzzle can be a subjective aspect and so attempts at describing the difficulty of a puzzle using computer based reasoning has sometimes lead to a grading of a puzzle that others disagree with. Although many grading systems work very well when generating puzzles, there are many that can often be misleading.

There are also many video games produced that are based on Sudoku, which have been released on multiple platforms and formats. However, many that are produced only republish ready made Sudoku puzzles, rather then generating them on the fly. Although there are software available that randomly generates Sudoku puzzles, very few of them present any sort of multi-player aspect to the game that allow two or more players to compete or play together. Many of the software available do not offer much more than what most newspapers now provide, giving the user a very limited feature set. A small analysis of these software can be seen in section 2.3.

The main motivation for this project is based on producing fun and entertaining Sudoku puzzles and to explore the different aspects in creating Sudoku software. This project also aims to introduce a multi-player aspect into the game so that players can enjoy a puzzle together. The requirements of which are explained in section 3.

1.2 Report Objectives

As the backbone of the project, the ways and variations in which a Sudoku puzzle can be generated, solved and graded have been researched in section 4. Explanations on how each variation works is followed by a summary and reasoning behind which one is favourable above the rest.

Important to all game designing is the design document, which outlines each mode, how they will play and each important factor that is necessary in the game. As an extension to the listed requirements, it explains how the project will set out to fulfil those requirements. This can be seen in section 5.

The implementation of the Sudoku aspect of the project is detailed in section 6, including how my game solves, generates and grades a puzzle. This is followed by the implementation of the rest of the project, outlined in section 7, detailing the structure and how each aspect works.

Chapter 2

Background

This chapter aims to provide an understanding of Sudoku puzzles. Firstly, the rules of Sudoku puzzles are explained, followed by a discussion of several Sudoku variants. This chapter then explains about the various existing projects and software that are similar in nature.

2.1 Explanation of the Sudoku rules

A Sudoku puzzle starts out with a simple 9×9 grid, which is also subdivided into nine smaller 3×3 grids. The aim of each puzzle is to place a number from 1 to 9 on to each cell in the grid, in such a way that each number only appears exactly once in each column, row and sub-grid. This creates a type of Latin Square but one with more constraints.

At the start of each puzzle, a number of 'givens' are provided that make up a partially filled grid. It is then up to the solver to place the various numbers onto the grid in such a way that the constraints are not broken. A puzzle is then solved once each cell contains a number. Each puzzle is solvable through logic (and hence contains only one solution).[2]

1	2	3
2	3	1
3	1	2

Table 2.1: An example of a Latin Square

2.2 Variants

It can firstly be noted that a Sudoku puzzle is not solely limited to the usage of just numbers, but any set of symbols, pictures or characters may be used. Puzzles that do not use numbers are also common and offer a small added degree of interest without much effort. Although they do not necessarily add anything substantial to a given Sudoku puzzle.

We can also note that a grid may not be solely defined as the typical 9×9 grid and could in principle be any size that is $n^2 \times n^2$. It is clear that a larger grid will take a longer time then a smaller grid, as well as having a larger span of difficulty. Again, the actual game mechanics does not change but rather just the necessary time it takes to search and fill a grid.

There are many different variations of Sudoku available. Below is a list of the few variants that I have chosen to take interest in:

2.2.1 Nonomino Sudoku

Nonomino Sudoku (also known as Jigsaw Sudoku[3]) uses varied nonomino shapes instead of sub-grids. Similar rules still apply on this variant. However, as sections are now defined by non-determined shapes, it requires a solver to generally spend a longer amount of time scanning a puzzle. This is an interesting variant as it offers enough difference to the regular Sudoku so that it gives more range but it also is similar enough to still feel like Sudoku.

2.2.2 Hyper Sudoku

Hyper Sudoku[4] variant uses an additional four internal 9x9 grid constraints in the puzzle so that not only the original nine grids have to contain the nine numbers exactly once but so do the extra four grids. As this is just an extra constraint on top of the original ones, it makes puzzles easier to solve as there is more leverage from the overlapping squares. This provides more information to logically reduce the number of possibilities of what can go in a certain cell. However, this also means that less givens could be provided than usual to make a puzzle harder, as it could now be solved using extra logical reasoning because of the extra grids. Hyper Sudoku generally has more of an emphasis on scanning the overlapped squares rather than columns and rows.

2.2.3 Diagonal Sudoku

The Diagonal Sudoku[5] variant adds more constraints to the original version so that solvers now have to make sure that the two main diagonals that run across the grid must also contain the digits 1 to 9. Again, this only adds to the extra constraints and so it is easier to solve a Diagonal Sudoku grid than a normal Sudoku grid as more logical deductions can be made. It would appear that this variant would not be difficult to implement along side the regular Sudoku as it would only require limited changes. However, it does not really offer much more from the original Sudoku as it is quite similar in most respects.

2.2.4 Nonogram

While this is not strictly a Sudoku variant, Nonogram[6] shares many of its qualities. It is also a logic based puzzle and the aim of it is to use logical deduction to create an image from the clues provided on the sides of the grid. Each grid would be arbitrary size with cells that can be 'coloured' or left blank. A set of numbers corresponding to each row and column provide the solver clues on what type certain cells need to be. The numbers measure how many unbroken lines of filled-in square there are on that given row or column. For example, "7 1 4" next to a row would mean that there are sets of seven, one and four filled in squares, in that order, with at least one blank square between successive groups. Like Sudoku, solving Nonograms is also an NP-Complete problem[7]

This is an interesting puzzle type and is different enough from Sudoku to give a different experience and game type when solving a puzzle. However, it shares enough qualities with Sudoku to feel similar enough to it so that solvers would not feel too unfamiliar to the game type as similar logical deductions are made to solve it.

2.2.5 Summary

These variants are interesting and varied enough to give a different experience for someone interested in Sudoku. While they are not necessarily important for the software that is being created in this project, my design should be left open enough so that these puzzles could potentially be implemented later in the project with as little effort or trouble as possible. Possible extensions to this project could be including a number of these variants.

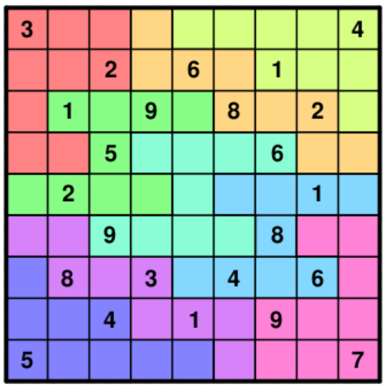
2.3 Existing Work

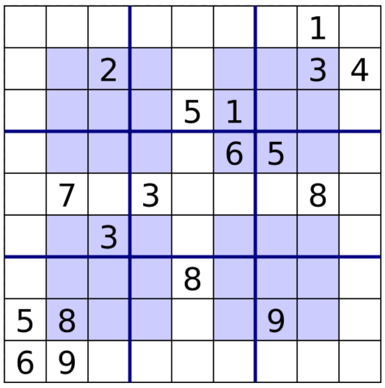
There are many Sudoku based software released on many different platforms, and they all offer various different features. Here is an overview on products of interest that I can take into account when designing my final product.

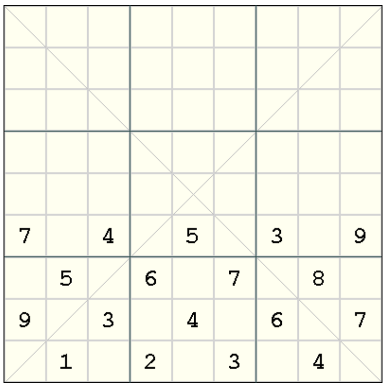
2.3.1 Simple Sudoku

Created by Angus Johnson, Simple Sudoku[11] was created for the Windows platform. As the name suggests, it is a very simple program that generates puzzles for a user to solve. It provides five levels of difficulties and also allows the user to input their own from scratch. All puzzles created use patterns and have rotational symmetry. It also provides hints step by step and also gives the name of the technique used for that move. It also allows Sudoku puzzles to be printed out but offers no multi-player aspect or any Sudoku variants. While notations cannot be made on the grid, it has colouring available to change the colours of specified cells.

While it does not offer an immediate solution to a puzzle, its generation was immediate and


(a) A Nonomino Sudoku puzzle


(b) A Hyper Sudoku puzzle


(c) A Diagonal Sudoku puzzle

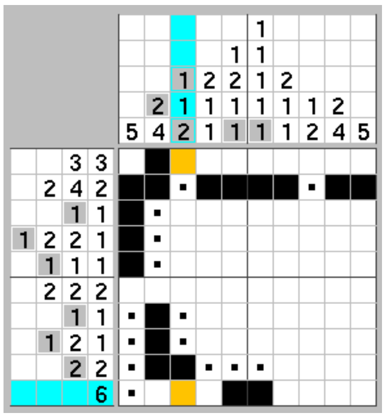

(d) A Nonogram puzzle

Figure 2.1: Variants of Sudoku

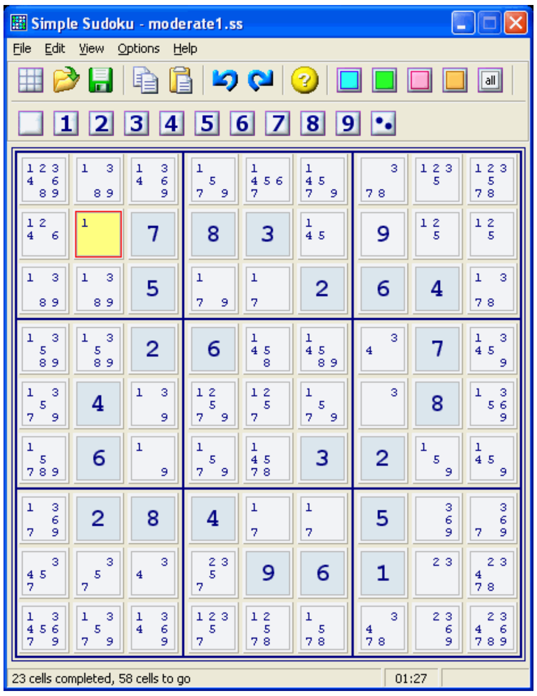

Simple Sudoku - moderate1.ss
File Edit View Options Help
23 cells completed, 58 cells to go
01:27

Figure 2.2: Simple Sudoku interface by Angus Johnson[11]

the puzzles seemed about the correct difficulty when tested.

Although the program is very simple in its implementation and limited in its options, it appears to be quite complex visually, with many unnecessary buttons and labels placed across the interface. However, the input is very effective, both giving the option to use the keyboard or keyboard and mouse for placement of numbers.

Summary: This piece of software has some good features such as a fast random puzzle generator with pattern based implementation, the ability to print out puzzles and the ability to provide hints with reasons as to why. The interface for inputting in solutions is also effective. However, it does not offer a notation solution, or the ability to immediately and automatically solve a puzzle.

2.3.2 Zen of Sudoku

Zen of Sudoku[12] was developed by Charlie Cleveland for Unknown Worlds Entertainment for the PC market. The main emphasis on this piece of software is to give the player a relaxing and calm experience, while playing Sudoku puzzles. It offers a quick tutorial and robust hinting system, offering players full explanations and reasoning. There is only one mode where players can play Sudoku with no time limit and this comes in four difficulty levels. Puzzles are generated completely at random, seemingly on the fly. Players are also able to mark up on a grid. There is also a saving feature that stores the last puzzle and it also has the ability to print puzzles using the default printer options. There is also a trophy system based on achievements made through playing. It offers no multi-player experience or variants.

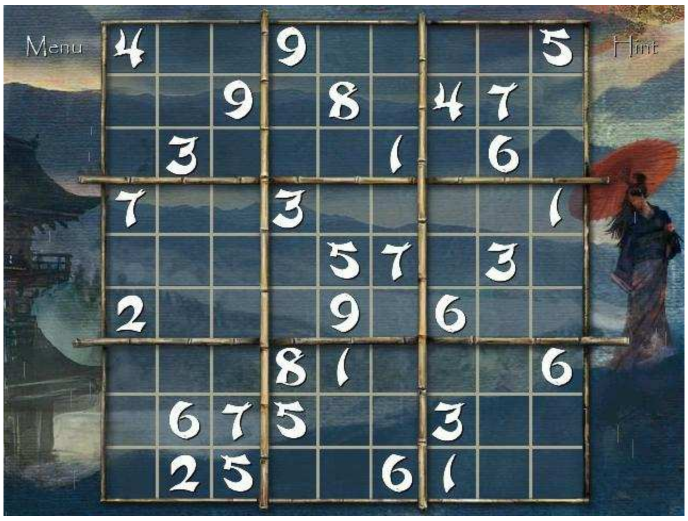


Figure 2.3: Zen of Sudoku interface by Charlie Cleveland[12]

What Zen of Sudoku offers is a calm and serene take on Sudoku. There is a strong emphasis on design, artwork and sound to help add to the experience. I found that this complements Sudoku highly as the nature of the very game is relaxing.

The hinting system is also very strong and not only provides players with information on where they can go next, but also gives an explanation as to why. The ability to print puzzles out is also a good feature. The puzzles that were tested in the game seemed to be about the correct level of difficulty, except for the hardest level, which did not seem that hard.

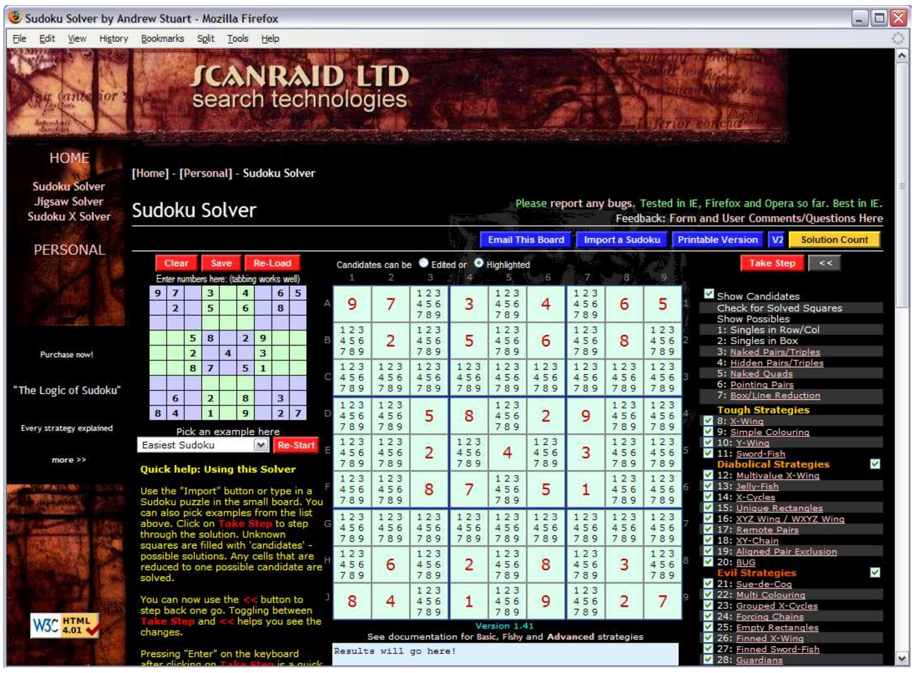
However, the biggest weakness to this game is that the input interface is not intuitive and it is very limited. Input is provided via the mouse only. Each cell is divided up into small blocks of nine and depending on which block is clicked on, determines what number is entered into the cell. This feels very clumsy in practice.

Summary: Zen of Sudoku shows that a good experience is also determined by strong sound and graphical designs and these should be taking in to consideration when designing the project. However it is very limited in features as it does not provide things such as multi-player, immediate solving or any variants.

2.3.3 Scanraid Sudoku

Scanraid Sudoku[13] is an Internet application, developed by Andrew Stuart for Scanraid LTD. This is a web based implementation of Sudoku developed using JavaScript. It does not perform random generation of puzzles but instead provides a number of ready made grids. It offers a comprehensive solver with many various options and techniques that can be selected. It also offers the ability to save a puzzle, e-mail a board, import a grid and provide a printable version. It also has other variants of Sudoku that the user can select.

This web implementation has the obvious advantages that it is cross platform and can be



Quick help: Using this Solver

Use the "Import" button or type in a Sudoku puzzle in the small board. You can also pick examples from the list above. Click on Take Step to step through the solution. Unknown squares are filled with 'candidates' - possible solutions. Any cells that are reduced to one possible candidate are solved.

You can now use the << button to step back one go. Toggling between Take Step and << helps you see the changes.

Pressing "Enter" on the keyboard

Figure 2.4: Scanraid Sudoku interface by Andrew Stuart[13]

easily accessed. The large amount of solving options that have been implemented is also strong and robust explanations for each technique is given.

However, the main problem is that it provides a very awkward interface. Numbers are entered in via the small grid with the large grid used to show the solving techniques used. Marking can be performed but it is tedious to do through the interface. It also does not take full advantage of the platform it uses, as there is no user generated content, such as ratings or user created puzzles, and it does not store any data on users that might be of interest, such as puzzles solved.

Summary: There is a large potential for web based Sudoku applications that could be given through aspects such as user generated content. However, the interface provides a large problem as input is limited. It is also not clear on how multi-player aspects could be integrated.

2.3.4 Sudoku Too

Sudoku Too[14] was developed by Carbonated Games in Macromedia Flash 9 for the MSN Games network. The biggest aspect of this game is that it allows multi-player on a single grid by using MSN Messenger as a client. It offers a co-operation mode and a competitive versus mode. Puzzles are created seemingly randomly with three levels of difficulty.

In the co-operation mode, players work on the same grid by entering in numbers. Pencil notation is shared and both players can view where the other player's cursor currently is. They also provide a simple hint system that only shows what number goes in a particular cell.

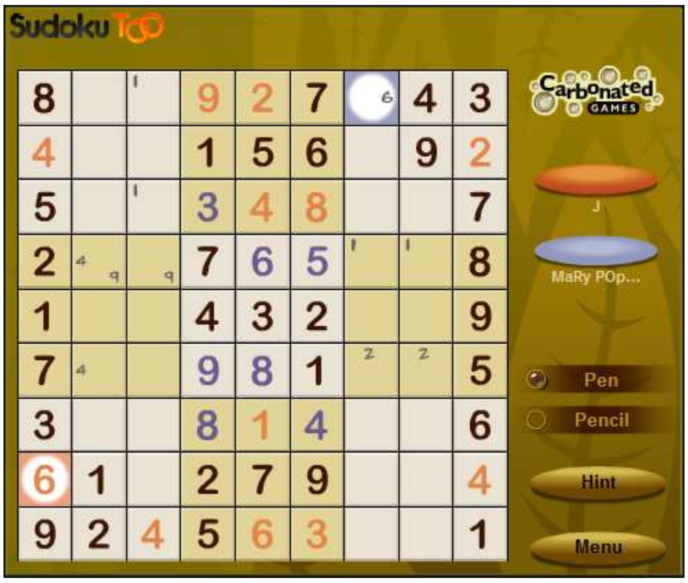


Figure 2.5: Sudoku Too interface by Carbonated Games[14]

Versus mode implements a competitive element into Sudoku. It is turn based, giving each player a certain amount of time for each turn. Players can only enter in three numbers on to the grid for each turn and after a turn ends points are added or removed from that player's score depending on whether entries were correct or not. Extra points are also given to the player with the most entries in a row, column, or sub-grid when that row, column, or sub-grid becomes complete. Throughout the whole game, markings can be made at all times and cannot be seen by the other player. The winner is the player with the most points by the end of it.

This type of play incorporates an extra layer of strategy involved, as experienced players have to choose specific places to go that will give them the biggest advantage, or give their opponent the biggest disadvantage. This gives a surprising amount of depth to the game.

The interface allows for both keyboard and mouse input and feels quite intuitive. However, it can also feel very unresponsive, due to its development in Macromedia Flash. It allows for chat as it is integrated into the MSN Messenger chat client.

Puzzles seemed to be on the easier side, however this may be due to the nature of the game, as it is a multi-player game and so a faster paced interaction is encouraged.

Summary: Sudoku Too incorporates a very interesting and unique Sudoku experience. It shows how multi-player elements can be incorporated into the game of Sudoku, using features such as playing on the same grid and using a turn based point system for its competitive aspects.

2.3.5 Zendoku

Zendoku[15] was developed by Zoonami for the Nintendo DS hand-held gaming platform. Rather then simply providing puzzles for the user to play, there is a larger gaming aspect involved. The

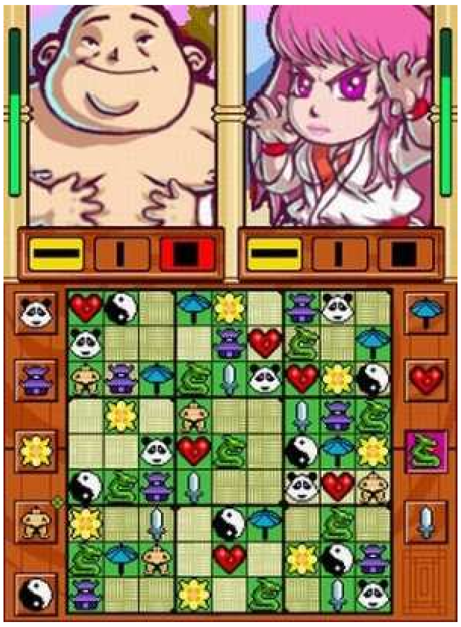


Figure 2.6: Zendoku interface for the Nintendo DS by Zoonami[15]

main game offers a multi-player experience where players get their own Sudoku puzzle to solve and must 'battle' to defeat their opponents. Each player has a life bar, and the more that a player fills in of his puzzle, the more that their opponent loses life. The player that completes their whole Sudoku grid first wins. To make it a more competitive and interactive experience, there is a 'mini-game' aspect involved. When a player completes a row, column, sub-grid or gets the complete set of a certain symbol, it sends a mini-game to the opponent's screen. This mini-game interrupts the player's puzzle and only the completion of the mini-game allows the player to continue on. The mini-games that are used including aspects such as blowing out candles, wiping items off of the screen, unrolling a scroll or tapping on certain areas of the screen, which all use the Nintendo DS's specific hardware. Numbers are also not used in each Sudoku puzzle, but instead symbols are used.

Modes included in the software include a single player story quest mode against AI players, a puzzle only mode (also available with numbers instead of symbols), a time trail mode, a mini-game mode, a multi-player battle mode and a multi-player co-op mode. It also provides a tutorial, 'infinite' amount of puzzles, and four levels of difficulty. Easy mode also provides hints during the game for players.

Zendoku uses puzzle generation to produce its own puzzles, which is quite uncommon for Sudoku titles on the Nintendo DS platform. It also does this without any detriment on the game play or loading screens. It uses the 'Dancing-Links' algorithm to produce a puzzle[25], and uses its own algorithm based on human solving techniques to grade a puzzle, for its AI implementation and for it to show a step-by-step solve for a puzzle. However, puzzles do not use any pattern based templates or symmetry. Puzzles are also not calculated on-the-fly but the game instead uses a buffer of puzzles to make it appear that they are generated instantly.

The biggest aspect of Zendoku is its multi-player mode and its orientation towards a full game instead of just providing puzzles for players to solve. It provides a good competitive nature for Sudoku and a sense of urgency in completion of a grid. This is one of the only Sudoku games that provides a multi-player aspect, and it essentially uses the fighting genre of games to produce a more interesting product. However, it only provides a battle mode where players play against each other on separate grids and a co-op mode where players play together on a single grid. It is clear that through the use of multi-player play, there could be many more other aspects that can be tried, such as a battle mode where players play on the same grid. The mini-game aspect in a game also feels more cumbersome then enjoyable.

Due to the manner of the completion of a Sudoku puzzle, a 'battle' generally is decided by the first person who gets to the closing stage of a game. This is because a puzzle generally gets exponentially easier the more cells that are filled in, and so by the end of the game a player is sending many mini-games to the opponent. This means that while the player is finding it easier and easier, the opponent is finding harder and harder as he also has to deal with completing his mini-games, generally meaning that the chance of the opposition winning gets exponentially smaller. It is clear that the developers also thought this and made the difficulty of mini-games reflect it in such a way that they become easier the bigger the gap is between opponents. However, having to suddenly complete a mini-game is bound to always interrupt the receiver's train of thought, regardless of its difficulty.

Although the story quest mode provides some sort of incentive to keep playing until completion, it is essentially only a list of games against AI opponents and does not provide the player with any real rewards for completing a puzzle. There also aren't any other gaming aspects such as levelling (to increase one's ability in the game, such as increase attack) or customization (obtain items to help during progress). Aspects such as these, which are common in games, may give the game the ability to keep a player's interest in the game longer.

The main game interface is also quite poor, as on screen buttons are used to select the symbol and the user has to place them on the grid using the touch screen. Unlike other attempts at Sudoku on the Nintendo DS that uses handwriting recognition to decide what number the player wants to place. No marking-up is also allowed in the game, which is considered vitally important in completing harder puzzles. However, the grading of puzzles seems to work rather well, giving the right set of puzzle for each level of difficulty.

Summary:

There is a lot to take from Zendoku, mainly the multi-player and gaming aspects that provide a much more interesting Sudoku experience. The use of the Dancing-Links algorithm also shows that puzzles can be created very fast and efficiently on a system with limited processing capabilities.

Chapter 3

Specification

This chapter contains the specification for the my Sudoku based game. The specification formally states its requirements and discusses its implementation. The aim of the this project is to explore the area of Sudoku in computing and to produce a piece of software of high quality. The resulting piece of final software can be broken up into essential and desired requirements.

3.1 Essential Requirements

The essential requirements are functionalities that must be fulfilled in the final piece of software. It must meet the following requirements:

- Create a Sudoku puzzle fast and efficiently with a wide range of varying difficulties.
- Solve a Sudoku puzzle from a given grid fast and efficiently.
- Allow players to enter in their own Sudoku grid so that it can be solved by the software or the player.
- Present the product in a well designed graphical interface, created for the Windows format as a stand alone application, allowing users to enter in entries in a range of methods via the keyboard and mouse. The final software should adhere to Nielsen's 10 usability heuristics (detailed below).
- Introduce a single player competitive aspect using AI. This is where one player plays against a computer player in a Sudoku based game.
- Provide a tutorial section so that players can learn how to play the game.

The final piece of software has to be able to create a Sudoku puzzle fast enough, so that there are no loading screens or noticeable pauses, as this would interrupt the flow of the game. It must also be able to provide a range of difficulties accurately so that it can cater to everyone who would want to use the software. This is a standard feature that most Sudoku programs already provide.

Generating and grading a puzzle may also have to incorporate many of the solving techniques and so the solver must be very fast and efficient. A fast solver is also essential so that users can input their own puzzles to be solved so that there is no noticeable delay from when they request a solve to when the results are produced.

An integral part of the software is that it is well designed so that it is easy to use and understand. This is because one of the main defining features that distinguishes this type of software from another is the interface. It must be easy to read what is happening and it must be immediately responsive to a range of inputs. Nielsen proposed the following 'Ten Usability Heuristics'[16]:

1. **Visibility of System status**
   The software should always keep the user informed and give immediate feedback about what is happening. It should display appropriate responses to the user's input.

2. **Match between system and the real world**
   The software should speak in the user's language. It should use words, phrases and concepts that the user is familiar with rather than technical terms.

3. **User control and freedom**
   The user should not be constrained by the software. The user should have the ability to easily leave a chosen option by selecting clearly marked buttons. It should support undo and redo options.

4. **Consistency and standards**
   An action should always have the same effect. Users should not have to wonder whether different actions mean the same thing. It should follow the standard conventions.

5. **Error prevention**
   The interface should be structured to avoid errors as best as possible. Error-prone conditions should be eliminated.

6. **Recognition rather than recall**
   The interface should not require users to memorize what every action does but instead should use recognizable features to explain its purpose. It should be intuitive or provide users with clearly visible or easily retrievable instructions.

7. **Flexibility and efficiency of use**
   It should be able to cater for both experienced and inexperienced users. The software could provide features that speed up interaction for an expert user. There should be a sense of customization so that users can tailor frequent actions.

8. **Aesthetic and minimalist design**
   The interface should not be cluttered so that only the important information and options are shown. It should not contain parts which are irrelevant or prevent the user from reading the page quickly.

9. **Help users recognize, diagnose and recover from errors**
   When error messages are shown, they should be simple, easily understandable, indicate the problem clearly and offer suggestions or options for a solution.

10. **Help and documentation**
    The software should provide a concise and relevant documentation so that necessary help can be provided to use the system. It should be easily accessible so that information can be reached quickly.

As there has not been many implementations of competitive elements to the Sudoku games, the final piece of software should allow players to compete against an AI competitor on a Sudoku grid. The AI should customizable in range of difficulties to cater for all Sudoku players.

A tutorial section should be provided for users who do not know the rules of Sudoku so that they can learn how to play. It should be detailed and easy to follow and explain the various techniques that are used in solving a puzzle.

3.2 Desired Requirements

The desired requirements of the resulting software are functionalities that are important but are not strictly necessary. It is desired that the resulting software has the following requirements:

- Integrate a larger gaming aspect in to the project, featuring role playing elements such as a story, character customization, items and a reward system.
- Introduce a multi-player aspect with other clients via a network connection.
- Allow multiple profiles for different users.
- Allow users to play various Sudoku variants
- Generate pattern and symmetrical based Sudoku grids.
- Publish Sudoku puzzles automatically into a publishable format.
- Include a medal based system that rewards players for continue play of the software. The more the user plays the game, the more medals that are given to the user.

A Sudoku based piece of software is usually a game product, however, few Sudoku games have actually provided as many features that would be expected of a game. Most Sudoku games only offer puzzles for users to complete and doesn't offer a great deal more a printed puzzle. However, software such as Zendoku[15] provides a mix of Sudoku, mini-games and fighting elements to offer a broader unique experience. The final piece of software produced should explore a different direction by incorporating role playing game elements, such as a story, characters, items, and a reward system. This will not only offer a more defined take on Sudoku based games but will also provide more of an incentive to play, longevity and greater interaction. Users should be able to participate in the story mode as an option, in which they can play as a character that will gain experience, fight other players and obtain items to enhance their abilities. This should all use the underlining competitive Sudoku game as the battle system.

Currently, there are also very few takes on multi-player aspects of Sudoku. The software produced should include a multi-player aspect that will allow two players to compete in real time over a network. Users should be able to search for games that have been set up on a network that they can participate in.

The remaining desired criteria adds more functionality to the over final piece of software. These are mostly small aspects but are highly desired and will increase the types of game play and longevity of the software.

3.3 Extensions

Subject to time issues, the following functionalities could be added to the project. These extensions are not required in the final piece of software but would enhance the functionality of the final piece of software.

- Allow players to create their own puzzle and share their creations over the network. This will allow players to download other people's puzzles so that they can try them out. Users will then be able to rate how fun other people's puzzles are. This incorporates the ideas of user-generated content, which generally increases the longevity of a product and encouraging the user community.
- Create a web-site to compliment the software, that also generates and solves Sudoku games at various levels, and documents and stores user information. Users can then play Sudoku puzzles on multiple platforms and on different computers with minimal set-up required. It would be ideal if users are able to rate the difficulty of puzzles that the web-site has generated so that it can then be seen how well the grading works compared to the general consensus of the users. A web-site based application could also serve as a type of demo for the final piece of software produced.
- Create a video game home console based version of the software, implemented to work on Microsoft Xbox 360. Networking between the PC version and the console version so that games can be played together. This will allow users to play the game on different environments, such as on the TV, which will cater to a greater audience.

Chapter 4

Further Research

This section outlines the possible technical choices and design options that have been looked at for the important aspects of the project. Possible algorithms for the project are outlined, detailing and discussing the possible methods and structures that could be used. Creation techniques for building a Sudoku puzzle are first discussed, followed by solving techniques and grading techniques. Possible interface designs are then discussed, followed by a look at possible frameworks for implementation.

4.1 Creation Techniques

Dating back to the 1970's[1], Sudoku puzzles were originally hand-drawn, and many printed puzzles still are[17]. Hand-constructed puzzles are usually created by first starting out with all the locations of the givens for the puzzle. Assignments for each given are only made when needed to make deductive progress. This allows the creator to have greater control over the general flow of the puzzle, as the solver will have to make the same deductions that the creator made, and so will have to follow the same path used to build the puzzle.

This allows for features such as symmetry to be added easily into a puzzle, as the placement of givens are defined at the start.

Although hand crafted puzzles can have a flow that not is not easily obtainable by computer generated puzzles, there is a large risk that a puzzle will either have more then one solution or no solution at all, resulting in an invalid Sudoku puzzle.

It is also worth note that there are 6,670,903,752,021,072,936,960 total number of valid Sudoku grids[9]. Of those grids, it has been found that there are 5,472,730,538 grids that are essentially different by taking into account relabelling, reflection, rotation and various permutations of sub-grids, columns and rows that will result in similar grids[10].

There are several computer based generation strategies (I have named them based on when givens are decided):

4.1.1 Known Sudoku Generator

The idea of this algorithm is to first generate a complete valid random grid, clear the grid and then continue to add randomly selected numbers from the solution on to the new grid until a certain number have been placed. This algorithm is outlined as follows:

1. Generate a completed random grid that meets all constraints set.
2. Clear the grid
3. Select a cell at random from the solution and add it on to the new grid until a number of k cells have been filled.
4. Check to see if it has multiple solutions, if it has then add another number and check again.
5. Check to see if it is the right difficulty, if it is not then discard and start again.
6. If the difficulty is acceptable, then a valid Sudoku puzzle has been generated.

The parameter k is the least amount of givens that will be provided for the puzzle, and is used to contribute in determining the difficulty of the puzzle. The data flow of this algorithm can be seen in figure 4.1.

Although it is simple and easy to implement, there is some inefficiency in that it effectively has to solve the grid at least twice. This is because it first solves it to generate a completed grid, and then solves it at least once more to check that it does not have multiple solutions. However, this also means that once it has found a complete grid, the puzzle is guaranteed to have a least one solution.

4.1.2 Complete Sudoku Generator

This method for producing Sudoku grids involves generating a complete Sudoku grid and then emptying the grid until a certain number have been removed. It then solves the puzzle to check for validity and difficulty. This algorithm is outlined as follows:

1. Generate a completed random grid that meets all constraints set.
2. Empty k number of cells randomly.
3. If it has multiple solutions, then add a random cell number from the solution and check again.
4. Check to see that it is an acceptable difficulty. If it is not then discard it and start again.
5. If the difficulty is acceptable then a valid puzzle has been generated.

The parameter k is selected before hand to determine the amount of numbers to be removed from the grid. This also contributes to the targeted difficulty rating of the puzzle.

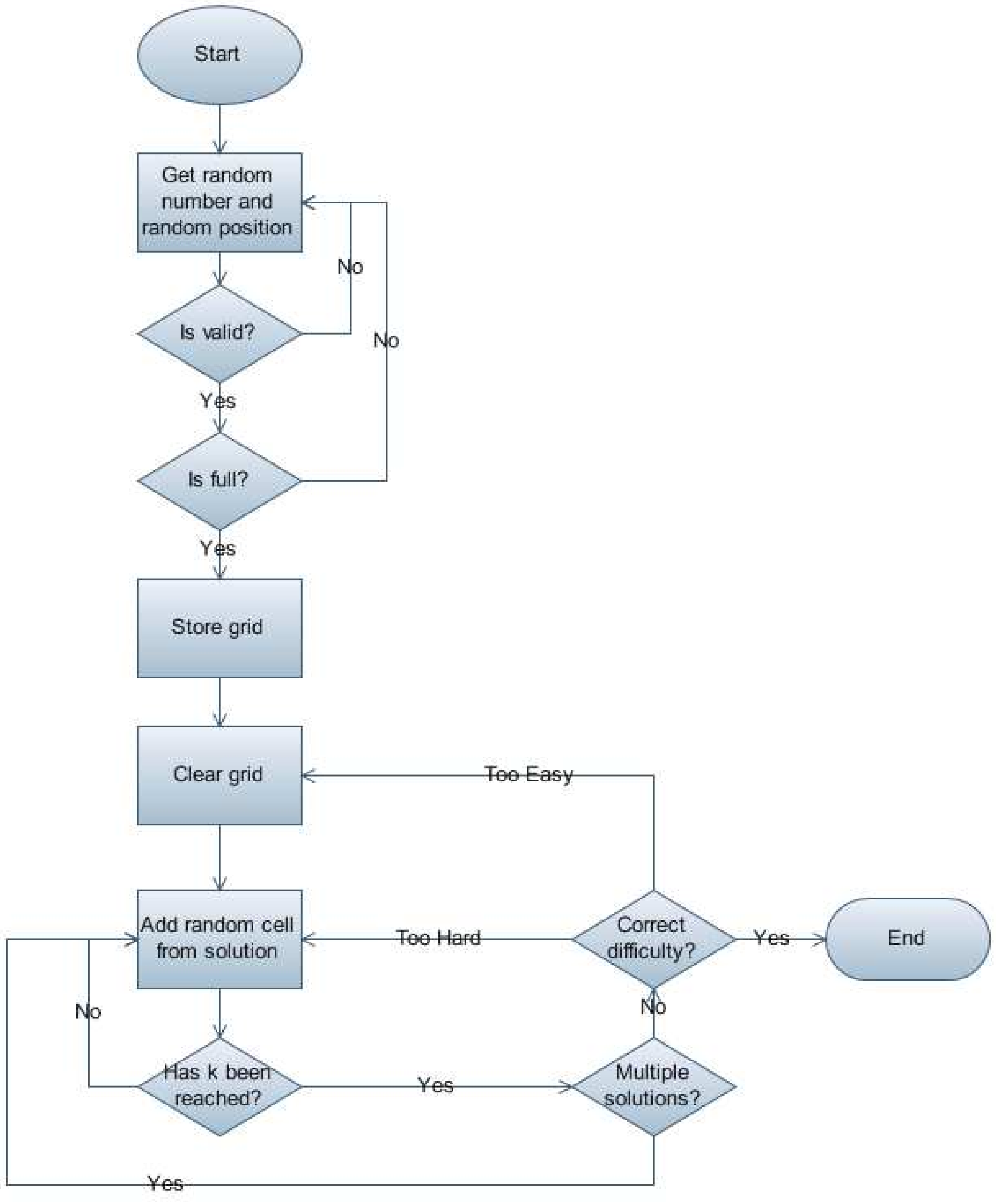


Figure 4.1: Flowchart for the Sudoku generation using the ‘Known Sudoku Generator’ strategy.

This is similar to the ‘Known Sudoku Generator’ method, but instead works back from the solved grid to remove elements. Therefore it also shares many of its qualities. However, it is potentially more inefficient. This is because there are generally more blank cells than there are givens in a typical Sudoku grid and so it has to remove more elements than the ‘Known Sudoku Generator’ method has to add elements.

4.1.3 Empty Sudoku Generator

The general idea of this algorithm is to randomly place a random number on the grid that does not violate the Sudoku constraints, until a certain limit of numbers have been placed. It then checks to see if it is a valid Sudoku puzzle. This algorithm is outlined as follows:

1. Place a random number in a random cell, without violating any of the constraints set.
2. Repeat step 1 until k numbers are inserted into the grid.
3. Check to see if the resulting Sudoku grid has a unique solution. If it does not have a solution then discard it and start again.
4. If it has multiple solutions then repeat steps 1 and 3 to add another element to the grid.
5. If it has a unique solution, then it is a valid puzzle. Check to see if it has the correct difficulty. If it does not then discard the puzzle and start again.
6. If it does have the correct difficulty then the puzzle has been created.

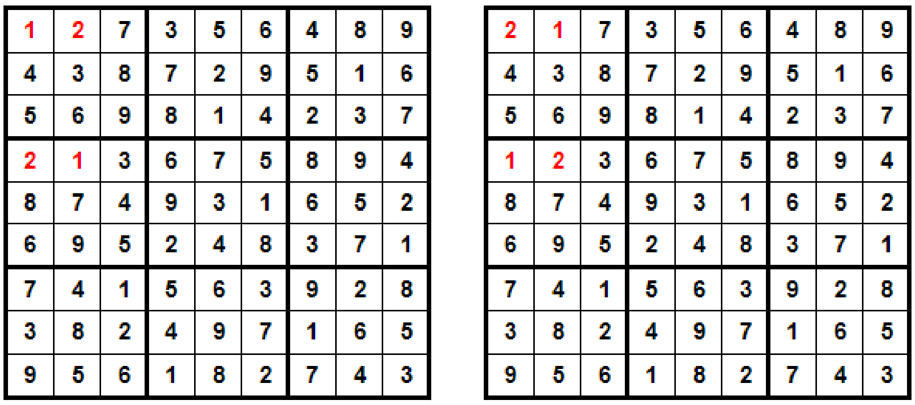
Parameter k is chosen before hand and defines the initial amount of givens on the grid, also contributing to determining its initial difficulty value. The choice of k has a large impact on the performance of the algorithm. If k<25 then it takes a long time as it takes longer to check if a puzzle has multiple solutions and if k>35 then the chances that it generates a Sudoku with no solution at all is higher.[18]

This is a simple algorithm that is easy to implement. It poses some efficiencies against the previous structures, as it generally needs to solve it less. This is because the other two structures need to solve the puzzle when filling in the grid and also when determining the givens to check for validity. This structure however only needs to solve the puzzle when checking for validity only, as this also solves the puzzle at the same time.

However, there is a noticeable inefficiency as if it does not have a solution at all then it has to be discarded. Backtracking could be used to counter this so that it tries a different number in a different cell. This means that it also potentially needs to solve more puzzles.

4.1.4 Unavoidable sets

The idea of this strategy is to identify the unavoidable sets from a given completed grid and then first begin by placing the minimum number of givens that cover all the sets. An unavoidable set



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

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

Figure 4.2: An unavoidable set with the highlighted numbers interchangeable

is a set of numbers in a grid where the numbers could be interchanged and it would still provide a valid Sudoku grid. This then means that it is *unavoidable* for the puzzle maker to place at least one of these numbers as a given, otherwise there would be multiple solutions. Figure 4.2 shows an example of an unavoidable set, where the highlighted numbers can be swapped to give two different solutions. This algorithm is outlined as follows:

1. Generate a completed random grid that meets all constraints set.
2. Find all of the most common unavoidable sets in the grid
3. Place the minimum set of clues that covers all sets.
4. If it has multiple solutions, add a random cell number from the solution and check again.
5. Check to see that it is an acceptable difficulty. If it is not, then discard it and start again.
6. If the difficulty is acceptable then a valid puzzle has been generated.

This method of creating a puzzle is more complex than the other algorithms. There needs to be a great deal of analysis in identifying unavoidable sets and the minimum number of clues that covers all of the sets. In general it will yield puzzles with fewer clues, resulting in a general greater depth of difficulty.

4.1.5 Summary

It can be noted that improvements can made overall to all of the algorithms. Most clearly is that when they have created an unacceptable level of difficulty for a puzzle, they could make an attempt to alter it so that the correct level has been reached. It is clear that if it is too hard then adding extra givens can be used to make it easier. However, if it was too easy, then making the puzzle harder would be more difficult and either backtracking or an substituting on the puzzle needs to be performed.

On conclusion, there are very few efficiency differences between the structures and there does not seem to be any large deficiencies to any of the algorithms. The 'Unavoidable sets' algorithm is the most intelligent and offers puzzles with the greatest depth of difficulty. However it is far more complicated to implement than the other algorithms. There is also no indication on how quick it will be able to find puzzles in comparison to the other structures. It also offers no clear solution to generating pattern based puzzles.

The ‘Empty Sudoku Generator’ presents a structure that uses the solving part to also varify its uniqueness at the same time and so offers some improvement on one side. However, this is to the detriment of producing a puzzle that *has* a solution, and so there are more inefficiencies when an invalid puzzle has been created.

The ‘Complete Sudoku Generator’ uses a completed grid to guarentee that there is at least one valid solution to the puzzle. It is also easy and simple to implement. However, ‘Known Sudoku Generator’ offers an improvement by choosing to add numbers rather than remove numbers when determining givens for a puzzle. It also offers a clear solution to add pattern based implementations, by simply choosing the locations when adding the givens.

For my final implementation, I have chosen to use the ’Known Sudoku Generator’ structure to create my puzzles as it is simple but efficient.

4.2 Solving Techniques

The strategy for solving a Sudoku puzzle is generally thought to consist of three processes:

1. **Scanning**
   This involves searching though the rows and columns that intersect with a sub-grid in order to determine whether a cell contains a certain number by using the process of elimination. This is generally done systematically when solving a puzzle by checking all digits the digits 1-9, typically starting from 1 and ascending to 9.
2. **Marking up**
   This process uses notation on the grid to help determine what candidate numbers can be placed on a certain cell. It is generally only limited to two of the same number in the marked up region, as any more would make it harder to read the puzzle. Marking up makes analysing the puzzle easier to perform as it is easier to spot double and triple sets for a row, column or sub-grid. It also generally speeds up scanning as well as it is easier to determine where a number that can only be in one of two places is placed if a different number is placed in one of those potential cells. This also suggests the next number that scanning may be performed on, which further speeds up the process.
3. **Analyzing**
   This involves logically deducing where a number can go by candidate elimination for a certain cell. One of the main ways to do this is to find doubles or triple sets for a region (row, column or sub-grid). The idea behind this is that if a there are two numbers that could potentially go only in the two same cells in a specific region, or if there are three numbers that could potentially go only in the three same cells in a region, then no other numbers could go in those cells. This now means that the number of possible placements for the other numbers for that region has decreased.

There are some specific known techniques that are used to logically solve a Sudoku puzzle and below are some of those techniques in general increasing order of complexity:

1. **Final symbol in a set** - When a row, column or sub-grid has eight completed cells, the final cell can be deduced.
2. **Naked Single** - When there is only one potential number that can go in a cell when its associated row, column and sub-grid contains the other eight numbers, then that number must go there. Figure 4.3 shows an example of a naked single.

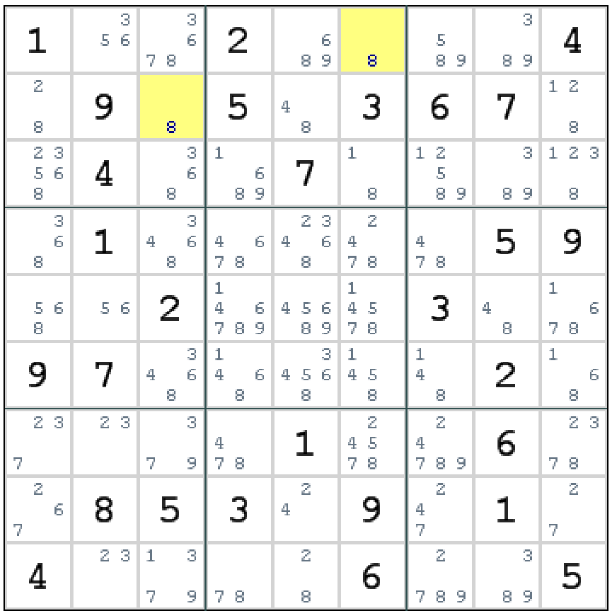


Figure 4.3: An example of naked single cells are highlighted in yellow. In these places, the digit 8 is the only candidate left.

3. **Hidden Single** - If a certain cell is the only one in a row, column or sub-grid that can hold a particular number, then it must contain that number. Figure 4.4 shows an example of a hidden single.

   (The remaining techniques are used to reduce the amount of possible candidates for a cell so that the first three techniques can be used to solve the puzzle)

4. **Naked Subset** - When only two numbers can go into two of the same cells in a row, column or block then the other possible placements of these numbers can be removed from that row, column or grid. This can be also used for more than two numbers. An example of this would be if a particular house has the following candidates:

   {1, 7}, {6, 7, 9}, {1, 6, 7, 9}, {1, 7}, {1, 4, 7, 6}, {2, 3, 6, 7}, {3, 4, 6, 8, 9}, {2, 3, 4, 6, 8}, {5}

   Here there are two cells that exclusively hold the numbers 1 and 7. As each cell must have a number and a house must have the digits 1 and 7, then these cells must hold either of these numbers. Therefore, these numbers can be removed from all other candidates lists for that house. In our example, this would result in:

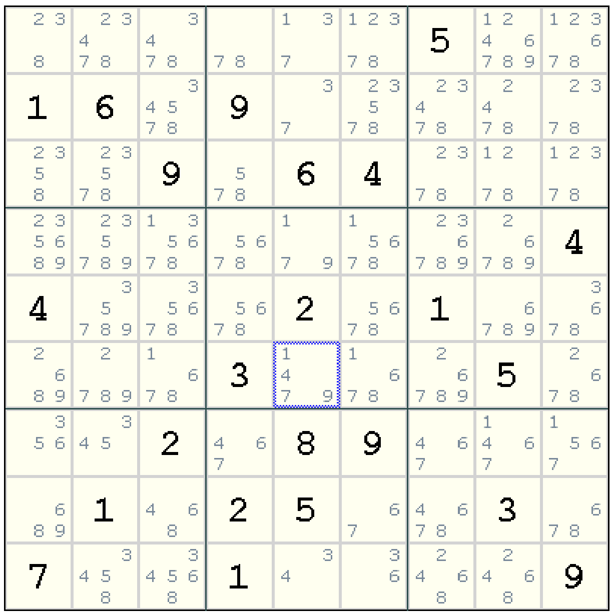


Figure 4.4: An example of a hidden single cell is selected. In the selected cell's row and sub-grid, this cell is the only place where the digit 4 can go.

{1, 7}, {6, 9}, {6, 9}, {1, 7}, {4, 6}, {2, 3, 6}, {3, 4, 6, 8, 9}, {2, 3, 4, 6, 8}, {5}

Again, the 6 and 9 are two further naked pairs and we can then further reduce to:

{1, 7}, {6, 9}, {6, 9}, {1, 7}, {4}, {2, 3}, {3, 4, 8}, {2, 3, 4, 8}, {5}

This results in a cell that now contains the single candidate 4 which can be solved using the Naked Single rule.

5. **Hidden Subset** - If two numbers can go into only two of the same cells in a row, column or sub-grid, then no other candidates can go in those cells. This can also be used for more than two numbers. An example of this can be seen in Figure 4.5.

6. **Locked Candidates - Pointing** - If all the candidates for a digit in a sub-grid are on a single row or column only, then all other occurrences of that digit can be removed from the other cells in that row or column. Figure 4.6 illustrates further how pointing locked candidates can be used to reduce the candidate lists.

7. **Locked Candidates - Claiming** - If all the candidates in a row or column are confined to a single sub-grid only, then they can be removed from the other cells in that sub-grid. Figure 4.7 illustrates further how claiming locked candidates can be used to reduce the candidate lists.

8. **X-Wing** - A more advanced technique, it is used when there are only two possible cells for a number in each of the two different rows and these candidates lie also in the same columns, then all other candidates for this number in those columns and rows can be removed.

9. **Swordfish** - Another advanced technique, it is a variation of the X-Wing pattern. This is when three rows each contain up to three cells that hold a matching locked candidate. This number must reside in each of the three rows and share the same three columns and so this

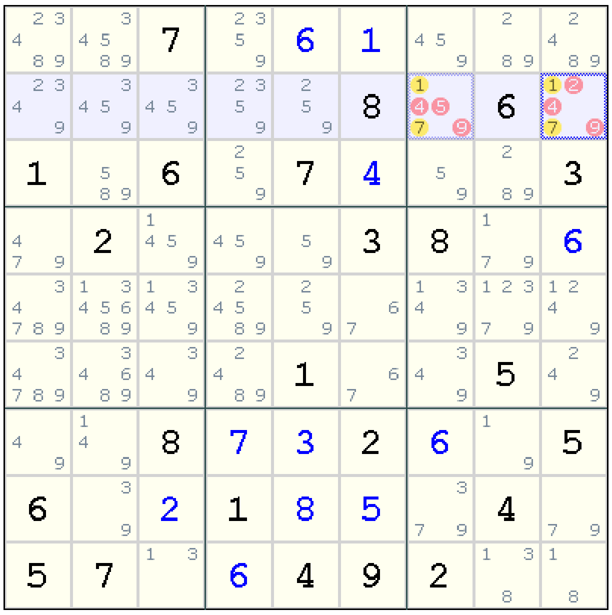


Figure 4.5: An example of a hidden pair. The pair can be seen highlighted in yellow and as these are the only two cells that these candidates appear in the row and sub-grid, they must go there and so all other candidates can be removed from those cells.

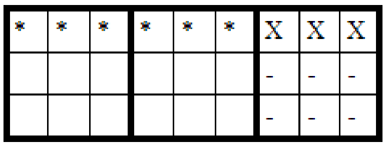


Figure 4.6: How a Pointing type of Locked Candidate works to reduce candidates. Here the X are the possible cells that can contain a certain digit. The - are the places that cannot contain that digit and so all of the cells with * can remove that digit from their candidate list.

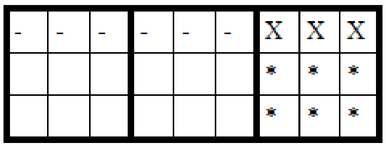


Figure 4.7: How a Claiming type of Locked Candidate works to reduce candidates. Here the X are the possible cells that can contain a certain digit. The - are the places that cannot contain that digit and so all of the cells with * can remove that digit from their candidate list.

number's other possible placements can be removed in those columns. This also works by using the columns instead of rows and vice versa.

In terms of a computer based Sudoku solver, there are generally three approaches that are used. They are presented as:

4.2.1 Human based solving methods

These solvers use human solving techniques to complete a Sudoku puzzle. They typically use a mark up matrix that shows the possible numbers that can go into each cell. A solver of this type has the possibility to be reasonably efficient, but also potentially has more work to do and so could work out slower, depending on how it is implemented. It is the most natural choice for grading a puzzle to see how difficult it is and would provide the most accurate answer. It also seems that using a human based solver would be good to show hints and explanations to solvers during a puzzle. It is effective in finding if a puzzle has a logically deduced solution, however it cannot easily determine whether a puzzle has multiple solutions as easy as other approaches.

4.2.2 Dancing Links (DLX)

The Dancing Links is a recursive, non-deterministic, depth-first, brute-force algorithm that was suggested by Donald Knuth as an implementation of his 'Algorithm X'[19].

Algorithm X is a technique used to solve all solutions to any exact cover problem. The idea is to represent an exact cover problem using a binary matrix, with each cell set depending on the constraints that are in the problem. The matrix will use columns to show the constraints and the rows as the possible choices. The idea is to choose a subset of rows in the matrix so that a set cell appears in each column exactly once. The resulting subset of rows is the choices made to obtain the final solution.

1. If the matrix is empty, then the problem has been solved.
2. Choose a column
3. Choose a row non-deterministically that has a cell in the selected column that is set to 1.
4. Include that row in the partial solution
5. For each column that in that row that has a cell set to 1, delete that column, and delete all rows that also have a 1 set in that column.
6. Repeat this process recursively on the matrix.

If a column appears to be entirely filled with unset cells then there is no solution with the current choices and it must backtrack to when it last chose a row and choose a different one. An example of Algorithm X applied to a 2×2 Latin Square can be seen in Appendix A.

To implement this algorithm more efficiently, Knuth suggested using the Dancing Links technique created by Hitotumatua and Noshita[20]. The aim of using Dancing Links is to reduce the amount of time searching for set cells when using Algorithm X. To do this, it uses a sparse matrix instead and only stores the set cells. Each node in the matrix will then only point to an adjacent node that has a set cell to the right, left, above and below it. Each row and column in the matrix uses a circular doubly linked list so that this is easier traverse. A special header row is also created that stores the amount of nodes there are in a given column.

The use of Dancing Links means that the time dedicated for searching the matrix for set cells (such as when selecting a row or a column) is decreased.

This is known to be a fast implementation for solving Sudoku puzzles. Also, by continuing on after it solves a puzzle, this type of solver will be able to easily determine whether there are multiple solutions to a puzzle. It can also be used for generating Sudoku puzzles as discussed earlier. However it will not be able to determine whether a puzzle requires guess work, offer logical hints or offer any help when grading a puzzle.

4.2.3 Stochastic search methods

These methods use a random-based search for their implementation. It first starts by randomly assigning numbers to the blank cells in a grid. The number of errors made is then calculated and the digits are then shuffled around until the number of mistakes reaches zero, which is when a solution has been found. Although this method appears to be fast, it appears to be reasonably complex, with the main intricacy stemming from the way in which numbers are shuffled. This algorithm however will not determine whether there are multiple solutions to a puzzle, determine whether a puzzle requires guess work, offer hints during solving or offer any help to grade a puzzle.

4.2.4 Summary

A stochastic search method offers an intelligent way of finding a solution but does not offer much for determining if there are multiple solutions or any grading help. It is also the most complex and would take longer to implement then the other methods.

Human based solving techniques offers the most when solving a puzzle, as it helps towards grading a puzzle and offering hints to players. However, DLX is known for being able to quickly find all the solutions to an exact cover problem.

For my final implementation, I have chosen to use Human based solving techniques as it will offer a far more accurate grading system. It will also offer logical hints to players. There is a concern for speed and for finding multiple solutions or finding a solution for puzzles that require guess work. For these situations, if the human based solver cannot be adapted, DLX may have to be used to quickly solve the problem.

4.3 Grading Techniques

Determining the difficulty of a puzzle is subjective and so there is no known mathematical way of deducing it. A puzzle that may seem hard for one puzzler, may be regarded as simple for another. However a strong attempt can be made based on using various methods. All of these aspects will need to be considered when grading a puzzle.

4.3.1 Number of givens

By using the number of givens that there are at the start of a puzzle is a very simple way to define the difficulty. Its implementation would simply be that the more givens that there are at the start of the puzzle, then the easier the puzzle becomes.

This would be very simple and easy to implement and would also be highly efficient to execute as little extra work would need to be done, however it would only provide a very rough idea of difficulty. This is because the difficulty is not based on the number of deductions required but by which logical deductions are needed and how long it takes to search for those deductions.

4.3.2 Difficulty of logic used

This aspect is important as there are a broad range of difficulties in the logical deduction techniques that need to be used to solve a puzzle. An easy puzzle should not need hard techniques to solve. This is regardless of how many givens there are from the start of the puzzle.

This would require using human based solving solutions to go through a puzzle and record each type of deduction it needs to solve it. These deductions would then have weights that would define how difficult a puzzle is. These weights can then be used to calculate how hard a puzzle is. This would provide an accurate difficulty grading as it is based closer to how a real solver would determine how difficult a puzzle is.

4.3.3 Width of the puzzle

This is defined by the amount of deductions that are available at a given time. If a user is at a certain point in the puzzle where there is only one logical deduction on the grid, a small width, it will then take a longer time to find that deduction. This is clear as how it will effect difficulty as if a puzzle has a large width then it will be easier to scan for a deduction, while it would take a lot longer when it has a small width, making it seem more difficult.

This is slightly more difficult to resolve as all possible deductions need to be found at each step. A single deduction will cause the width to go down by one but can potentially offer more available deductions, which would increase the width. Therefore, the choice in which deduction is used when solving will also have a direct effect in the next step's width and so the order in which deductions are made also determines the width.

4.4 Frameworks

There are various frameworks that will help me to implement and design my final piece of software. They all offer various advantages upon creating a graphical based product. Here is an overview of the main frameworks that were of interest to the project.

4.4.1 Popcap Games Framework

Created by PopCap Games[21], it is designed to let programmers easily and quickly create 2D puzzle style games. The main idea of the framework is to allow developers to concentrate more on the game design and logic instead of compatibility issues or technical multimedia problems.

It is principally a 2D game engine, but also supports 3D acceleration for graphical effects. It was used to create popular games such as 'Bejeweled', 'Peggle Deluxe' and 'Cake Mania'. It also uses C++ and works only on Windows platforms.

There is a small developers community that supports this framework, with the framework developers often active to help developers. Although there is little technical documentation, there are a few detailed tutorials, highlighting the usage of the framework.

This framework appears to be ideal for my Sudoku based game, as it will also be a 2D game and some what similar to other puzzle games that use the Popcap Games framework. The concentration towards 2D game graphics and game design would be the main advantage in using such a framework.

4.4.2 Microsoft XNA

Developed by Microsoft[22], XNA is a set of tools and a runtime environment for computer game development on Microsoft platforms. It is based on the .NET Framework across the Microsoft formats and promotes code reuse across the different platforms. The main idea is to use the framework to run games on multiple platforms with little or no modification to code.

It supports both 2D and 3D gaming, but with a larger concentration towards the latter. It uses C# and Microsoft's Visual Studio is solely officially supported. It also supports development on the Microsoft Xbox 360 gaming console and provides a networking API that uses Microsoft's LIVE services on Windows and Xbox platforms so that cross-platform networked multi-player games can be easily built.

Documentation for the framework is vast, with a strong active developer community. There are many detailed tutorials and guides showing the use of the framework to produce example software.

Microsoft XNA is a strong framework to develop games. It is vast and in-depth but also more complex. It is very suited to a Sudoku based game and there is also a large potential for development on the Xbox 360 gaming console.

4.4.3 Java Swing API

The Swing API for Java is part of Sun Microsystem's Java Foundation Classes[23]. It provides a graphical user interface for Java programs and includes a set of widgets such as buttons, text boxes, tables and split-planes. It is designed to provide sophisticated GUI components that map directly on to the current platform's graphical interface without any modification, giving it a native feel for the platform that it is run on.

The main advantage to using Java is that it is platform independent and so could potentially work on all Java enabled platforms. It is very easy to use and implement and is highly customizable. However, it does not provide any strong advanced graphical features and due to the Java architect, it is less efficient than other programming languages.

There is a vast amount of documentation and publications on the Java Swing API and many freely available tutorials. It holds a large developer community thanks to its easily approachable nature.

The Swing API is a reliable framework that is simple to use. A Sudoku based software can be easily created using widgets and give the user an integrated feel. Although it is slower in execution, it provides uniform behaviour across all platforms. However, it requires users to have Java installed on their system to run and it provides very limited graphical abilities.

4.4.4 Summary

For my project I have chosen to use Microsoft's XNA framework. This is because of the quality of the framework is known to be at a high standard, offering many features that are easy to implement. Using the framework, there would be a simple method to be able to implement networking and working with the Xbox 360 gaming platform.

Chapter 5

Game Design

This chapter presents an outline of the game's mechanics game designs. Each mode and aspect in the game is detailed. Screen prototypes and interface designs for each section is also discussed.

5.1 Main menu

The first screen that the user will see is the main menu. This is a simple interface and serves as a central hub for selecting the different modes in the game. A screen prototype for the interface is shown in figure 5.1. This is a clear interface with each button clearly labelled and visible. It also shows the title of the game clearly. These are important interface design aspects so that the user easily understands everything.

5.2 Quest Mode

The Quest mode is designed to attempt to incorporate Role Playing elements and major computer game aspects into traditional Sudoku puzzles. It is a single player experience in which players take control of a character in a story. The idea is for the player to go around a map world, under take certain quests and defeat enemies. Completion of quests result in rewards such as items, money, experience points and progression in the story. Once a player is engaged in an enemy, a battle occurs in which a player has to compete against an AI opponent in a game of Sudoku.

I have chosen to base the designs and artwork on the American comic book 'Battle Chasers' by Joe Madureira, published by Wildstorm and Image Comics[24]. This is mainly due to time and resource issues, as this will mean that I can focus the project on the development of the game, rather than the story and artwork. A short story based on these designs has been created to show off each aspect of the game that will be used. The story is short mainly due to the short scale production time and values (compared to traditional game productions).

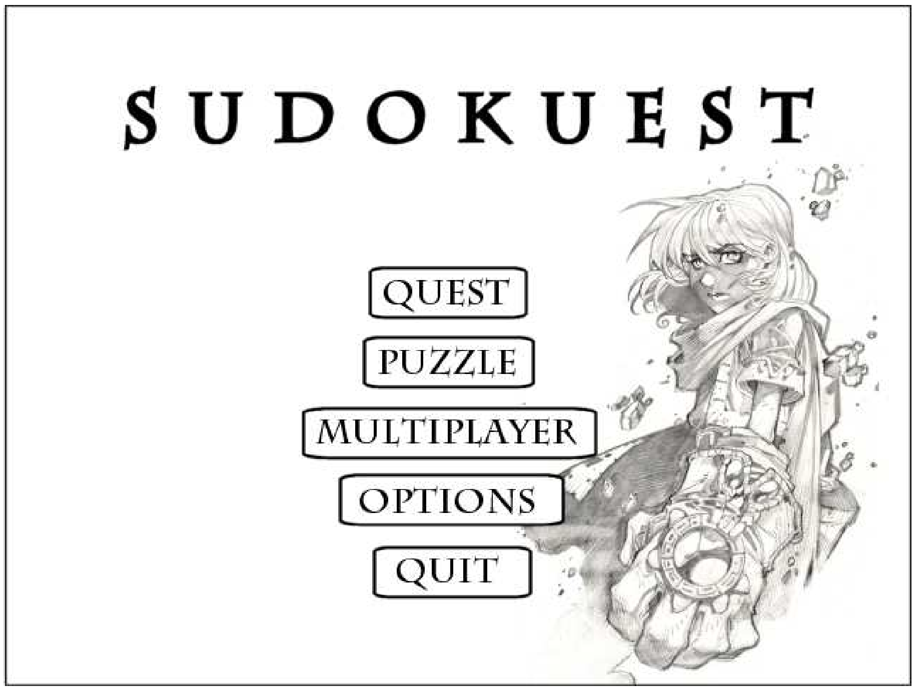


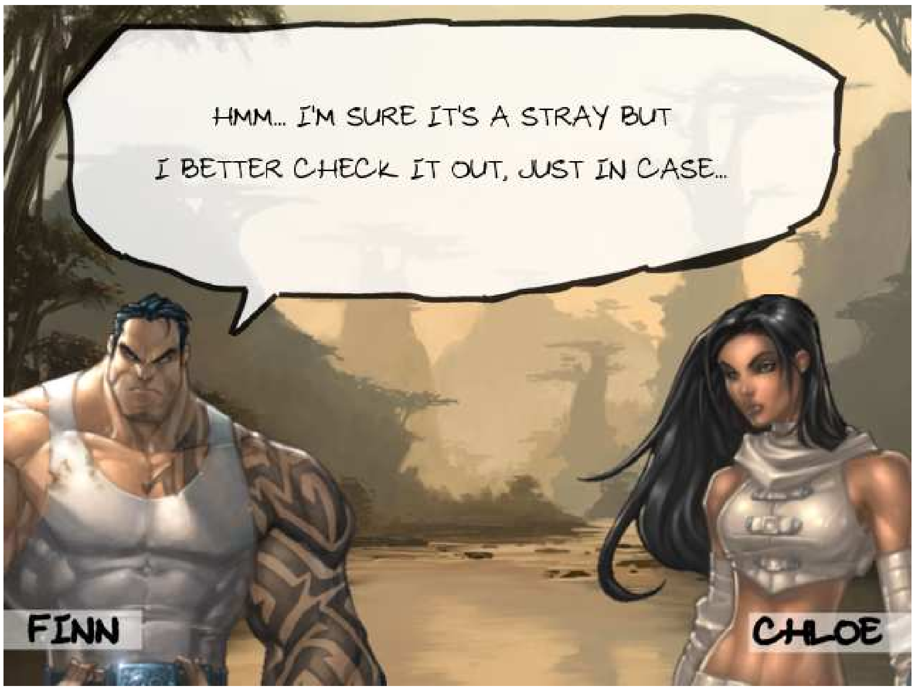
Figure 5.1: A screen prototype for the main menu

5.2.1 Story

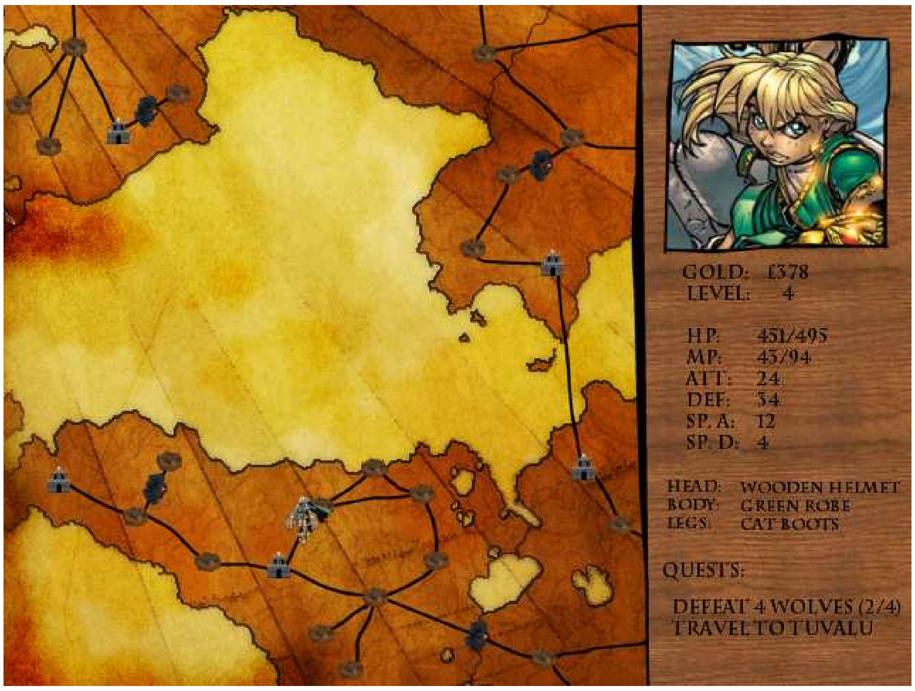
The story is set in a traditional fantasy world, with an Early European Middle Ages theme and players will take the role of Gully, a female child protagonist. Below is a short synopsis of the story.

The story starts with Gully finding an injured man in the middle of the road. The man then instructs Gully to give a message to travel to a sorceress and give her a message. On her way to the sorceress, Gully is attacked by a wolf. Upon defeating the wolf, she delivers her message and it is revealed that they must travel to a volcano to stop an evil demon from awakening. Once they travel to the volcano, Gully then fights him. The script for this can be found in Appendix B.

Much of the story will be told through dialogue scenes between the characters. A screen prototype of a dialogue scene can be seen in figure 5.2a. The main idea of scenes are to progress the story and so the interface is designed so that it gives the user an atmospheric feel around what is happening. For this reason, the background fills the entire screen and the characters talking are clearly visible. A comic style speech bubble is also used to show the text to add to the style of the game. Players will be able to click to move on to the next piece of dialogue. It is very straight forward and simple so that the user can concentrate on the story as easily as possible, so that they can follow it without problems.


HMM... I'M SURE IT'S A STRAY BUT I BETTER CHECK IT OUT, JUST IN CASE...
FINN
CHLOE

(a) Mock of a dialogue scene screen



GOLD:	£378
LEVEL:	4
HP:	451/495
MP:	43/94
ATT:	24
DEF:	34
SP. A:	12
SP. D:	4
HEAD:	WOODEN HELMET
BODY:	GREEN ROBE
LEGS:	CAT BOOTS

QUESTS:

DEFEAT 4 WOLVES (2/4)
TRAVEL TO TUVALU

(b) Mock of the map screen

Figure 5.2: Dialogue scene and map screen prototypes

5.2.2 Map

The main interface that the player will be presented with in Quest Mode is the overview of the world map. The map represents where the player, towns, cities, roads, dungeons and enemies are. This interface then dictactes the movement of the player.

The map features all of the key destinations that the player can currently go, with roads that they player can use to travel to these key destinations. The map is a type of networked graph, with each key destination a node on that graph, and each road between destinations is an edge in the graph.

A node in the map can be:

- **City** - Consists of an armor shop, a potion shop and a quest list. Players can purchase items using gold to increase certain attributes of thier character. Attributes of their character are discussed in more detail in subsection 5.2.3. They can also undertake a quest from the quest list. Quests are also discussed in more detail in subsection 5.2.4.
- **Town** - Does not contain anything of special. Players may have to travel through a town in order to get to where they need to go, or complete a quest there if the quest requires them to complete it there.
- **Dungeon** - A place that are enemies main hideouts. Players complete a dungeon by facing a specified number of opponents from it in a row. They are then rewarded with gold and/or items. There may also be an unusual large amount of enemies on the roads near a dungeon.

As the player takes part in more quests, more nodes and edges are revealed to them, increasing their visible size of the map. Enemies also appear randomly on edges and are visible to the player on the map. When a player crosses the path of the enemy, they engage in a battle.

Figure 5.2b shows the screen prototype for the map interface. This has been designed so that the player can easily see everything on the map, such as their character, enemies, destinations and roads. This clearly shows how all the destinations link up with each other and also shows where enemies are so that the player knows when they are about to battle them. The old style map design also adds to the fantasy themed setting.

Details about the user's character is also outlined and detailed. This is so that the user can see important information such as their gold, items and quests. It is important that they can see it on the same screen as the map so that they have quick and easy access to this information.

5.2.3 Battle

The main part of the game is the battle state. This is based on the Sudoku puzzle, where players compete on a single shared grid simultaneously. They both have health meters and an entry into the grid causes damage to their opponent's health. Extra damage is caused when there is a completion of a row, column or sub-grid. The winner is the one who depletes their opponent's health. Once a grid is completely filled in, a new grid is given.

A level based system is used to determine the player's initial attributes. Each time they defeat an enemy or complete a quest, they obtain experience points (XP) and once they get their XP to a certain threshold, their level goes up. As their level goes up, the player's attribute are then permanently raised.

A player has these attributes:

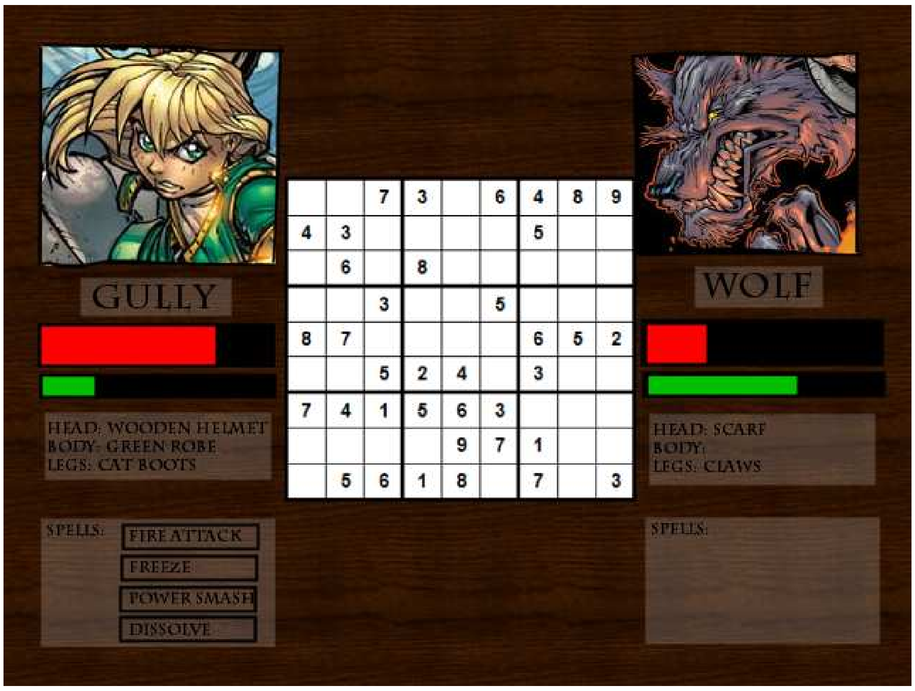
- **Health Points (HP)** - The amount of energy the player has.
- **Magic Points (MP)** - The amount of magic power the player has. Magic is used to cast spells (discussed below).
- **Attack** - How powerful the player's attack is.
- **Defence** - How well the player's defence is when being attacked.
- **Special Attack** - How powerful the player's magical spell attack is.
- **Special Defence** - How well the player's magical defence is against spells.
- **Health Points Regeneration Rate** - How quickly a player's HP regenerates during a battle.
- **Magic Points Regeneration Rate** - How quickly a player's MP regenerates during a battle.

When one player attacks another, the amount of damage that is dealt is based on the attacking player's attack attribute, the defending player's defence attribute and whether a row, column or sub-grid is also being completed.

Each player can hold a maximum of four items. Certain items can also temporarily boost certain attributes and these items can be bought from a shop or obtained through quests. Potions can also be used during a battle but are limited to the number that they have in the inventory. Potions can do such things as heal, or increase attack power.

Spells can also be cast at any time during a battle. These spells can range from a simple attack spell that damages opponents or spells that doesn't allow the opponent to make a move for a period of time. All spells use a certain amount of MP and cannot be cast if the player does not have enough MP. Spells are gained when reaching certain levels.

A screen prototype of the battle interface can be seen in figure 5.3. For this screen it is important to show not only the puzzle board but also the relevant information such as the player's health, items and spells that they can cast. It is also important to show the competitor's information as well so that users can visibly see who they are facing and so this gives a more immersive game play experience. Showing the competitor's health is also important so that users can obtain visible feedback from when they damage the enemy.



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

Figure 5.3: Mock of a battle screen

5.2.4 Quests

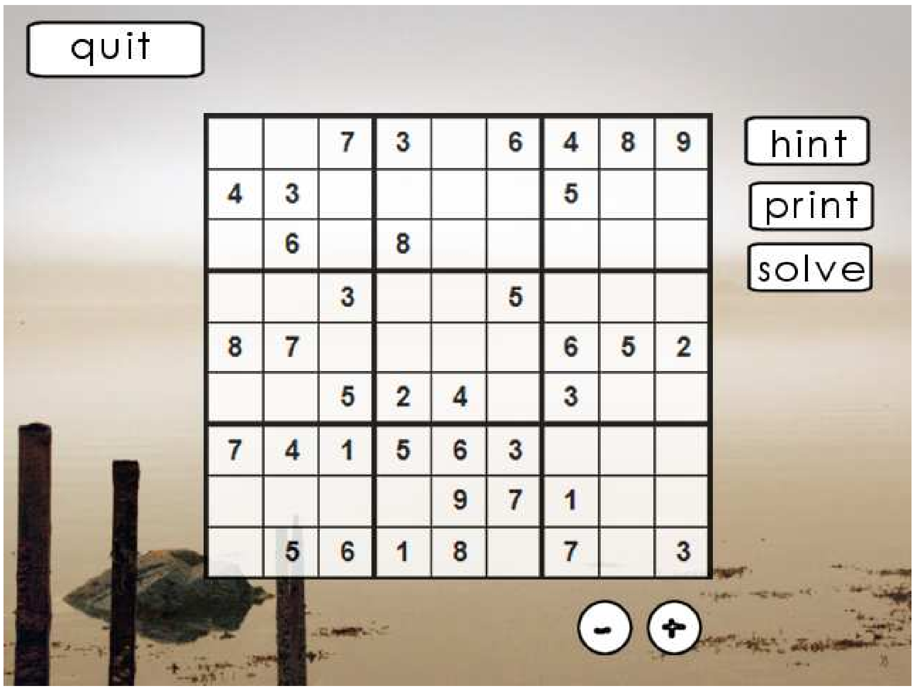
There are two types of quests, a main quest that progresses the story, and a side-quest that does not. Quests can involve things such as having to go to a certain town to deliver a message or clear a dungeon. Completion of a quest results in rewards such as gold, XP, and/or items.

5.3 Puzzle Mode

The puzzle mode presents the original Sudoku game in its original format. Like Zen of Sudoku[12] it hopes to offer users a calm and serene Sudoku experience. Players can select what type of Sudoku puzzle they would want from the various variants available and a range of difficulties. They can then enter and mark up numbers into the grid to solve it. To remove the element of trail and error not associated with a traditional puzzle, any number can be accepted into the grid, regardless of whether it is correct or not.

It will also offer hints to the player, by flashing the next logical step available or showing an incorrect cell and also offer a simple explanation if needed. The puzzle can also be printed out and user can select to show the solution throughout. They can also change the size of the how the puzzle is presented so that they can adjust it to how they want while playing.

Players will be treated to a calming background and soothing music to enhance the relaxing atmosphere. A tutorial difficulty will also be offered to teach players how to play Sudoku.



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

Figure 5.4: Mock of the puzzle screen

Figure 5.4 shows a screen prototype of the puzzle mode screen. This has been designed with a strong emphasis on calmness and simplicity. All the features which are necessary are there and clearly visible such as the buttons and grid and all irrelevant information has been minimalized. Rounded edges have also been used to enhance the calm feel to it.

5.4 Multi-player Mode

This mode aims to bring a networked multi-player Sudoku experience to the game.

This will allow players to compete against each other on a single grid simultaneously, similar to the battle system in the Quest mode. The characters that they use will be from the quest mode and so they will keep their attributes, spells and items. This means that they will each have a health bar and will damage the opponent after they have entered a number into each cell.

The way that the multi-player mode will work is that a player will decide to host a game. They will then connect to a central server, which will register that they are hosting a game. Another player will then decide to look for a game and then they will choose a game from a list of hosts sent to them from the central server. This client will then connect to the player hosting to play the game.

5.5 Print Mode

This will allow users to select a type of difficulty and the number of puzzles that they want to print. This will then give players to print the selected amount of puzzles using their default printer. Solutions can also be printed out at their request.

Chapter 6

Sudoku Implementation

This chapter details the implementation of the Sudoku aspects of the project, including puzzle generation, solving and grading. The techniques used for each aspect are explained in detail.

6.1 Dancing Links (DLX)

As suggested in Section 4.2.2, the Dancing Links (DLX) algorithm is able to efficiently solve Sudoku puzzles by representing them as a constraint satisfaction problem. The strategy used to generate a complete Sudoku puzzle lies in solving an empty grid.

6.1.1 Brute force solving

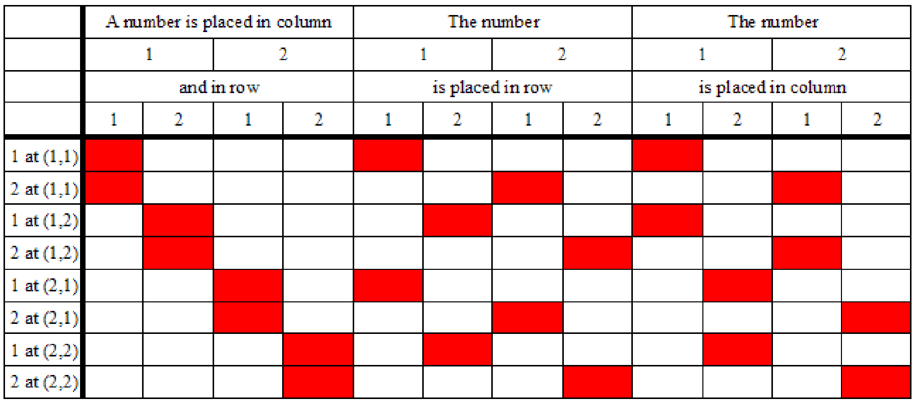
The main outline of the algorithm for Algorithm X is explained in Section 4.2.2 but a detailed step-by-step example is explained below.

The application of Algorithm X can be applied to all constraint satisfaction problems, and so we will examine an example applied to a 2×2 Latin square (so that in each row and each column, there are the numbers 1 or 2). An example of a solved grid is given in table 6.1.

For a Latin square there are three types of constraints:

1	2
2	1

Table 6.1: An example solution of a 2×2 Latin Square



	A number is placed in column				The number				The number			
	1		2		1		2		1		2	
	and in row				is placed in row				is placed in column			
	1	2	1	2	1	2	1	2	1	2	1	2
1 at (1,1)	🟥				🟥				🟥			
2 at (1,1)	🟥						🟥				🟥	
1 at (1,2)		🟥				🟥			🟥			
2 at (1,2)		🟥						🟥			🟥	
1 at (2,1)			🟥		🟥					🟥		
2 at (2,1)			🟥				🟥					🟥
1 at (2,2)				🟥		🟥				🟥		
2 at (2,2)				🟥				🟥				🟥

Figure 6.1: A binary matrix implementing Algorithm X for a 2×2 Latin Square

- We must place a number in each cell.
- We must place each number in each row.
- We must place each column in each row.

This creates a binary matrix with the twelve constraints acting as columns and each of the eight ways of placing a number as the rows as shown in figure 6.1. Each cell in the matrix is marked so that each number placement corresponds to the associated constraints.

Algorithm X dictates that we first pick an unsatisfied constraint, then pick a row that satisfies that constraint. That row is added to the solution set and all conflicting rows are then deleted. This process is repeated until all constraints are satisfied and backtracked if an unselected constraint has no more candidates. A 9×9 Sudoku grid has 324 different constraints and 729 ways to place a number [26]. Appendix A shows a step-by-step run through of this algorithm, applied to our Latin Square example.

Although it is a brute force technique, the solver proved highly efficient in practice. Figure 6.2 shows the speed differences of my human solver program (detailed in section 6.2) compared to the DLX implementation, running on the same computer set up. As we can see from the tests, the DLX gave approximately an 1000% increase in speed.

6.1.2 Testing for multiple solutions

To test for multiple solutions, the solver simply carries on with the solver after it finds its first solution, backtracking and solving until it finds another solution. If it has parsed the whole search tree without finding another solution, then it is unique, otherwise it is not.

At this stage, I found that a further speed up in the DLX implementation could be achieved by taking advantage of DLX's use of storing how many rows each constraint has left. Using this, I found that by selecting the constraint with the smallest amount of rows, the solver used a fewer amount of search nodes to prove that a puzzle has a single solution. Under 1,000 randomly generated puzzles at varying difficulties, I found that on average this speed up used 3,564 nodes

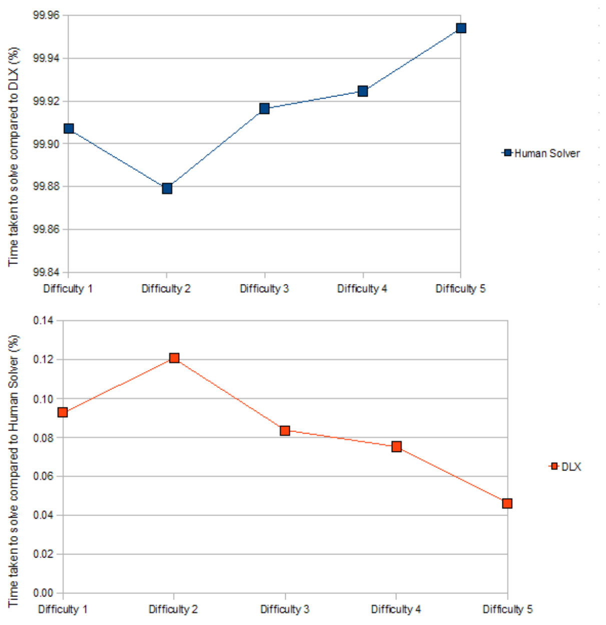


Figure 6.2: Graphs comparing the speed differences of my DLX solver and my Human solver. Separate graphs are used as data ranges between the two types are too vast to put on the same figure.

for each puzzle, as opposed to the 517,431 nodes on average that it took without the speed up for the same puzzles. This is approximately 145% speed increase. The distribution between each difficulty and how many nodes the two different solvers used in their tests are shown in figure 6.3. As we can see in figure 6.3 the range and distribution between the solvers for each difficulty is quite large, with difficult puzzles proving far easier to solve using the sped up solver but with easy puzzles giving a much smaller difference in efficiency.

6.1.3 Generation a complete grid

The generation of complete Sudoku grids is based on using the solver on an empty grid, and making random choices when selecting which row to choose that satisfies a constraint.

I found that a simple speed increase could be attained by first populating the top row randomly with each nine symbols and then feeding that grid into the generator. this gives a general speed up of 4% against generating with an empty grid.

6.1.4 Generation a complete grid

Using the strategy outlined in section , givens were added from the solution to the grid rather then being removed, until the solution was unique. The idea of this is that there would be less givens to be added to the puzzle than would be needed to be taken away. A test of 1000 generated puzzles using this strategy was run to give the distribution of how many givens were generated,

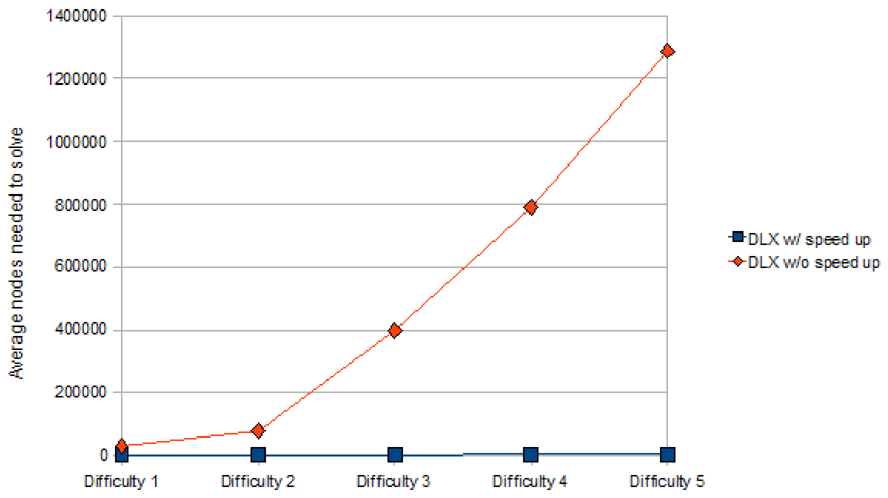


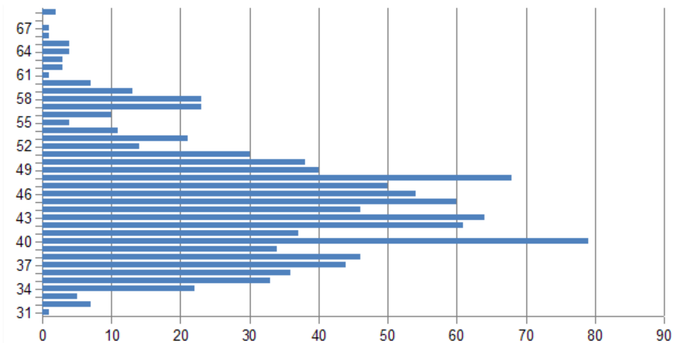
Figure 6.3: Graph showing how many nodes are needed to solve using a DLX solver with the speed up and a DLX without.

as shown in figure 6.4a. While I was expecting a range closer towards 20-25, it was instead far higher, with an average of 43. Running these puzzles through a grader (detailed in section 6.2) also showed that the puzzles generated were highly skewed towards the easy difficulty setting. The main problem that I found with this is that it would then be very difficult to produce a hard puzzle, as I would have to either keep on recreating puzzles until a hard puzzle is provided or remove as many givens as I could in hope that it would produce a hard puzzle.

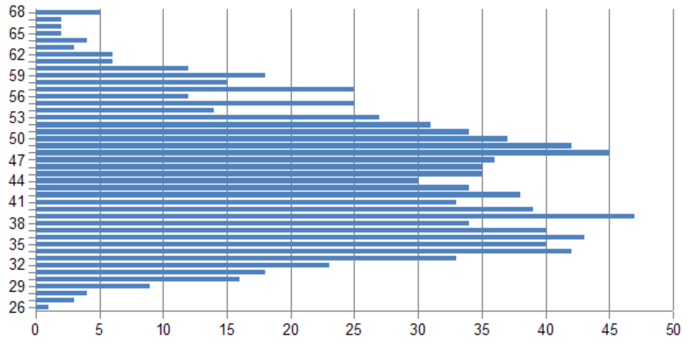
Instead, I chose to attempt to follow the strategy detailed in section , randomly removing from the solution until it removes in a case where the puzzle is no longer unique. By running the same test of generating 1000 puzzles, I received the distribution as shown in figure 6.4b. Although still not as adequate as I had hoped, it still gave a general distribution that was averaged towards a lower number, with the average being 39. Again, the puzzles produced were generally towards the easier side.

To further improve the strategy, once initial givens were removed, I attempted to remove givens incrementally that could be removed while still making it a unique puzzle. This gave the test result as shown in figure 6.4c. This was much closer to a more sufficient number, with an average towards 24 givens. The puzzles produced were also very hard, which meant that I would be able to adjust puzzles easier to other difficulty ranges (as it is easy to make a hard puzzle easy but difficult to make an easy puzzle hard). However, the problem with this solution is that because it incrementally tried to remove unnecessary givens, the puzzles produced tended to have many of the top elements empty and many of the bottom elements of the puzzle filled.

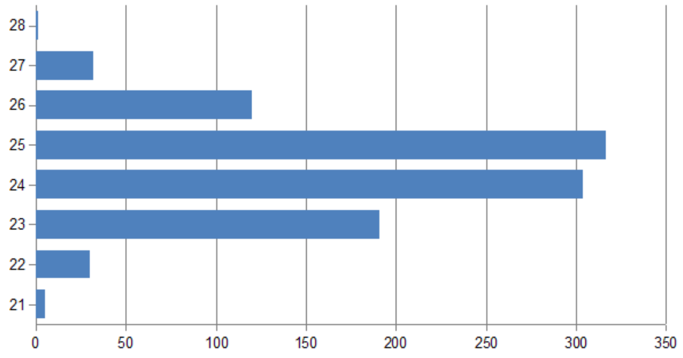
My final attempt at producing a sufficient generator was to instead randomly remove givens to make sure that no other numbers could be removed. This was done by generating a random permutation of the remaining cells in the puzzle that had givens and checking each one to see if it could be removed. The test results are shown in figure 6.4d, and as we can see it still produces the desired results but now with a much more even grid.



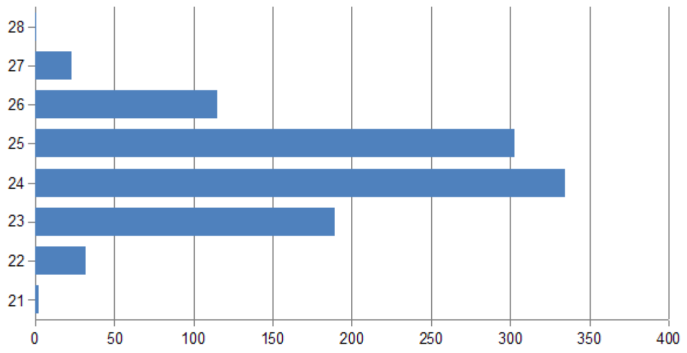
(a) First implementation: Add givens until unique



(b) Second implementation: Remove givens until unique



(c) Third implementation: Remove givens until unique, then incrementally remove until cannot remove anymore



(d) Forth implementation: Remove givens until unique, then randomly remove until cannot remove anymore

Figure 6.4: Tests of each implementation to see the number of givens from generating 1000 puzzles

6.2 Grading Techniques

To grade the difficulty of a puzzle, I implemented a range of known human-like techniques used to solve. These techniques are outlined in section 4.2. Other than grading, these techniques can also be used to offer hints to players and a method for AI in the battle versions of the game. Difficulty is based on what technique is needed to solve a puzzle. The reason that I used the hardest technique needed as a difficulty measure, as opposed to using a cumulative measure, is that I found that there is no amount of easy techniques which equates to a single hard technique.

The techniques are distributed between difficulty levels in the following way:

- Difficulty 1: Final Single, Hidden Single (Box)
- Difficulty 2: Naked Single
- Difficulty 3: Hidden Single (Row, Column)
- Difficulty 4: Locked Candidates, Hidden Pair, Naked Pair
- Difficulty 5: Naked Triple

The 'width' of a puzzle is also taken into account for easy puzzles. The grader uses the average of the width for a difficulty 1 puzzle to determine if it is small enough to be a difficulty 2 puzzle. This is because I found that a difficulty 1 puzzle has a large range in difficulty, and so this spreads the distribution out.

6.3 Application

Functions were then created based on my Sudoku implementation so that it can be used easily in the overall project. The `int[][] create(int box_size, int desired_difficulty)` public static function were made so that only a size and difficulty needs to be given and a puzzle, and its solution we be returned.

Similarly, a `KeyValuePair<int, KeyValuePair<int, int» solve_one(int[] sudoku, int box_size)` public static function was made, which from a puzzle, gives a solution to a single cell in the puzzle and the solving technique of which it came to that solution.

A `List<int>[] find_candidates_list(int[] sudoku, int box_size)` function was also created, that returns all of the immediate candidates for each cell for a given puzzle. These three functions are then able to be used throughout the remaining of the project quickly and easily.

Chapter 7

GUI Implementation

This section discusses the implementation of my game, with details about each aspect and information on how they were implemented where appropriate.

7.1 Structure

Figure 7.1 shows the class structure of the project. The overall structure is based on the game state and each screen that is used to represent each of those states. Figure 7.2 shows a state diagram of the project, which uses menus to transition from each state. The sample provided at the XNA Creators Club web-site[27] on Game State Management development was used initially as a basis for the project and provided a simple structure for displaying and navigating menus and handling input.

Each screen class that was implemented extends the abstract GameScreen class so that polymorphism can be used to deal with common functions needed for each screen, such as screen transitions, handling input and drawing. There is also a further abstract MenuScreen class that extends the GameScreen, which is used for all menu options throughout. Each individual menu screen is a subclass of the MenuScreen, showing each of the menu options available for that particular menu. Figure 7.3 shows a screen shot of the main menu.

The ScreenManager class handles what screen classes are currently being shown. Each screen can either be added to the current screen (stacked on top of the current screen, such as a menu screen) or can be loaded from scratch so that each previous screen in the stack is unloaded. The ScreenManager class also deals with what screen is on top so that it can detect what screen is taking in the input, to prevent inactive screens from also reacting to input or updating.

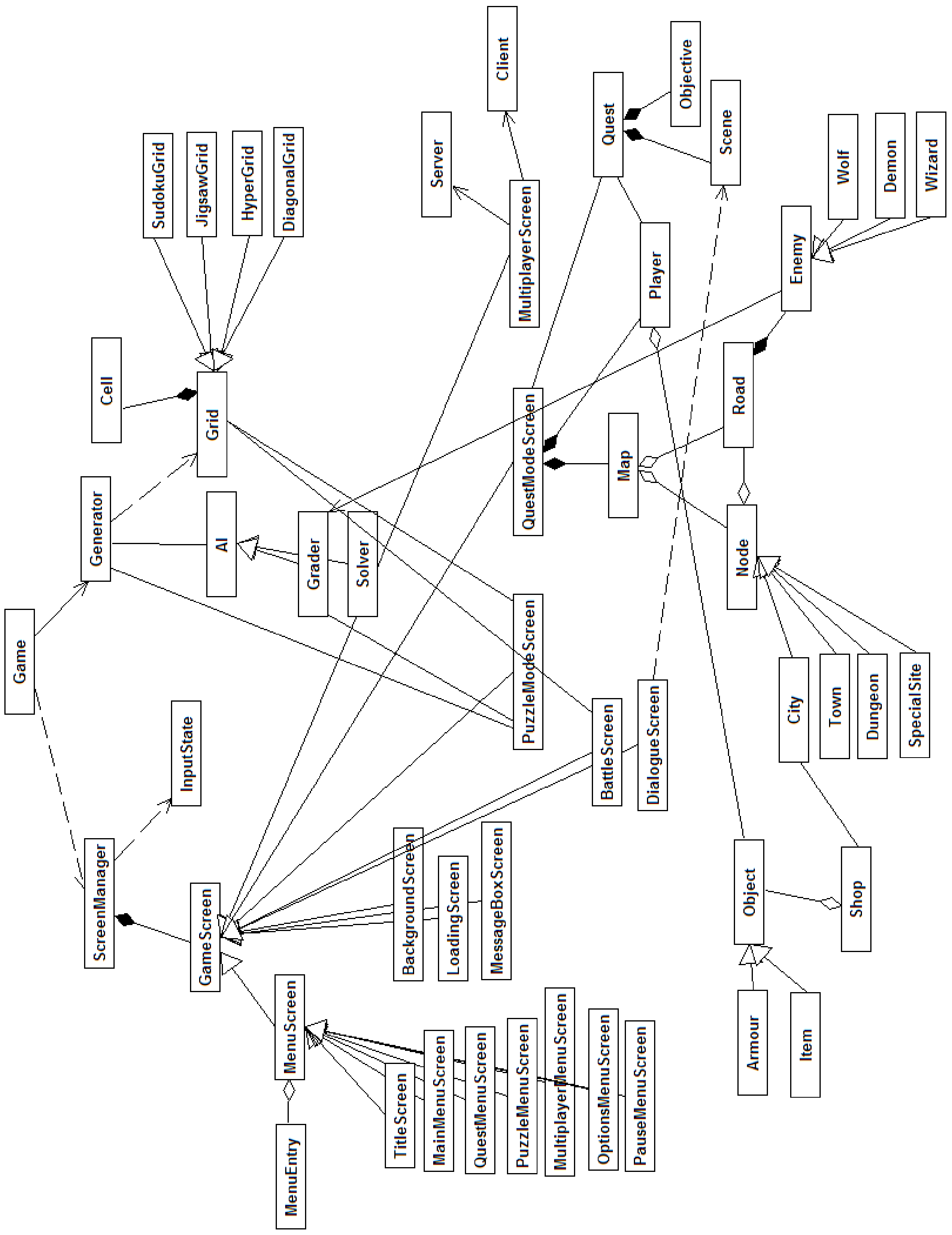


Figure 7.1: UML Class diagram showing the class structure of the program

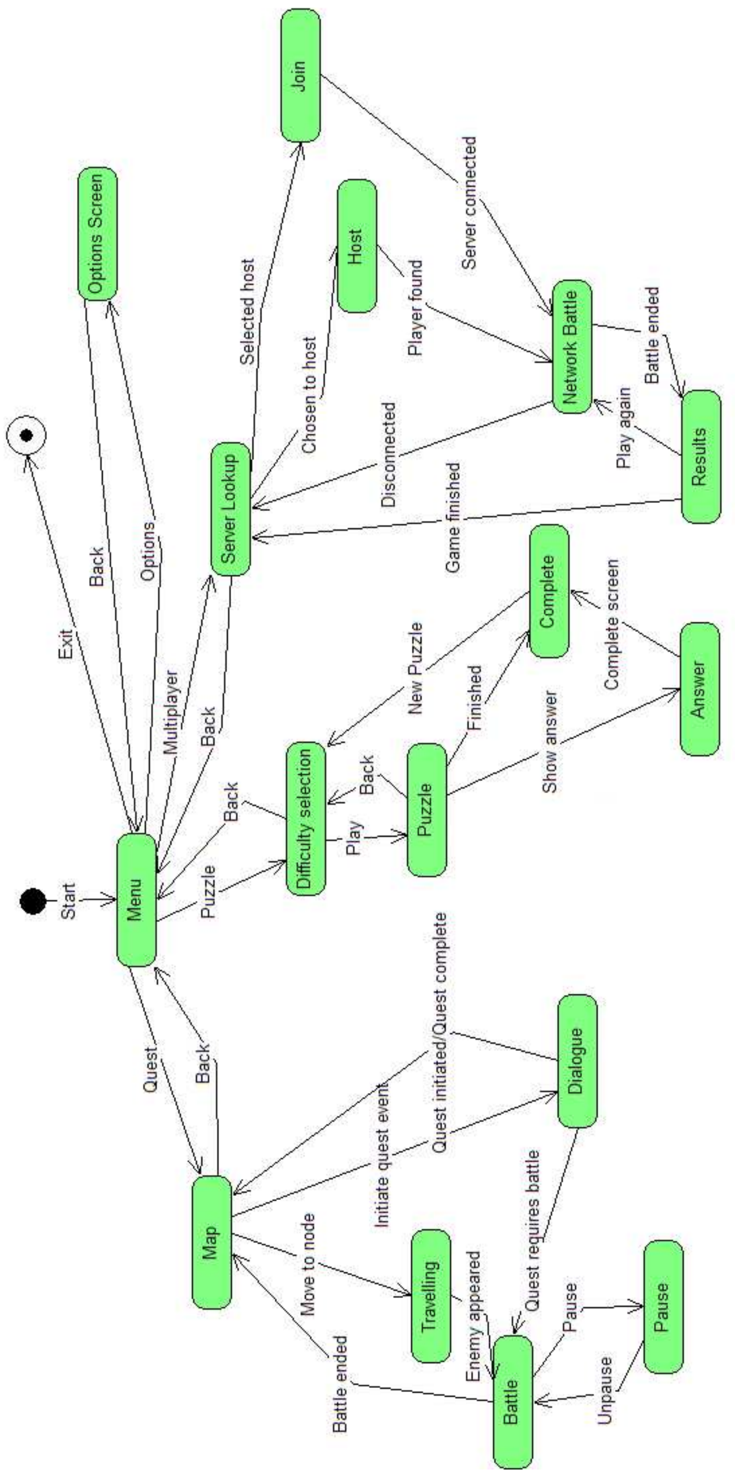


Figure 7.2: UML State diagram of the different game states

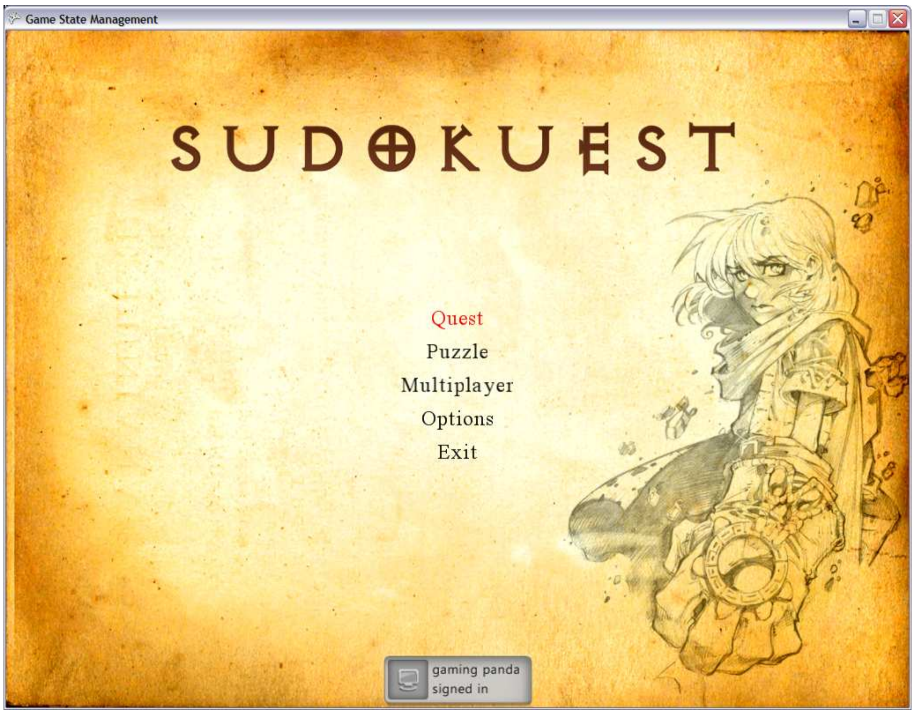


Figure 7.3: A screen shot of the Main Menu

7.2 Puzzle Mode Screen

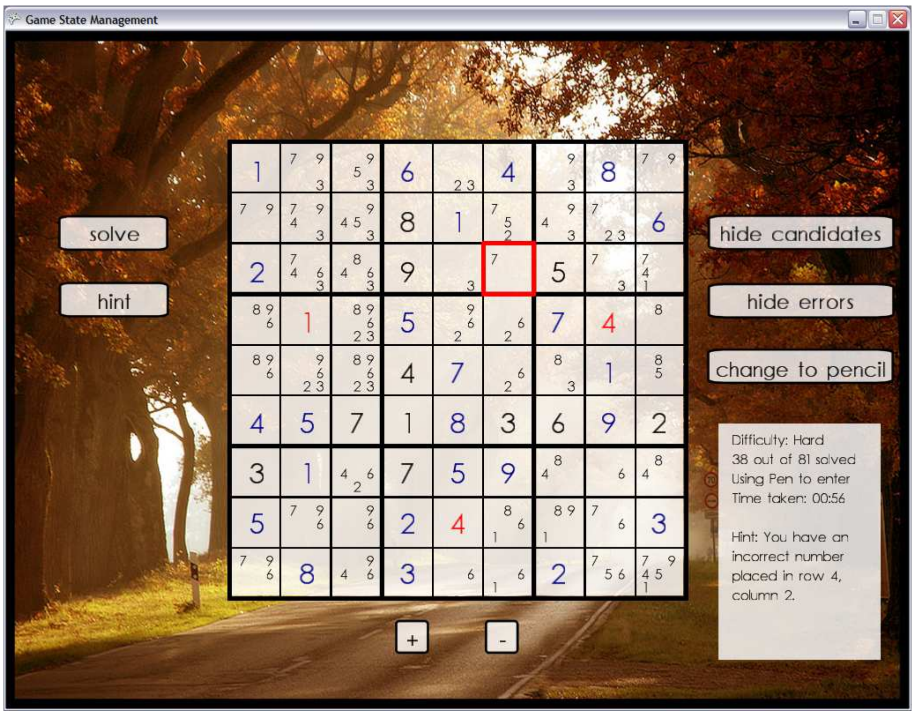
The design of the Puzzle Mode was outlined in section 5.3. The main focus while creating the Puzzle Mode screen was to allow the user to play Sudoku puzzles at a range of difficulties, with a strong emphasis on a 'relaxing' and easy to use GUI, while also offering an extended feature set that the player can use. A screen shot of the interface is shown in Figure 7.4.

7.2.1 Solve and Hint buttons

These are essential features for the user so that they can ask the game to solve or help them in a puzzle. The solve button shows the solution, which has been stored from when the puzzle was created. The hint button uses my solver's `solve_one()` function and tells the user in a message box what number can be placed in what cell, the technique used to solve it and a short reason why. I have then left it to the user to enter in that number that was given from the hint, encouraging to learn the reason behind the placement.

7.2.2 Candidates and Notation

Another essential feature was the ability for the user to make notations on the puzzle. This is universally regarded as an important feature when people solve Sudoku puzzles (on paper or otherwise) and allows the user to write down possible candidates for each cell to help solve problems



Difficulty: Hard
38 out of 81 solved
Using Pen to enter
Time taken: 00:56

Hint: You have an incorrect number placed in row 4, column 2.

Figure 7.4: A screen shot of the Puzzle Mode

of greater difficulty. For my implementation, they can easily switch from a 'Pen' (the ability to write in cells) or a 'Pencil' (the ability to notate in cells), and notations for a cell are automatically removed once they enter in a number with a Pen.

The user can also turn on 'Show Candidates' which automatically shows all the possible candidates for each cell at that moment in time (without taking into account more advanced solving techniques). This is also automatically updated on the grid every time they enter in a new number into a cell. This is performed by calling the `find_candidates_list()` function. This helps the player search for solutions for cells and gives them the option to view what can and can't currently be entered into the grid. This feature can also be turned off by the user.

7.2.3 Show/Hide Errors

A function has been provided to allow the user to show the errors that they have made in the puzzle so far. Once turned on, the program checks the user's answers so far against the solution and if they do not match then the erroneous numbers are highlighted in red for the user to easily identify. The user can also be turned off if they wished. This assists the user in knowing if they have gotten anything wrong in a puzzle and what they have gotten wrong if they have.

7.2.4 Zoom in/out

Two further buttons have been given to allow the user to increase or decrease the size of the puzzle board to how they prefer. This is done by scaling all of the lines and strings drawn in the puzzle up or down.

7.2.5 Random Backgrounds

To add to the ‘relaxing’ atmosphere, specifically calm backgrounds were chosen to be used for the puzzle mode screen. When the user goes into Puzzle Mode the background that shown is selected at random so that it is rarely the same each time the user uses it, adding to the overall interestingness.

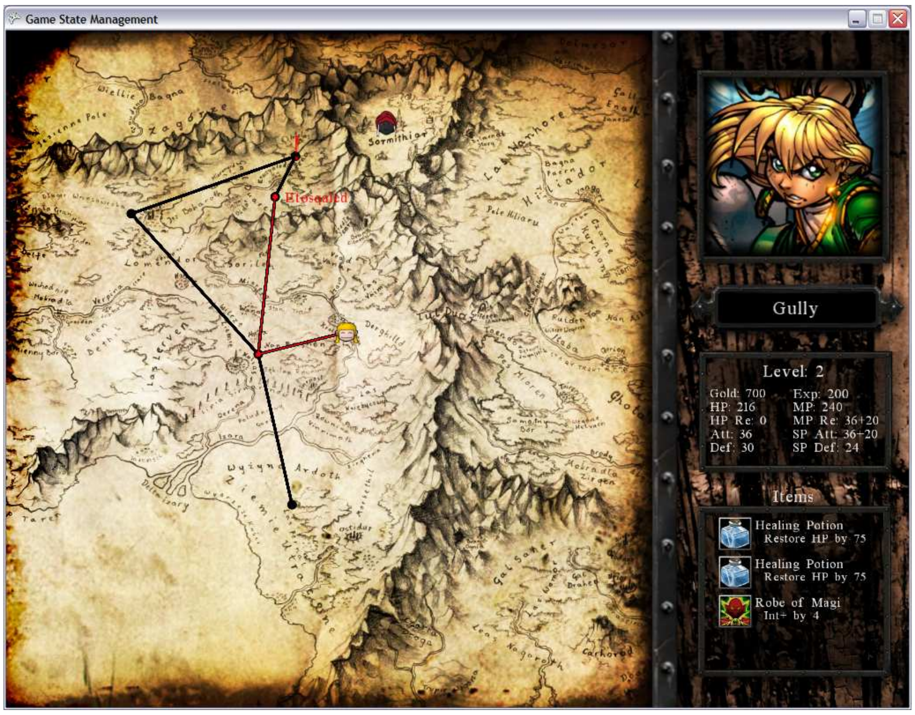
7.2.6 Tutorial Mode

As of writing, the tutorial mode has not yet been implemented. However, I hope to present a screen that shows the rules and techniques of playing and completing a Sudoku puzzle. This would be done through showing the user screens and explanations detailing each step on solving a sample puzzle.

7.3 Quest Mode and the Map Screen

The design for Quest Mode is outlined in section 5.2. The main idea is to offer players a Sudoku experience that incorporates general gaming ideas and concepts. Players embark on a story mode with their customizable character and face off against enemies. The main screen for this is the Map screen which a central hub for the player, and allows the player to perform most of the major functions for this mode. A screen shot of this screen can be seen in Figure 7.5. Many of the other screens in Quest Mode will load up from this screen. An attempt has been made to cut down on wasted resources by often unloading the map screen when a player moves on to different screen, instead of loading each one on top of each other.

Given the time constraints, the length and content of the Quest Mode has been stripped down from what the ordinary player would expect from a commercial product. However, an attempt has been made to include all the features that you might expect in a game. For instance, the number of enemies, items, spells, locations and scenes are limited but have been implemented to show the full range of effect of what kind of experience a player would receive, given a longer production schedule. Also, I have tried to design it so that it is robust and easy to extend any of these areas.



Level: 2

```
Gold: 700     Exp: 200
HP: 216       MP: 240
HP Re: 0      MP Re: 36+20
Att: 36       SP Att: 36+20
Def: 30       SP Def: 24
```

Healing Potion
Restore HP by 75

Healing Potion
Restore HP by 75

Robe of Magi
Int+ by 4

Figure 7.5: A screen shot of the Map Screen in Quest Mode

7.3.1 Player Class

Quest mode starts out with the player creating and customizing their character. A screen shot of this screen can be seen in Figure 7.6. This is so that they can rename their character from the default setting and also distribute the strength of their attributes. This allows the player to increase or decrease their attributes to suit their playing style. For example, a player who is not as good as solving a Sudoku puzzle could add more points to their magic and spell capabilities. Players can distribute points between three parts of their character, their Strength (adds to their HP and Attack), their Agility (adds to their Defence and SP. Defence) and Intelligence (adds to their MP, SP. Attack and MP Regeneration Rate). More details on these attributes can be found in section 5.2.3.

Once created, their information is stored in the Player class. It also stores what items and spells that a player has, and has the appropriate methods for managing them, such as adding an item or removing an item.

The Player class also stores the amount of Experience Points that the player has. Players earn Experience Points by defeating enemies or completing quests and when they have earned enough, then they level up, causing all of their attributes to be multiplied by 1.2. The amount of experience points that a player needs to earn is based on this equation:

$$\texttt{if } (XP > 100 \times 1.75^{Level}) \texttt{ then levelUp()}$$

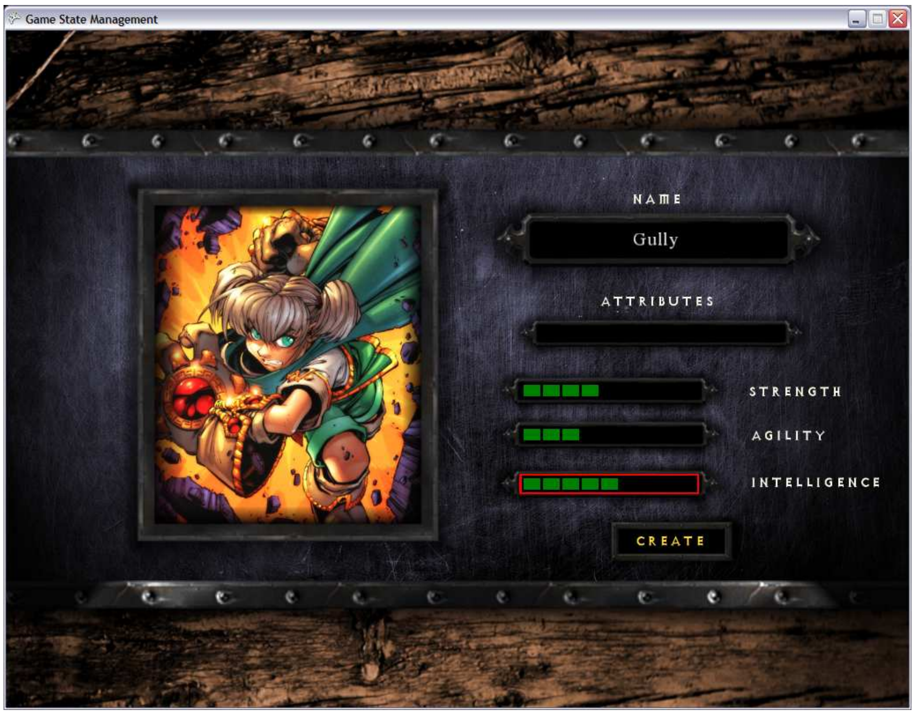


Figure 7.6: A screen shot of the Player Create Screen in Quest Mode

This means that the higher a level a player is, the more points that they need to earn to level up compared to the previous level. This is a common gaming design used in Role Playing games, which offers an early incentive to carry on playing the game (as their character rapidly progresses early in the game) but gives levelling up more significance later on in the game.

The Enemy class is a subclass of the Player class and inherits all of its variables and functions. The Enemy class acts as the place holder for the AI characters in the game, and has additional variables such as how clever it is (how long it takes to solve using a certain technique), and how many often it makes a mistake (by entering in the incorrect number in to a puzzle for example).

Further enemy characters in the game are a class of their own and further extend the Enemy class, so that the program can deal with all enemies equally but they still have the possibility of implementing features of their own. These can be seen in my implementation with the 'Wolf' class (which has simple AI) and the boss enemy 'Rilustiarr' (which works things out faster and makes less mistakes). Further extensibility can be easily added to the program by simply adding further classes for each type of enemy.

7.3.2 Items and Spells

Players hold items and they consist of two types, usable items and non-usable items. All items hold information such as their cost, graphical data and descriptions, and they can also implement two functions, `use()` and `effect()`. All non-usable items are able to implement `effect()` to be able to change the properties of the player that equips it or the player's enemy, and the `use()` function is for items that are usable during a battle, also affecting the player, or their enemy but then disappears once used. This gives me the ability for example to make items that have an active effect on a player but can also be used (and subsequently discarded), which has a different effect the enemy.

These items have been implemented into the game:

- Mithril Hammer: Non-usable, increases Attack attribute.
- Plate Mail: Non-usable, increases Defence attribute
- Robe of Magi: Usable and active, increases SP Attack and MP Regeneration rate. When used it deals a large amount of damage based on SP Attack.
- Ring of Regeneration: Non-usable, increases HP Regeneration rate.
- Energy Booster: Non-usable, increases maximum MP.
- Vitality Booster: Non-usable, increases maximum HP.
- Healing Potion: Usable, heals HP.
- Clarity Potion: Usable, heals MP.

Each player also has a list of spells that they can use, with each spell implementing their own `use()` function that has a special effect on the player or the enemy during a battle. This is different to usable items in that they cost the user MP and is not discarded once used. They can also not be bought or won off of enemies but are instead earned through gaining XP and Levelling up the character.

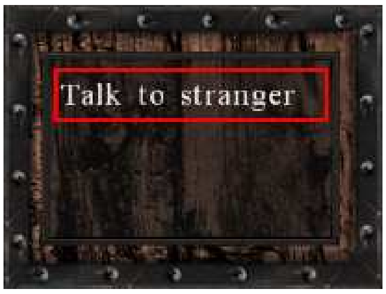
In my finished implementation, each player has the possibility of using three different types of spells that have been implemented:

- Level 1: Laguna Blade - Deals damage to the opponent.
- Level 2: Frost Bite - Freezes the opponent for a short time so that they can't move.
- Level 3: Purify - Heals the player a small amount.

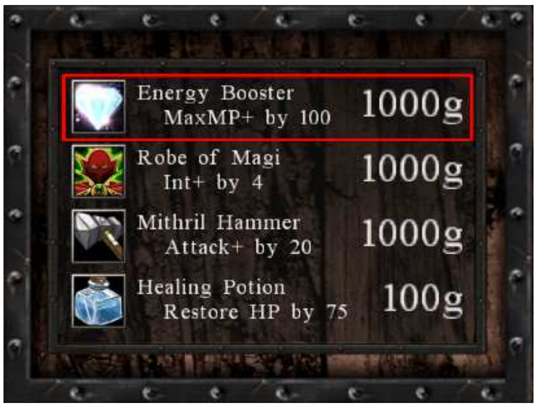
With my implementation, I attempted to make it easy to add further items and spells, with the ability to do anything they want to the player or enemy's attributes, allowing for a large range of different types of effects and usages to be implemented. This freedom will hopefully allow for unique and interesting items or spells that can be obtained.

7.3.3 Nodes, Cities, Towns, Dungeons and Roads

In the map, there are various nodes that the player can travel to and from. There are three types of nodes, a City, a Town or a Dungeon (as described in section 5.2.2), and these nodes are connected through roads. This is implemented by using an abstract Node class that the City, Town and



(a) A screen shot of a Quest List Menu



(b) A screen shot of a Shop Menu

Dungeon classes extend, allowing for each of the classes to hold their own specific information and perform specific functions related to their type.

The City class specifically allows players to shop for items, with the ability to make each City sell specific and unique items. The Dungeon class on the other hand allows the player to initiate in a fight against an enemy.

The roads in between nodes may also contain an enemy. If the player travels in between these nodes and runs into a enemy, it detects the collision and initiates a fight against them. This leads to the Battle Screen (the implementation of which is detailed in 7.4).

7.3.4 Item and Quest menu

Nodes of special interest may have a menu available to them so that specific actions can be performed. This can be the case for taking a quest at certain nodes, buying or selling items at a City or fighting an enemy at a Dungeon. This is implemented on the Map screen and when a node has been selected, the screen enters a menu state, so that input is directed to the menu. Figure 7.7a shows one such menu with one available quest.

Each City node has a list of items that it sells, and when selected, the menu displays those items. The user can then buy those items (given that they have enough gold to do so and space in their inventory), or they can sell items to the shop.Figure 7.7b shows one a screenshot of a shop menu.

7.3.5 Scene screen

To carry on the story of the quest, there are scene screens that have the player's character talking to another character in the story. This is implemented by having a Scene super class that holds many of the common functions and variables that each scene holds. The Scene class also allows each subclass to add any additional functions that maybe needed.

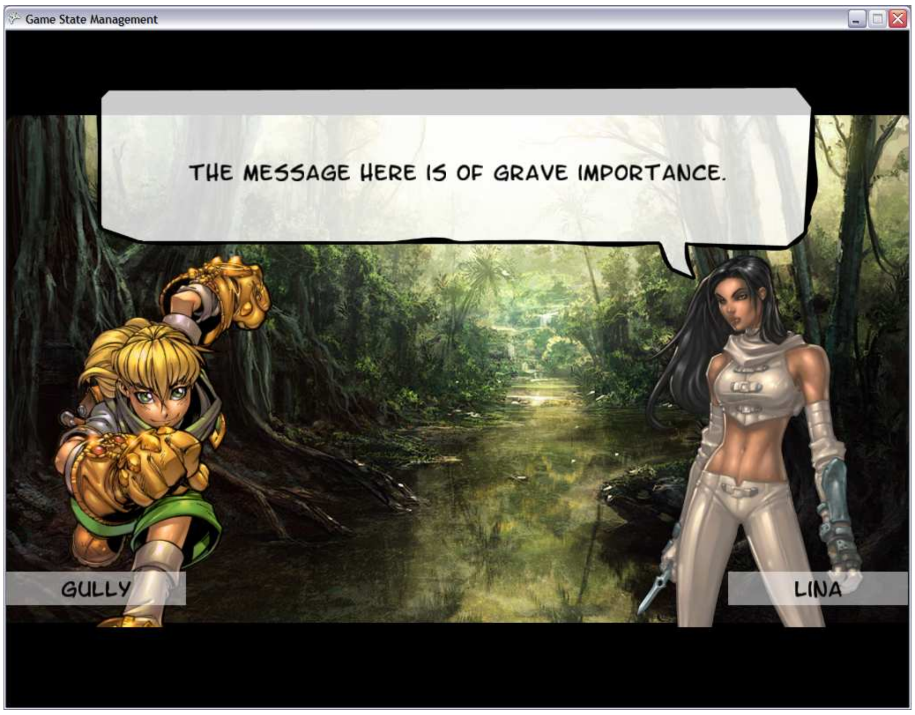


Figure 7.7: A screen shot of Scene 2

Each subclass that extends the Scene class holds the script of the scene and information such as what characters are there. The Scene class then uses this information to print onto the screen. Figure 7.7 shows a screen shot of scene 2.

7.4 Battle Screen

The Battle Screen is one of the most important parts of Quest Mode and the Multi-player Mode, and allows for the player to compete against an opponent on a single Sudoku grid. If someone enters in an incorrect number into the grid then they incur a penalty; their player will not be able to move or enter in any numbers for a short period of time (however they will be able to cast spells or use items). The opponent against the player could either be an AI in Quest Mode or a human in Multi-player Mode. A screen shot of the Battle Screen can be seen in Figure 7.8

7.4.1 AI

The AI used for the Battle Screen is predominately taken from the `solve_one()` function. This is combined with the intelligence of the AI enemy, to work out how quickly the AI enters a number into the grid. This is implemented by first calling the `solve_one()` function, working out what number to place and in what cell and then effectively feinting dumbness. The AI takes the difficulty of its solve (given by the `solve_one()` function) and adding a weighted amount of time to it (what time added is dependant of the ‘intelligence’ of an enemy). Once that time has passed, the enemy

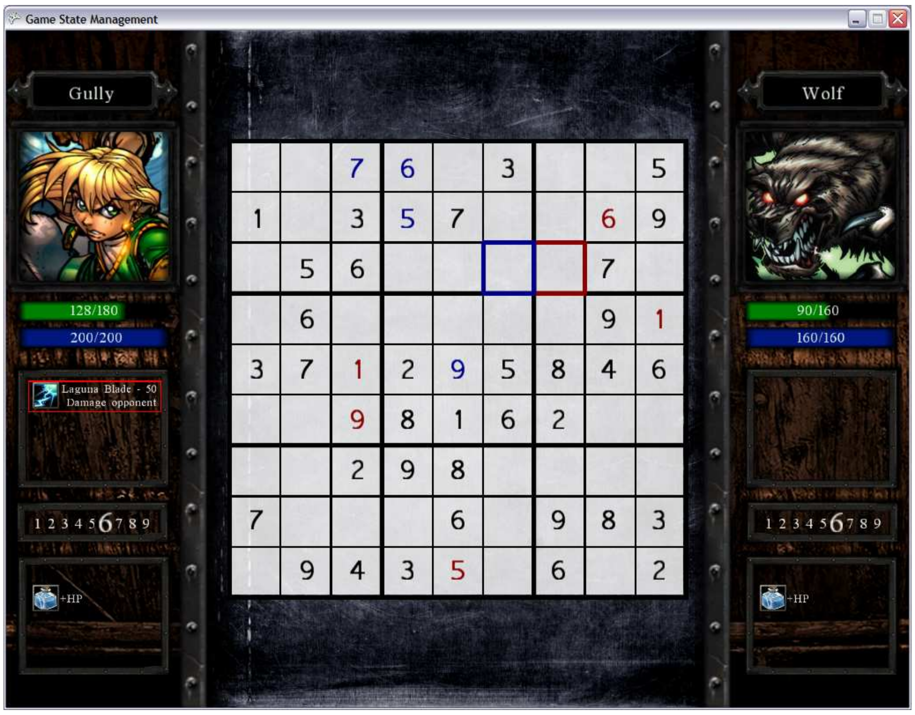


Figure 7.8: A screen shot of the Battle Screen

will then move to that location at the simulated speed of a human and then enter a number. There is also a small chance that the AI will enter in an incorrect number, incurring a penalty. The chance of this is also determined by the specific AI's characteristics.

7.4.2 Determining a winner

Each time that a player enters a correct number into the grid, then their opponent is attacked. This is based on the player's attack attribute, the opponent's defence attribute, a certain amount of randomness and whether that number completes a column, row or sub-grid (adding bonus damage). The value of the damage that is caused is given from a simple calculation://

$$\texttt{Damage = (EnemyAttack} \div \texttt{(Defence} \times 0.1)) \times ((\texttt{rand.NextDouble()} \div 4) + 1)$$

Each time a player is attacked, then the damage caused is subtracted from their HP and they lose if they have lost all of their HP.

7.4.3 Single screen multi-player

This allows two human players to compete against each other on the same screen on the same platform. This is implemented much like the normal Battle screen, without the AI and separate

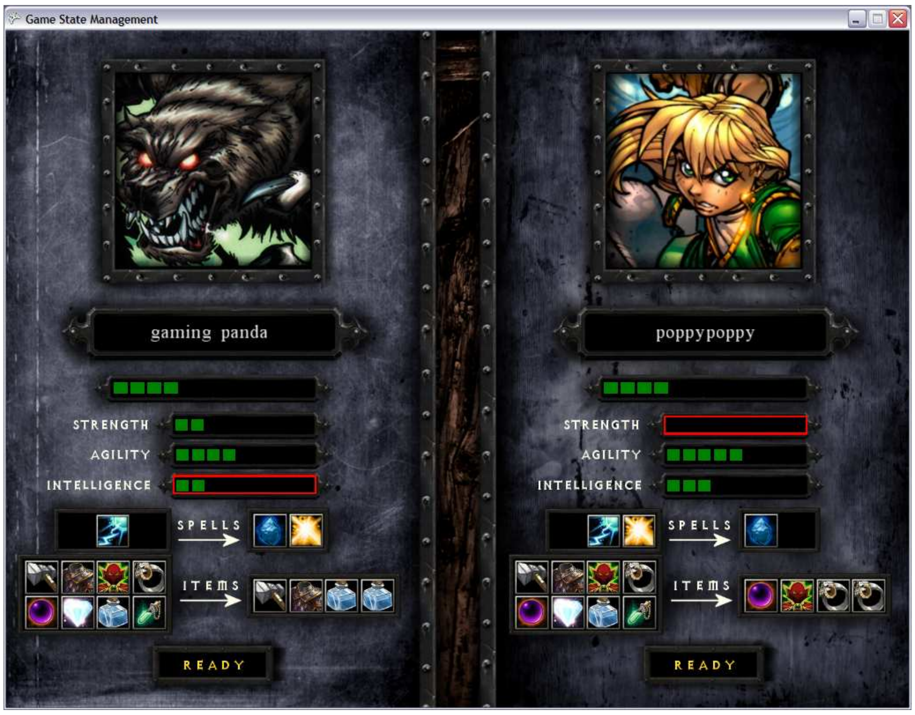


Figure 7.9: A screen shot of the Multiplayer create screen

controls for a second human player.

This mode also has a player create screen, allowing for the players to pick their own attributes, spells, items and avatar before they compete against each other. This offers another level of strategy where each player can select items, spells or attributes that maybe more effective than their opponent's choices. A screen shot of the multi-player create screen can be seen in Figure 7.9.

7.5 Network Mode

The network mode allows players to battle each other through a LAN connection on a PC or an Xbox 360. To implement the networking aspects of the project, the Network Game State Management sample[28] from the XNA Creators Club web-site was used as a basis for hosting and connecting clients to servers.

For the networking, I integrated the Microsoft Games for Windows LIVE and Xbox LIVE system into my project by using the GamerServices library provided with XNA. This allows the user to use a gamer profile that they may have already set up through the LIVE systems, and can be used with any other LIVE game. This integration also allows the user to create a new profile through the game and allows other features such as voice chat between clients to be used.

Once logged on the the LIVE system, the user can then choose between creating or finding a game. Players that create a game will the go into their Lobby, and they will then broadcast their game over the network, while waiting for other players to join them. Users who decide to find a game to join will search the network for all available hosts. They will then be offered a selection of games available to choose from and their latency to that host. Once players have joined, they enter into the Lobby with the host. Once they are connected to each other, voice chat is enabled for them to be able to talk to each other while playing.

Throughout the whole networking process, the NetworkSessionComponent class deals with network problems, such as a player leaving or connection losses. In these instances, the players are notified and taken back to the main menu screen.

7.5.1 Network Player Create Screen

Similar to the single screen multi-player create screen, this allows the players to select attributes, spells and items to be used in the battle. The player names are taken from the player's gamertag that is associated with their LIVE profile. As this is a simple menu screen, each node in the network only sends updates of itself and only receives updates of their opponent, as opposed to the server receiving all inputs from clients, updating the game state and then sending the game state to all clients for them to update themselves. This is because there is no interactivity between the server and client at this stage so clashes with input does not need to be dealt with. Each packet sent between the host and client has the following information:

- Sender's selection entry (four byte signed integer).
- Sender's avatar selection number (four byte signed integer).
- Sender's strength attribute (four byte signed integer).
- Sender's agility attribute (four byte signed integer).
- Sender's intelligence attribute (four byte signed integer).
- Sender's spell selection (four byte signed integer).
- Sender's item selection (four byte signed integer).

Once a packet has been received, they update the sender's half of the screen. Data is only sent to each other when there is a change between the players, so that network issues are reduced. Once players are both ready, they both move onto the Networked Battle Screen. A screen shot of the multi-player create screen can be seen in Figure 7.9.

7.5.2 Network Battle Screen

Also similar to the single screen multi-player battle screen, it starts with the server creating a Sudoku puzzle and then sending the puzzle to the client. Once the client receives the puzzle, the battle then starts. The data flow between the host and client can be seen in Figure 7.10. The server continuously looks to see if there is an incoming packet, and in the case that there is or if the server's own input has changed, then the server and game state is updated and the update of the entire game is sent to the client. The client also continuously looks to see if there are incoming

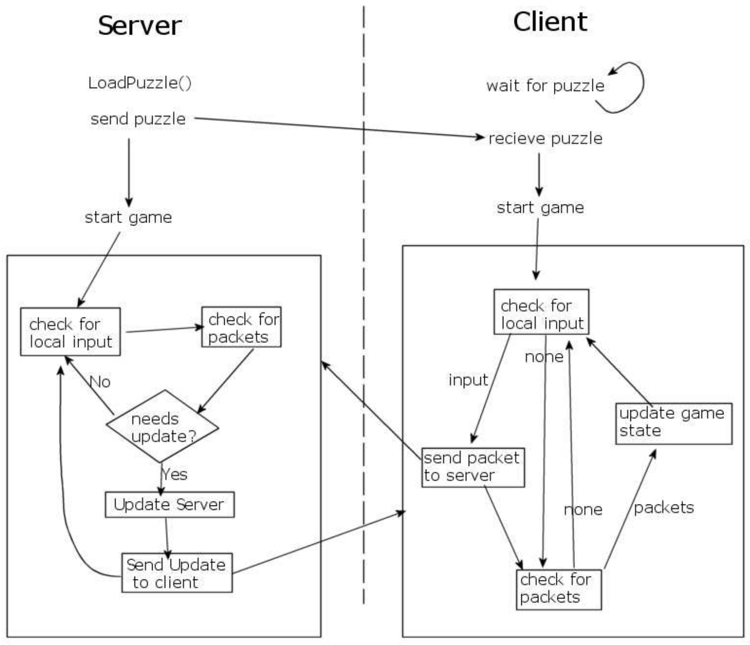


Figure 7.10: The data flow of the Client-Server network during the Battle Screen.

packets from the server, and updates the game once there is. If the input changes on the client side, the change in input is sent the server and only updated once the server responds to the update. This helps reduce network and input clashes between the server and client as both inputs are treated the same. Again, both the client and server only sends updates if there is a change in input, to reduce network issues.

A packet sent to the server from the client holds the following information:

- Client's cell selection entry (four byte signed integer).
- Client's number entry (zero if none)(four byte signed integer).
- Client's spell/item selection(four byte signed integer).
- Client's use of spell/item (boolean).

The server's packet sent to the client holds the above information for each player as well as the additional information:

- Server's current HP (four byte signed integer).
- Server's current MP (four byte signed integer).
- Client's current HP (four byte signed integer).
- Client's current MP (four byte signed integer).

Once each side detects a winner, the game is finished and they return to the Lobby screen where they can decide if they want to play another game or exit. A screen shot of the Battle Screen can be seen in Figure 7.8

7.6 Graphics

As mentioned in the design (section 5.2), a heavy amount of graphical resources used in the program have been taken from the comic 'Battle Chasers'[24]. There has also been a large amount of images used that were taken from other sources such as textures and art from the web-site www.deviantART.com and www.flickr.com. All images have been heavily modified to suit the purposes within the game.

To draw the images and text on the screen, Microsoft XNA's spritebatch libraries have been used. However, as XNA does not immediately offer a technique to draw lines, Michael Anderson's Line library[29] has been used to draw the lines used in the game.

7.7 Input

Throughout the game, the user can control the game by using a keyboard or a Microsoft Xbox 360 game pad on both the PC or Xbox 360. This is handled through the HandleInput class and causes the program to deal with a keyboard input equivalently to how it deals with a game pad. This takes key and button presses and assigns them to commands, taking the specific input away so that the program does not need to deal with what specific buttons or keys have been pressed. This allows easy functionality to be programmed for both input types.

Even though the game pad does not have a number pad like a keyboard does, extra measures have been placed to allow for number input from a keyboard. This is a vitally important feature for users who play using a keyboard as it allows for much easier input. To compensate for the lack of buttons on the game pad, both inputs can also place numbers by cycling through numbers.

At time of writing, mouse input for Windows has not yet been implemented, but I hope to add this to my program so that users can select buttons or cells in a grid by clicking on the associated item. This will further add to the available options of input and the interactivity between the program and the user.

7.8 Save/Load

At time of writing, saving and loading puzzles, and game state for Quest Mode has not yet been implemented, but I hope to present a feature that will allow the user to save and load so that they can easily come back to a puzzle or save their character and items earned in Quest Mode. This would be implemented by using XNA's save game state libraries that use XML to store information.

7.9 Sounds

At time of writing, sound effects and music has not yet been implemented, but I hope to present this feature, which will add additional enhancements to the game. This would be done by using the XNA sound libraries that allows WAV sound files to be played at parts of the game.

Chapter 8

Testing

This chapter outlines the testing stage of the project. Various qualitative and quantitative tests are performed to measure the quality of the software, in preparation for evaluating the product.

8.1 Sudoku Implementation

During implementing the main Sudoku aspects of my project, the solving, generating and grading of a Sudoku puzzle, there was an attempt made to test how well the a technique worked and if it can be improved upon. These tests are detailed throughout section 6.

Below outlines further testing of how well my program can solve, grade and generate puzzles compared to other software.

8.1.1 DLX Solver

The first test I ran for my solver was to test it against the Sudoku Top 1465[30], a well known list of exceptionally difficult Sudoku puzzles. My solver managed to solve all 1465 puzzles completely and correctly. This was expected as it is a brute force solver and does not use Sudoku logic to solve them. This shows that my solver is robust and can handle even the most difficult Sudoku puzzles. This test was run on Pentium 4 3.0GHz processor with 1.5GB of RAM and took 1.21 seconds to solve all 1465 puzzles.

As can be seen from the results, my solution ran very fast, with an average of 0.82ms per puzzle. This was also to be expected, as the DLX implementation is a known method for solving Sudoku puzzles very fast. This shows that my implementation is very efficient at solving puzzles and can solve one in a very short period of time.

An attempt to test my result of how long my solver took to solve the Top 1465 puzzles compared to other solvers was sought but I could not find any software that allowed to easily batch test the puzzles. Testing against other software would show how efficient my solution is compared to others. However, this may not be strictly important as it has been shown that it is able to solve the hardest puzzles very quickly.

8.1.2 Grading

To test the proficiency of my grading algorithm, I ran the grader using Sudoku puzzles created by the Times Newspaper. The Times Newspaper is regarded as a well respected newspaper and the first to introduce Sudoku to the British public. Since then they have run a Sudoku puzzle at least once a day and offer four difficulties: Super Fiendish, Fiendish, Difficult and Mild. Testing my grader against these puzzles will hopefully show that it is accurate and in line with the common perception of Sudoku difficulties. The results are shown below:

- Puzzle #1946 (Super Fiendish): rated difficulty 5, extreme.
- Puzzle #1949 (Super Fiendish): rated difficulty 5, extreme.
- Puzzle #1948 (Fiendish): rated difficulty 3, medium.
- Puzzle #1950 (Fiendish): rated difficulty 3, medium.
- Puzzle #1943 (Fiendish): rated difficulty 3, medium.
- Puzzle #1945 (Difficult): rated difficulty 3, medium.
- Puzzle #1942 (Difficult): rated difficulty 2, easy.
- Puzzle #1941 (Mild): rated difficulty 2, easy.
- Puzzle #1944 (Mild): rated difficulty 2, easy. (Rated difficulty 1 when not taking width into account)
- Puzzle #1947 (Mild): rated difficulty 2, easy. (Rated difficulty 1 when not taking width into account)

The results show that the grader is generally in line with the Times Newspaper's opinion. Their hardest difficulty is similarly graded, as are their mild puzzles. There is a gap between my difficulty 4 grade that does not seem to match directly with one of their grades. This is not necessarily a bad thing, as it shows that my difficulty scheme is spread out between grades in the harder end, offering a more detailed range of difficulties. Their easiest difficulty does not line up with my easiest difficulty, however, this only means that I offer a broader range of puzzles, which is expected as they have four difficulty levels and I offer five.

The puzzles used can be seen in Appendix C.

8.1.3 Generating

A further test was carried out, to see how well my generator is at creating puzzles. Three puzzles were generated at each of the difficulty ranges that my generator offers. These were then timed to see how long it took for three people in their mid-twenties to solve each one and then compared to how long those three people took to solve the Times Newspaper puzzles. To ensure fairness and to reduce testing time, each person knew the rules of Sudoku and had said that they played on a regular basis. They also played all of the puzzles using a pen and paper only, as this was to test how well the software generates puzzles, not how well it presents them. The results are shown below:

- Difficulty 5 puzzle average: 44 minutes 19 seconds.
- Difficulty 4 puzzle average: 34 minutes 47 seconds.
- Difficulty 3 puzzle average: 22 minutes 20 seconds.
- Difficulty 2 puzzle average: 7 minutes 34 seconds.
- Difficulty 1 puzzle average: 3 minutes 37 seconds.

- Super Fiendish puzzle average: 47 minutes 24 seconds.
- Fiendish puzzle average: 26 minutes 04 seconds.
- Difficult puzzle average: 12 minutes 47 seconds.
- Mild puzzle average: 6 minutes 12 seconds.

Although using time is a crude and rough test, it still shows a basic measure of the difficulty of a puzzle. The data shown here, shows that the times it took to solve the puzzles that my software generated were on par with the times that it took for those same people to solve the Times puzzles used. However, there may have been many other factors involved, such as a spike in tiredness for certain individuals and people using their first puzzle to get into their 'zone' for solving.

8.2 Focus Group

To test how well the game played and how easy the GUI was to use, three different testing sessions were carried out. Sessions were carried out in groups of three people, and an attempt was made to have each person in the group at a different Sudoku skill level; one would be an expert (someone that played Sudoku often and could confidently solve the average puzzle), one would be a intermediate player (someone that would only occasionally solve puzzles), and one would be a beginner (someone that knew the rules but had little to no real experience playing Sudoku). Each session lasted approximately thirty minutes and included all aspects of the game running and networked on both the PC and Xbox 360 console. In each session, we tried to gauge what each person thought about the presentation, each mode, how it ranked against competing software, and how much they would pay for the game.

The first session was with three people from the Gaming Society at Imperial College. This was an important typical gaming demographic of 18-25 year old males. They are considered the slightly older and more experienced type of gamer, having played a wide range of games. The second group were three GCSE students from BETHS Grammar School. This is the newer generation of gamers, that are less experienced than the University students and have a keener interest in action orientated games. Lastly, the third group selected were parents of those students, who have had little experience with games. From the sessions, it was found that:

- Each person thought that the presentation and graphics of the game were of a high quality. Two people mentioned that the interface was well designed and felt very intuitive to them. The parents felt that some sections of it were slightly more complicated such as the Quest mode but they quickly felt comfortable with the puzzle mode.
- It was generally regarded with the University students that although the story was a bit silly, it was not the most important part of the game and it served its purpose. The younger audience felt that the story was funny and they mentioned that it helped keep their interest. The parents did not care much for the story.
- Everyone except the parents were highly impressed that it ran on an Xbox 360, and said how they were more interested in playing it on there instead of on the PC, despite the limited controls. The GCSE students were especially excited to see it on the Xbox.
- The University students mentioned the lack of sound dulled the experience slightly, one person saying how without sound, they were given less feedback on what they were doing.
- Although a few of the students did not care much for the puzzle mode, they were impressed with how many features it had that they did not expect a Sudoku game to have. All of the intermediate players said that they would consider using the puzzle mode as their main means to play Sudoku. Many of the parents felt that it was the part that interested them the most.
- One of the expert University students found that the Quest mode was of less interest to him, and he stated that this was due to his lack of enjoyment for Role Playing Games. All of the other students found that the inclusion of Quest mode was highly interesting to them, and played it till then end, stating that they would like a longer game. The GCSE students were especially interested in the role playing elements such as the levelling, spells and items.
- All of them found that the battle mode was a unique and fun way to play Sudoku. They also mentioned that the multi-player aspects would probably keep them interested for a lot longer. The parents were not as interested in it as the students. The novice students felt that even though they did not know a great deal about Sudoku, they still enjoyed the battle modes and were keen to learn more.
- All of the student regarded the Network mode as a key feature and were happy with the LIVE integration. They also praised the support for voice chat during sessions.
- They all found that the lack of a mouse for the PC was bothersome at first but for most of the students it quickly became not too much of an issue for them. The parents struggled with selecting a cell using the keyboard but did not have trouble entering in a number.
- No one mentioned any problems with the difficulty settings, and two of the older beginner players praised that there was a setting below easy, with one stating that the intimidation of harder puzzles was a key factor in what was keeping her from playing more Sudoku.
- Only the expert parent mentioned that Sudoku puzzles were suppose to be symmetrical, and stated that because the puzzles generated were not, they were less enjoyable. One University student also mentioned that they wish it had variants of Sudoku.

- Of the novice players, only the GCSE student still did not want to learn more about Sudoku after the session. However, all the novice players felt that a lack of a tutorial mode was a key factor for them.
- Over competing software, many praised that the my game offered random puzzles, which most other commercial products that they had experienced did not have. They all generally praised that there were more features than other software. The parents however stated that although the puzzle mode was simple to use, other products had a better interface that they felt appealed to them more. Many people also mentioned that they enjoyed using the hand recognition touch screen input from the Nintendo DS game 'Brain Training' more than the input options offered from my game. However, a few people also said that because keyboard and number input was allowed, it was far better than the many software that did not allow this. Many of the students said that they would prefer to have this game than any other Sudoku game because of the Quest and Battle elements, and also because it can run on their Xbox 360.
- The amount that they would be willing to pay for the game ranged from £5 to £15. People on the lower end of the scale stated that the Quest was too short and that if it were longer, then they would be willing to pay far more. People on the higher end said that because it generated Sudoku puzzles, they felt that they would be willing to pay more.
- When asked for them to rank the game out of 10 akin to how a gaming publication might view the game, the average was 6.7. The GCSE students said that it was not as interesting as the commercial action games available, and the University students felt that a longer Quest would have raised their score significantly. The parents were the most generous when giving their scores.

8.3 Network Efficiency

We then tested the efficiency of the game during the network mode. To do this, we simulated a busy network by having four computers connected to a single router. Two of the computers would then transfer large amounts of data continuously back and forth to each other, while the other two computers tried to connect and play a game against each other.

We found that while there was some lag during the session, it was not significant enough for the flow of the game to be broken up or interrupted. The game on the client side would occasionally pause between inputs. On one extreme case, the connection failed and error messages on both sides appeared.

Overall, it seemed that there were not many network issues. No packets seemed to have been lost and the game state was consistant on both computers throughout.

8.4 Monkey Testing

The following test was to see how well the software handled with input errors. With a piece of software that has many different aspects to it, it generally means that it is more open to errors as it requires more checking of the program. To ensure that the program worked under various erroneous input commands, Monkey Testing was performed.

We took the software and tested it by entering in many different combinations of input via both the keyboard and Xbox control pad in a nonsensical manner. to try to find if the system would give an incorrect output. In most of the cases, it was found that the user was not able to input an incorrect command and only took in sensible input.

Chapter 9

Evaluation

This section evaluates the software and test results carried out in the previous chapter.

9.1 Evaluation of SudoKuest

The main objectives of the project was to create a piece of software that offered a new an exciting take on a classic puzzle game, to offer an extended and deep experience and a rich feature set to solve traditional puzzles. Through SudoKuest, we have managed to meet many of these specifications.

9.1.1 Random Sudoku puzzle generation

SudoKuest offers fully random Sudoku puzzle generation at a wide range of difficulties. The puzzles generated were found to be accurate in their difficulty grading and also offered a wider range of difficulty than what was expected against the Times Newspaper puzzles. The grading also took into account the width of a puzzle, an important aspect for easy puzzles that other graders often disregard.

However, although the puzzles are generated very quickly, almost always instantaneous, without any noticeable delay, puzzles that are graded as Extreme may take a while longer. These puzzles that are generated do often take a longer time to load, reaching as long as 2 seconds to appear. Although this is a small delay and does not reduce the quality of the experience, it is not desirable to have to wait when playing the game. To over come this, it was intended to use the save feature to also store buffered puzzles that have been calculated and to load those puzzles when required, similar to the way Zendoku works (detailed in section 2.3.5). This would result in no loading times at all. However, as of writing this has not yet been implemented.

The puzzles that are generated are also not symmetrical, a feature that some would view as key for Sudoku. One tester indeed did mention that it was missing during the testing session. However, the majority did not know that they were suppose to be symmetrical in the first place, and so it was not seen to be a large detriment to their experience with the game.

Although it was initially explored and set as a desired requirement, the game does not offer variants of Sudoku to be played. A few of the testers did mention that they would have liked to see other types in the game and so this highlights a key possible extension to the software. Many of the testers however, were more interested in the other key features that SudoKuest offered, and spoke far more of their desire for extended Quests than variants.

9.1.2 Feature set for Puzzle Mode

Our testers all mentioned the rich feature set during Puzzle Mode. Many stated that the features that were included, they did not expect to find and have not been included in many of the other Sudoku software that they had experienced.

The ability to solve a puzzle, show errors, and hints were universally regarded as features that they felt were important in a Sudoku game. We also found that the ability to easily notate and show all the candidates for a puzzle and have them automatically update as the player enters in numbers was well liked by many of the testers. The ability to zoom in and out of a puzzle so that they can display it how they liked was also a feature that many liked, especially when they played on a PC.

The interface for puzzle mode was especially praised by the testers. They found that it was very calming and relaxing, and that it suited Sudoku very well.

However, many thought that they should be able to save and load puzzles, so that they can come back to them at a later date. As this has not yet been implemented as of writing, it would be an important feature to add for a further iteration of the project.

The essential requirement of having ability to also enter in their own Sudoku puzzle so that it could be solved was not implemented. However, with talking with the testers, they did not rate this feature highly. It also missed the desired requirement of being able to publish and print Sudoku puzzles. Although a few of the testers thought that this feature would have been nice, they didn't really feel that the lack of this feature hurt the end product.

Although there was no mouse input, it was regarded that the ability to enter in numbers with the keyboard was a key feature, one that was lacking from many other software available.

9.1.3 Quest and Battle and Multi-player Modes

Quest mode and the added RPG elements to the game, such as the story, items, spells and levelling offers a more traditional gaming experience, whilst still including Sudoku solving as its key ingredient. Many of the testers thought that the Quest and battle modes were very exciting and something that they would want to explore more of.

Whilst the story and length of the Quest mode was very short, everything was there as a basis that could be extended in future iterations. Many key features such as spells and items offered an interesting experience during the battle sections, adding further depth and different strategy to how the gamer plays. Many felt that with a Quest mode that is extended to a typical length of a video game, the game overall would be of a standard as high as many other commercial puzzle and downloadable games available on the market. This is backed by the 6.7 average rating the testers gave the game, and against the other games on the market, is a highly respectable score.

The multi-player and network modes offer a unique and interesting Sudoku experience for two players to compete against each other on the same Sudoku grid. This offers a new experience and a different layer of interactivity between the two players. While the other multi-player Sudoku games available only offered two players to compete against each other on separate grids or in a turn based mode, SudoKuest offers a sense of special urgency between players. Many times during testing,a scenario happened where the two gamers tried to scramble across the grid as quick as they could to reach the same cell to place a number, and combined with freezing spells sent across by each player, it offered a strong sense of competitiveness that is not usually associated with Sudoku or experienced in other Sudoku games.

Also having a network mode, with voice chat support, is a feature that not all commercial games tend to implement, despite it being highly regarded by the gaming industry and its audience. While the game mechanics does not complicate the implementation of the networking, it still manages to achieve this, when very few other games do.

However, the lack of sound is a key feature to any video game, and the lack of sound and music is a detriment to SudoKuest. As time of writing, it has not yet been implemented but is highlighted as a vital extension to the project.

9.1.4 Nielsen's 10 Usability Heuristics

Against the ten heuristics outlined by Nielsen, SudoKuest completes most of the requirements for each heuristic. The interface is generally clear and concise, and displays text, prompts and messages clearly to the user. However, the are various aspects that it has not met. It does not support undo and redo options for the user, it does not offer any real sense of customization for the user so that they can tailor frequent actions and it only offers documentation in a limited user manual form.

9.1.5 Compatibility with the Xbox 360

Although it was only regarded as an extension in the design of the project, many of the testers were very excited to see the game run on the Xbox 360 and stated that they were much more interested in the game because of this. Many of them stated that they preferred playing it on the Xbox 360 despite the limited controls, as this was where they tended to play their games instead of the PC.

Chapter 10

Conclusion

10.1 Key Points

The project has managed to achieve many of its main goals set in the specification, as well as implementing what turned out to be an important extension of the project with the ability to play the software on an Xbox 360 console.

What this project tried to do that was new in the over saturated Sudoku software market was to offer a competitive and interactive multi-player experience. The software allows players to compete against each other on the same PC or console or across a network using either of the two platforms, on a single grid in real-time. Although, this way of playing Sudoku had not been tested in known previous Sudoku software, it became an important and popular feature.

This project has demonstrated that while the market is over saturated on Sudoku software, with each gaming platform holding a range of 3 to 10 Sudoku based games available, very few of the games offer as much quantity of content and quality of content that is usually expected from other games on the same platform. This project offers an experience that extends the Sudoku puzzle into what should be expected in the gaming industry.

The final deliverable of the project has resulted in a game that works on multiple platforms and is in a state where it can nearly be released for public use.

10.2 Further Work

Throughout the programming of the game, many aspects were left open for further expansion. Large aspects of the Quest mode such as the story, scenes, items, spells and enemies could easily be added into the game. As the Quest is short and a longer Quest was highly requested by the testers, a key point for extension would be to add more in this area.

Other key features such as being able to save and load puzzles would be ideal. This would also reduce loading times so that a buffer of puzzles could be used to load puzzles instead of having to wait for the puzzles to be generated when requested.

Features for the Sudoku implementation could be included in the next iteration, with aspects such as symmetry for puzzles, or any variants, such as Nonomino, Hyper, Killer or Diagonal Sudoku. DLX was also chosen with having variants in mind as it could easily be extended so that the extra constraints would be added to the binary matrix. However, a more detailed human solver would be required. Different size puzzles could also be included, with the ability to change the symbols used, so that it is not just limited to numbers. Much of the code implementation was created so that different sized puzzles could be implemented, so a further extension to certain areas would make this easily possible. All of these aspects would further the feature set for the game and extend its longevity.

Other features could be included that were initially mentioned in the specification such as allowing the player to enter in their own Sudoku for the program to solve and offer hints, a tutorial section, multiple profiles, and the ability to print and publish puzzles. Many of these elements could be added on top of the software with little change to the current algorithms.

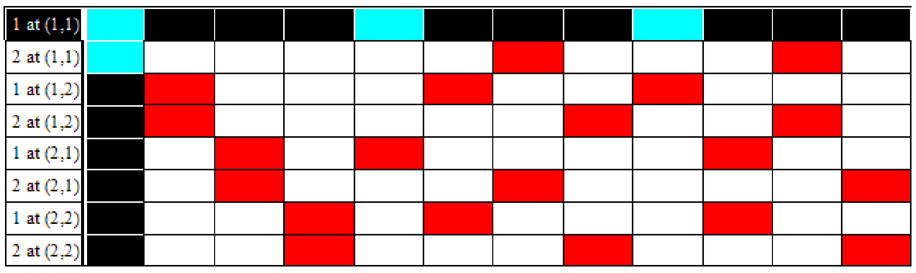
Sound and music would also be a key factor that would need to be implemented in the next iteration, so that a more interactive multimedia experience could be offered to the user. Mouse input would also be another feature that could be added.

In conclusion, with a little effort and sweat, a new experience can be added to Sudoku to turn it into a product that would be viable for a consumer purchase. Although both Sudoku and gaming are popular, it is possible to combine the two industries to create a new and interesting product for the market.

Appendix A

Example of Algorithm X applied to a 2×2 Latin Square

First an unfulfilled constraint is selected. In this example, we have chosen the first column (this constraint in our example happens to be 'A number is placed in column 1 and row 1'). This is satisfied by the first two rows, by placing either a 1 or a 2 in the first cell. Here we have chosen the first row, placing a 1 in the that cell.



1 at (1,1)	🟦	⬛	⬛	⬛	🟦	⬛	⬛	⬛	🟦	⬛	⬛	⬛
2 at (1,1)	🟦						🟥				🟥	
1 at (1,2)	⬛	🟥				🟥			🟥			
2 at (1,2)	⬛	🟥						🟥			🟥	
1 at (2,1)	⬛		🟥		🟥					🟥		
2 at (2,1)	⬛		🟥				🟥					🟥
1 at (2,2)	⬛			🟥		🟥				🟥		
2 at (2,2)	⬛			🟥				🟥				🟥

Figure A.1: Step 1: Pick unsatisfied constraint and a satisfying row.

Adding this row to the solution set, we can now delete all the rows that conflict with this row. Here we select the constraints that this row fulfills and then delete all other rows that also fulfill those constraitns. Leaving us with the remaining four rows and nine remaining columns.

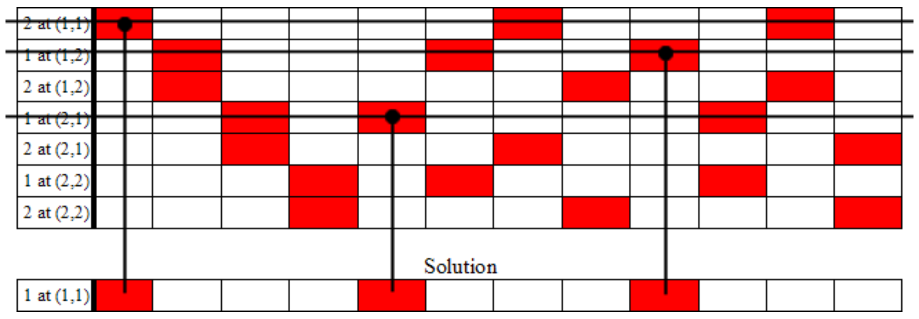


Figure A.2: Step 2: Add chosen row to solution set and delete conflicting rows.

We can then select another unfulfilled constraint, and for this example we have selected another one with two possible rows that fulfill it. We choose the last row to satisfy this constraint (which happens to be ‘A number is placed in column 2 and row 2’).

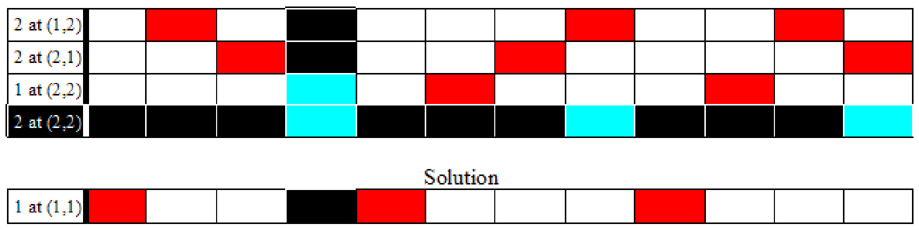


Figure A.3: Step 3: Pick next unsatisfied constraint and a satisfying row.

We can then add that row to the solution set and delete all conflicting rows. These two rows in the solution set has thus far given us a potential solution of:

1	
	2

Table A.1: Our potential half complete solution of a 2 × 2 Latin Square

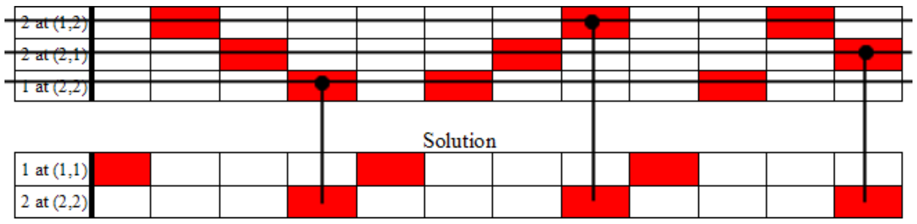


Figure A.4: Step 4: Add row to solution set and delete all conflicting rows.

This now leaves us with just a solution set, with no more rows. However, we still have unfulfilled constraints and so we then have to backtrack to select a different branching path.

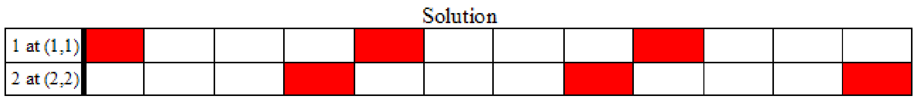


Figure A.5: Step 5: Remaining binary matrix empty and solution set not complete

We now select a different satisfying row.

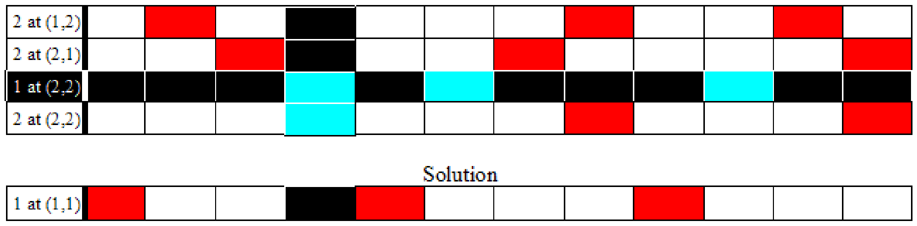


Figure A.6: Step 6: Backtrack and select a different satisfying row.

This row is added to the solution set and the conflicting rows can be deleted (which happens to be the previous branching path we took in step 3).

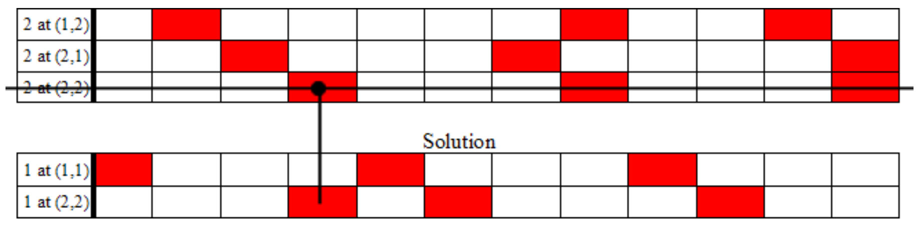


Figure A.7: Step 7: Add row to solution set and delete conflicting rows.

As all constraints are now satisfied, it means that we have a finished solution:

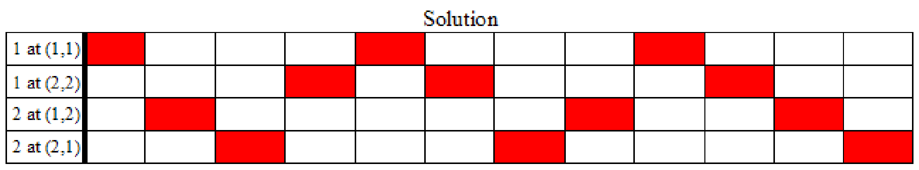


Figure A.8: Step 8: Add remaining rows to solution set as there are no more conflicts

1	2
2	1

Table A.2: Our completed solution of a 2×2 Latin Square

Appendix B

Script used for Quest Mode

Scene 1: Forest 1

Man: Hey you! Little girl, help me... *cough cough*

Gully: Mister, are you OK?

Man: Listen... I don't have much time *kakk*

Gully: Oh my God, you're bleeding badly!

Man: You need to- *kaacck* deliver... don't have time... they're sending wolves...

Gully: Mister! Don't die! I'm too young to be emotionally scarred.

Man: deliver... need to deliver... give this... give this to... Lyehon... Sorce...ress... Lye...hon...

Gully: Lyehon? Sorceress Lina Lyehon of Krolan?? Hang on Mister! Hang on!

Man: .....

Gully: Are you dead?

Man: .....

Message: You receive old man's boots, remaining gold, and some letter.

Scene 2: Forest 2

Gully: Sorceress Lina! I have a message for you.

Lina: Young girl, what is it that you hold for me?

Gully: A letter from a man who just passed away.

Lina: What is your name little girl? In your presence, you seem perturbed.

Gully: My name is Gully maam, and I was just attacked by wolves, who tried to stop me from giving you that letter.

Lina: Gully is a dumb name, I'll shall call you Cuddlynit..... The message here is of grave importance. This land is in great danger, I must set off immediately if I were to stand a chance.

Gully: What did the message say?

Lina: Ril'ust-iarr has risen from the depths of Sormithiar. The seals that kept him a slumber have been purged. Cuddlynit, you must join me in my journey to mount Sormithiar.

Gully: Whaaaatt??! Are you crazy, I'm just a little girl!

Lina: Like my father once said, 'It dun't tek a beag axe tuh kyut down a beag tree'.

Gully: You're father sounded like he had a speech impediment.
Lina: Nonetheless, let's set off to mount Sormithiar!

Scene 3: Bottom of Mountain

Lina: Here we are Cuddlynit, this is mount Sormithiar. Once you enter, you're the only thing stopping Ril'ust-iarr from mass destruction and world domination.
Gully: ME???!?? Why aren't you coming?!??!? What the hell can I do against a demon?!??!?
Lina: It has come to my attention that other issues that I must unfortunately attend. You shall go and defeat Ril'ust-iarr in my place.
Gully: WHAT!!??!? What could be more important than stopping a demon from rampaging across the land??!??
Lina: .... Other important matters... Good Luck!
Gully: What the hell!!??!?

Scene 4: Inside the mountain

Ril'ust-iarr: Raaaghghhahaa!
Gully: Oh dear...

Appendix C

Times Newspaper Sudoku puzzles used for testing

Super Fiendish
#1946:
```
..7.4..5....7...428....57...9...15..1...9...6..86...1...61....923...6....8..7.3..
```
#1949:
```
.1..7.4..8....9.....4..8..63.7...2..98..2..64..2...8.15..9..6.....3....7..6.8..1.
```

Fiendish
#1948:
```
.4...187.6.......913.5....6.5.1.....8.4...1.7.....9.6.2....7.319.......2.639...8.
```
#1950:
```
..42.3.1.2..8.......8...2.44..3.1.98....9....91.7.4..26.7...5.......8..7.5.4.91..
```
#1943:
```
...42.1..16.7......4......2829......7..1.4..8......9536......9......3.21..1.78...
```

Difficult
#1945:
```
....576.2...91............8.8.3...5665.....1773...6.8.9............89...5.362....
```
#1942:
```
1...4...9.73...21..9.7.1.4...7...4..2...8...7..5...1...2.5.3.9..18...75.5...7...4
```

Mild
#1941:
```
...321.7..3.8...549.......2....7.63.1.......7.78.9....4.......869...7.2..2.934...
```
#1944:
```
63..7..92..9...1..2...1...3.4.1.3.2....754....9.6.2.3.5...4...7..8...9..97..2..16
```
#1947:
```
.48.31.5.9...4...7...6.9..35.2...8..83.....42..4...3.52..9.8...3...7...8.8.32.91.
```

Appendix D

User Manual

The requirements for running SudoKuest on a PC are:

-Windows XP SP2/Vista

-.NET 2.0 framework PLUS the C# 2005 SP1 Redistribution OR .NET 2.0 SP1 framework or higher.

-DirectX9.0c

-XNA Game Studio 2.0

-A graphics card that supports DirectX9.0c and Shader Model 1.1

To run SudoKuest on the Xbox 360, a PC that fits the above requirements is needed. The Xbox 360 must be connected to Xbox LIVE network, with an active Creators Club Membership.

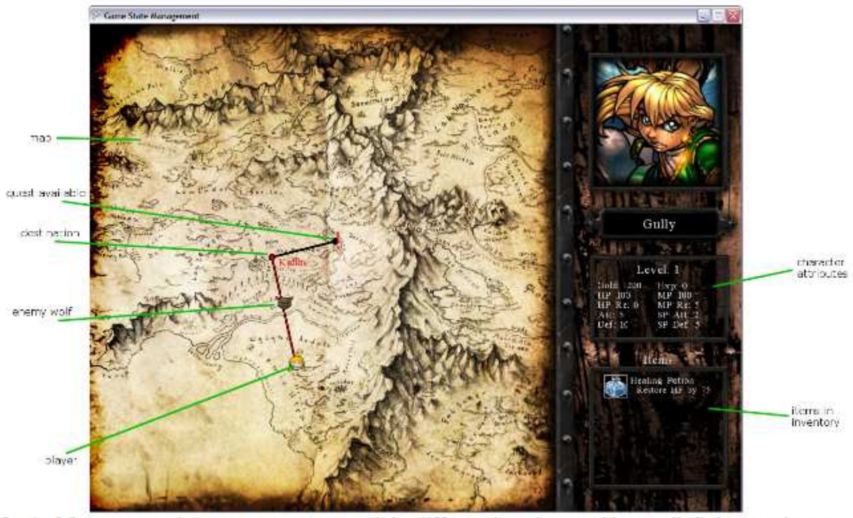
Detailed instructions on how to connect a PC with an Xbox 360 is given at Microsoft's MSDN website: http://msdn.microsoft.com/en-us/library/bb975643.aspx

Sudoku rules

The objective is to fill a 9×9 grid so that each column, each row, and each of the nine 3×3 sub-grids contains the digits from 1 to 9 only *one* time each.

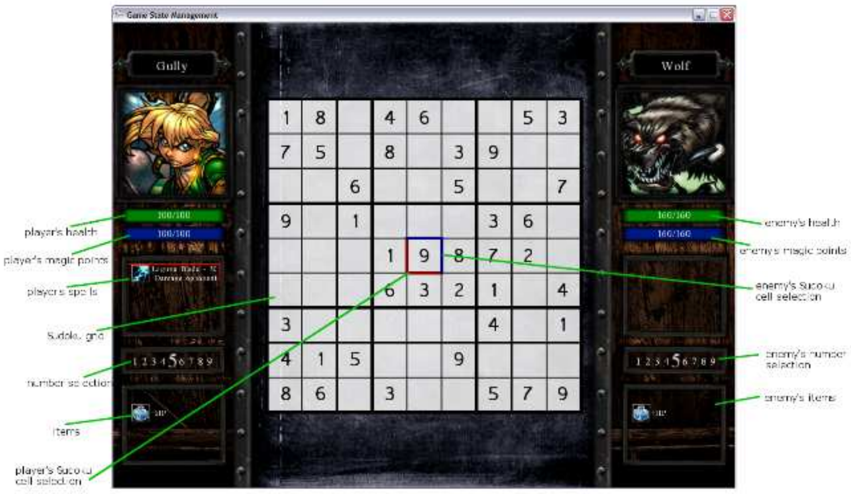
Quest mode

Entering the Player create screen, the player can change the name and attributes of their character that they will use in the Quest. Each attribute determines the statistics of their character. Strength adds health and attack power, Agility adds defence and magical defence and Intelligence adds greater magic abilities. By selecting CREATE, you can start the game and enter the Map screen.



In the Map screen, players can move around the different locations and buy, sell, fight enemies or participate in quests. Places with an exclamation mark shows places that has a quest available. If you run into an enemy between locations, you will have to fight them.

Battle screen



During a fight against an enemy, you have to enter in numbers into the Sudoku grid to damage your opponent. The player who loses all of their health first loses. Spells and items can also be used to aid you in battle.

Spells

Frost Bite: Freezes enemy

Laguna Blade: Attacks enemy

Purify: Heals health

Items

Mithril Hammer: Adds attack power.

Plate Mail: Adds defence.

Robe of Magi: Adds Intelligence.
(Can also be used to cause damage)

Ring of Regeneration: Increases health regen.

Vitality Booster: Increases health

Energy Booster: Increases magic points

Healing Potion: Heals health

Clarity Potion: Heals magic points

Controls:

Command	Keyboard	Xbox control pad
Select/place number	ENTER, Y	A
Cast spell/item (Battle)	Player 1: H,X. Player 2: *	B
Cycle spell/item (Battle)	Player 1: G,J,Z,C. Player 2: /,-	L, R
Cycle number (Battle)	Player 1: T,U, Q, E. Player 2: 0, ..	X, Y

Multi-player

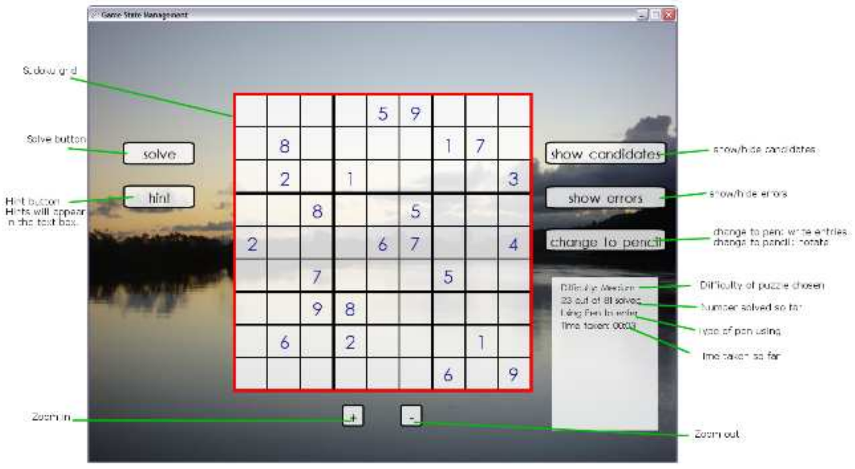
To play networked multi-player, both players must have a LIVE profile. To create a LIVE profile, the user will be prompted with a create profile when they select NETWORK MULTIPLAYER. Alternatively, they can press HOME and the LIVE prompt will appear with profile details.

The first player has to create a game and the other needs to join it. Once connected they will be in the same Lobby with each other, where they are able to use voice chat.

Players may need to unblock the game from their Firewall settings for networking to work.

Puzzle mode

By selecting a difficulty level, you be taken to your randomly generated puzzle.



Difficulty: Medium
23 out of 81 solved
Using Pen to enter
Time taken: 00:03

Options

Here you can change the screen resolution size (if your computer supports it), or enter full screen mode.

Bibliography

[1] TopListed.net - All about Sudoku!
http://sudoku.toplisted.net/about-sudoku.php

[2] Nikoli - Sudoku
http://www.nikoli.co.jp/en/puzzles/sudoku/index_text.htm

[3] Puzzle Club - Jigsaw Sudoku Rules
http://www.saidwhat.co.uk/puzzleclub/jigsaw/jigsaw-sudoku-rules.php

[4] HyperSudoku - History
http://www.hypersudoku.com/GameBuilder/history.php

[5] Concept is Puzzles - Diagonal Sudoku
http://www.conceptispuzzles.com/products/sudoku/diagonalsudoku.htm

[6] Nonograms
http://www.puzzle-nonograms.com/

[7] Nobuhisa Ueda, Tadaaki Nagao: NP-completeness Results for NONOGram via Parsimonious Reductions
http://www.phil.uu.nl/ oostrom/oudonderwijs/cki20/02-03/japansepuzzles/complexity.ps

[8] Takayuki Yato, Takahiro Seta: Complexity and Completeness of Finding Another Solution and Its Application to Puzzles
http://www-imai.is.s.u-tokyo.ac.jp/ yato/data2/SIGAL87-2.pdf

[9] Bertram Felgenhauer, Frazer Jarvis: Enumerating possible Sudoku grids
http://www.afjarvis.staff.shef.ac.uk/sudoku/sudoku.pdf

[10] Ed Russell, Frazer Jarvis: Mathematics of Sudoku II
http://www.afjarvis.staff.shef.ac.uk/sudoku/russell_jarvis_spec2.pdf

[11] Angus Johnson - Simple Sudoku
http://www.angusj.com/sudoku/index.php

[12] UnkownWorlds - Zen of Sudoku
http://www.unknownworlds.com/zen

[13] Scanraid LTD - Sudoku
http://www.scanraid.com/sudoku.htm

[14] Carbonated Games - Sudoku Too
http://www.carbonatedgames.msn.com/games/sudokutoo.htm

[15] Zoonami - Zendoku
http://www.zendokugame.com/

[16] J. Nielsen (1994) Ten Usability Heurisitcs
http://www.useit.com/papers/heuristic/heuristic_list.html

[17] Nikoli - Sudoku - Why hand made? http://www.nikoli.co.jp/en/puzzles/sudoku/hand_made_sudoku.htm

[18] Moritz Lenz - YasSS - Sudoku Generator http://moritz.faui2k3.org/en/yasss#generate

[19] D. Knuth (2000) Dancing Links
http://lanl.arxiv.org/PS_cache/cs/pdf/0011/0011047v1.pdf

[20] H. Hitotumatua, K. Noshita (1979) A technique for implementing backtrack algorithms and its application

[21] PopCap Games - PopCap Developer Program
http://developer.popcap.com/forums/pop_index.php

[22] Microsoft - XNA Developer Center
http://msdn2.microsoft.com/en-us/xna/default.aspx

[23] Sun Microsystems - Java Foundation Classes - Swing API
http://java.sun.com/javase/6/docs/technotes/guides/swing/

[24] J. Madureira, M. Sharieff (1999) Battle Chasers: A Gathering of Heroes
ISBN-13: 978-1563895388

texttthttp://www.amazon.co.uk/Battle-Chasers-Gathering-Joe-Madureira/dp/1563895382/

[25] Gareth Rees, Zoonami Limited (2007) Zendoku Puzzle Generation
http://garethrees.org/2007/06/10/zendoku-generation/

[26] Bob Hanson - Sudoku Exact Cover Matrix
http://www.stolaf.edu/people/hansonr/sudoku/exactcovermatrix.htm

[27] Microsoft (2007) XNA Game State Management Sample
http://creators.xna.com/en-us/samples/gamestatemanagement

[28] Microsoft (2007) XNA Network Game State Management Sample
http://creators.xna.com/en-us/sample/network_game_state_mgt_sample

[29] Michael Anderson (2006) Lines: 2D, thick, rounded line segments for XNA programs
http://blogs.msdn.com/manders/archive/2007/01/07/lines-2d-thick-rounded-line-segments-for-xna-programs.aspx

[30] Top 1465 Sudoku puzzles, ordered by difficulty.
http://magictour.free.fr/top1465

List of Figures

List of Tables