Final Year Project Report

Full Unit – Interim Report

Binary Puzzle Solver and GUI

Bradley Gauntlett

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Magnus Wahlström

Department of Computer Science
Royal Holloway, University of London

12 March 2017

November 17, 2016

Declaration

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 Where other published and unpublished source materials have been used, these have been acknowledged.

1. Word Count:
    1. 14, 417 (Excluding Appendixes)
    2. 15, 709 (Inc. Appendixes)
2. Student Name: **Bradley Gauntlett**
3. Date of Submission: **30/03/2017**
4. Signature:

Table of Contents

Abstract

The basic goal behind this project is to implement a program that will be able to solve Binary Puzzles including a well-designed, user friendly GUI, as well as that, the program will be user interactive in the sense that it will allow custom grids to be inputted by the user and then solved by the algorithm that will be implemented within the program.

The problem itself is that to correctly solve the binary puzzle, there must be no more than two similar numbers below or next to each other. Similarly, each row must be unique and each row and column will have the same number of 1's and 0's.

The algorithm that I will implement will take these principles and correctly provide a solution. As an extended goal for the project, as well as the stand-alone solver, I will add features to provide more interactive features for users as well as help and hints to improve user interaction and experience as well as the generation of grids.

Project Specification / Program Requirements

Aims: To design and implement a program to play and solve Binary Puzzles on Linux.
Background: Binary puzzles are a Sudoku-like game where a grid must be filled with 0 and 1 under some conditions: each cell should contain a zero or a one; no more than two similar numbers below or next to each other are allowed; each row and each column is unique and contains as many zeros as ones. Grids contain some initial marking and the goal is to fill the empty cells.
The goal of the project is to allow people to play this game off-line, generate new grids, solve existing ones.

Early Deliverables

1. Proof of concept programs which tests if a grid is correct and complete
2. Proof of concept program which gives a solution to a grid.
3. Proof of concept GUI design.
4. Report on design patterns and code organisation.
5. Report on Backtracking and Recursion.

Final Deliverables

1. The program must have a reasonable design, with a complete, documented implementation.
2. The program must at least accept grids of a different sizes, allow a user to fill it with a GUI, test if a completed grid is correct and solve grids if asked.
3. The program must be able to produce grids that have only one correct solution.
4. The report will describe the software engineering process involved in generating your software.
5. The report will describe at least a backtracking and recursion solving algorithm and will compare them (including benchmark).
6. The report will discuss the choice of data structures and their performance impact on the solving and generation algorithms.
7. The report will discuss if the program can be open-source and patent-free.

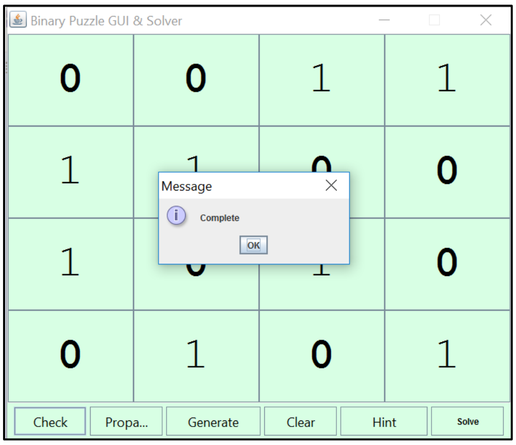
Suggested Extensions

- The program gives hints for users when they are stuck.
- The program has a difficulty index for generated grids.
- The program can get the binary puzzle of the day from the website.
- The program can support variations of the rules (using 3 numbers, rectangular, etc.)
- The program support printing of grids.

Chapter 1: Introduction

This report which will detail in varied amounts of depth the work that I have carried out so far, including proof of concept programs, initial readings which include the 2 reports in the project spec on 'Design Patterns and Code Organisation' and 'Recursion and Backtracking', a section on 'Professional Issues' that could be related to this and other similar software development projects, and also I will talk about the 'Software Engineering Process' and what techniques I have used, a section on 'Open Sources Code', as well as organisational factors and planning and time scales including my work diary and SVN commit log.

The program itself will consist of a few main components, firstly a Verifier for game grid, this algorithm will take a grid current state and systematically check through to check its correctness, the GUI in this phase has been designed so that it will change colour to signal to the user its correctness, Figure 1: GUI output when a correct grid is entered and Verification algorithm is run via the check button.



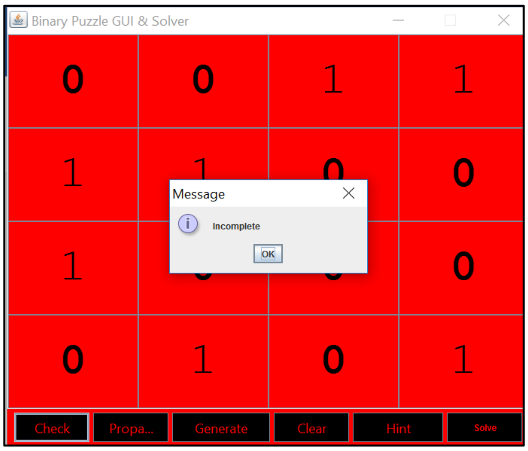
0	0	1	1
1	1	0	0
1	0	1	0
0	1	0	1

Message

Complete

OK

Figure 1: GUI output when a grid is correct.



0	0	1	1
1	1	0	0
1	0	0	0
0	1	0	1

Message

Incomplete

OK

Figure 2: GUI output when an incorrect grid is entered

The second part is a solving algorithm which will consist of 2 components: a propagation solver and a recursive solver using backtracking techniques, the combination of these 2 techniques will be used to solve the puzzles in an optimal way, more technical detail of this will be explained in later chapters.

The problem itself, Binary Puzzles, is a Boolean Satisfiability Problem which is NP-complete, in the sense that each cell can only be one of two possible values we call this a binary variable array. Due to the fact that Binary puzzles must adhere to the aforementioned conditions of their correctness stated in the *Abstract*:

'The problem itself is that to correctly solve the binary puzzle, there must be no more than two similar numbers below or next to each other(i). Similarly, each row must be unique (ii) and each row and column will have the same number of 1's and 0's. (iii)'.

“the conditions are considered distinctly and the computation is divided three-ways, where each part corresponds to a condition. That means we consider the following collections of m × n binary arrays that are constrained:

$A^{m\times n} = \{X \in F^{m\times n}{}_2 \mid X$ satisfies (i) $\}$;
$B^{m\times n} = \{X \in F^{m\times n}{}_2 \mid X$ satisfies (ii) $\}$;
$C^{m\times n} = \{X \in F^{m\times n}{}_2 \mid X$ satisfies (iii) $\}$;
$D^{m\times n} = \{X \in F^{m\times n}{}_2 \mid X$ satisfies (i), (ii) and (iii) $\}$;
$F^{m\times n} = \{X \in F^{m\times n} \mid$ each column of X is balanced $\}$;

where $F^{m\times n}{}_2$ is the set of all m × n binary arrays. “

(Utomo and Pellikaan, 2015: p2)

My motivation behind doing this project is that the concept behind the algorithms can get quite complex, so this will help improve my skill in writing recursive algorithms as well as understanding the heavy underlying theory relating to the structures and logic behind this kind of problem. I will be using Java for this project will allow me to extend the Java and Swing knowledge I have already and put existing skills into practice through using software engineering techniques.

1.2 Literature Review and Background Reading

To help me understand concepts related to my project I have done a certain amount of research into various aspects of the project, in particular I have been using books on ‘Software Engineering’ and ‘Data Structures in Java’, the Pearson Software Engineering book I believe was on the recommended reading list for the CS2800 Software Engineering course and has reiterated relevant sections of knowledge relating to Design Patterns, also has been used to aid me in writing the below report named ‘Design Patterns and Code Organisation’.

As well as book resources, I have used various websites that are specific to Binary Puzzles, these were to help me understand the logic and strategies related to checking and solving. Some of these strategies I have implemented in my code, see 4.3.1 for more details.

In terms of understanding the concepts behind the solving algorithm, I have used mainly a series of lecture slides from Harvard Faculty of Arts and Science (David G.Sullivan PhD, (n.d.).).

This presentation has helped me greatly in understanding this concept, at first I had a real struggle on getting a grasp on Recursive Backtracking, however now I feel, conceptually, that I understand the process.

Other sources that I have used include a report by P. Utomo and R. Pellikaan of the Eindhoven University of Technology. Department of Mathematics and

Computer Science, Coding and Crypto. This report relates to constrained arrays and has specific examples relating to Sudoku and Binary Puzzles, an interesting and high-level resource.

All references are cited in my bibliography in the final chapter of this report, the majority of my background reading has been described in application within further sections of this report where the chapter has been labelled (Report).

Chapter 2: Interim Project Work

This chapter aims to detail all the work undertaken before the project in preparation and all the work undertaken between September and the Christmas break that was submitted as part of the interim review in December.

2.1 Proof of concept programs

2.1.1 Grid Verifier

For my grid verification proof of concept program i have produced an iterative solution that verifies if a grid is complete or incomplete, as a proof of concept I have used a 2-dimensional array rather than the GUI for simplicity.

The algorithm uses a simple combination of nested for loops and IF statements. It works by iterating through the 2D array used as the grid, and for each cell will check if both cells below and above are the same, it does the same for the sides as well, if any of these conditional statements return that there are 3 in a row it will return incomplete, likewise, if the rows and columns do not have an equal number of 1s and 0s for each the program will also return incomplete and will display where the grid fails.

2.1.2 GUI Design

Below is a screenshot of my GUI running, it has been programmed in Java using Swing as I previously mentioned, I think the aesthetics are presently fit for purpose and for the final design I will have added buttons so check, solve and generate puzzles.

2.1.3 Propagation Solver

For my solver for my proof of concept programs I implemented a propagation solver, which essentially loops through the grid checking for known patterns and filling the blank cells in each pattern with the inverse value:

I am using iteration for the propagation of the grid, so filling in gaps that I know must be one or the other, see below:

4.3.1.1 'Piggy in the middle':

Using nested for loops I am able to iterate through the 2-dimensional array and check for various conditions, one that is looked for is a 'Piggy in the middle' this is where a gap is placed between 2 of the same number and thus we know that the gap must the opposite.

1		1	

1	0	1	

4.3.1.2 Finding pairs:

Another strategy I'm using for the initial solver is to find pairs in the grid, to do this I iterate through the grid looking for 2 spaces that have 2 one way or 2 the other way, then this allows the program to know that that space must be the opposite of the state of the paired cells.

	1	1	

0	1	1	0

Similarly, for pairs facing a wall or only placed next to one empty cell, the same rule must be applied:

		1	1
		0	
		0	

	0	1	1
		1	
		0	
		0	

- ***Please See 4.3.1 My Strategy***

Chapter 3: Design Patterns & Code Organisation (Report)

3.1 What are Design Patterns

Design patterns are a method for object-oriented design, they are well known in software engineering as a 'description of good practice', different design patterns have different purposes and it is the job of the Engineer to decide when to use each as appropriate, for example some are used for architectural purposes whereas others are used for behavioural interaction.

3.2 Types of Design Patterns

3.2.1 Behavioural

Observer

The observer pattern is uses displays which are used to separate the actual object and its state, it uses the state to provide multiple displays and allows the displays to be updated from the data.

An example of this is where you may have multiple graphical displays that all use data from the same source and, if the source data changes the design pattern is built so it will know when the state has changed, in this case the source data, and will then update the graphs (displays).

Potentially I could have used this when the GUI is displayed after the verifier and solver get called on the grid.

3.2.2 Structural

Adapter

The Adapter design pattern is used to provide an interface to third party resources, or in other words it is a way in which two incompatible interfaces can be used together, an example of this could be a database.

A benefit of using the Adapter design pattern is that it decouples the client from the underlying back-end system.

3.2.3 Architectural

MVC

MVC stands for Model, View, Controller, an architectural design where the purpose of using this design pattern is to de-couple the model and view classes which will decrease complexity.

The MVC design pattern is widely used in Software Engineering as it is applicable to pretty much any type of project. MVC promotes re usability and makes it simpler for developers to maintain code and offers increased flexibility for growth and further development of the system.

It separates the presentation of data and user interaction from the actual system data, the Model component is used to manage the system data and interactions/operations that can be carried out on that data e.g. SQL queries. The Controller component interprets interactions and communicates these interactions with the View and Model components and finally the View quite simply is the UI and controls how system data gets presented to the user.

3.2.4 Creational

Singleton

The purpose of using this design pattern is to restrict instantiation to just one single class and the general purpose of this is to centralise elements of the system, for example the Model component in MVC pattern is usually a Singleton implementation.

The instance created in the Singleton class has a private constructor which allows only one instance to be instantiated and that instance can be accessed from anywhere in the system, however as the constructor is private new instances cannot be created.

A downside of using the Singleton design pattern is that it limits the ability to subclass.

Factory

The Factory design pattern is a creational design pattern which is used to create objects. Factory will be used to create objects without the user interacting with the logic of the object construction.

In my program, I would hope to use a factory for generating the grids and another factory for generating single solution grids.

3.2.5 My use of Design Patterns

Interim Design Pattern Deliberation

I decided not to utilize design patterns for my proof of concept programs, however will be using them for my main development.

MVC (described above will be used for certain), Singleton will need to be used in at least one instance (for the Model), the rest I will plan further and make decisions over the Christmas break. Once design pattern i am particularly interested in is the Factory, I would like to try to use this to generate grids, also could create another for the extended task of generating random grids with only one solution.

See: Software Engineering Process - Design Patterns used

3.3 Code Repositories and Version Control

3.3.1 SVN

SVN (Subversion) is a version control system that is usually used for software development in teams, although this project is obviously not a team project I am using it so that my supervisor can have access to my code at any time.

3.3.2 Branches

A Branch is essentially a snapshot of the trunk at that time which is used as a working copy, in most cases there will be multiple branches used for developing separate tasks independently, once the code is complete for a specific branch the branch can be merged back to the trunk providing the code has been checked to keep the trunk stable and solid code.

In my code for I haven't used branches, purely because as I am working on this independently there isn't much need to, although saying this, it may have been a good idea in some cases for example when I changed from working on the proof of concept programs, however, in reality it was a bit too time consuming where the need for the process wasn't that high.

3.3.3 Tags

Tags are essentially copies of the trunk that the team wish to preserve, this could be for a number of reasons, most commonly it is when major releases occur, alternatively it could be to preserve the current most stable point of the trunk – essentially this case would act as a backup if the trunk was corrupted or merged with a branch that contained 'bad' code.

In my code for the proof of concept programs and for the early development of the project I haven't used tags, this is purely because the outputs are not complete with reasonable functionality for the user. A release tag will be used for the final verison of the product.

3.4 Packages

Packages are an organisational technique to logically arrange and structure related functionality, i.e. combining classes and interfaces that are related.

Packages are very important in large software project as it allows developers to easily organise code which helps for maintainability as well as organisational issues.

Chapter 4: Recursion and Backtracking (Report)

4.1 What is Recursion

Simply a recursive algorithm or function is one where it repeatedly carries out an action until the output is correct, in programming to do this a function is defined queries a base case, does some computation and then carries out the recursive call by calling itself (the function) which passes the new values i.e. the ones that have been obtained through the computation, through the function again, this process is then repeated until the base case is satisfied.

4.2 Recursion Examples

In this sub-section I will talk about a few basic recursive programs and explain code examples written in Java.

4.2.1 Factorial

Below is an example of a simple program that works out factorials recursively, as you should be able to see from the code it maps my above description of general recursive algorithms.

On line 3 it checks if the given input N is equal to 0, then if it is it will return 1 as 0! = 1. This is the base case, then it enters an else block where it carries out some computation (line 6), this computation multiplies N by the result of the function itself passing N-1, now this action will be repeated. I have also tested this in the main method and it works by outputting 120 when N=5, the recursive steps are as follows;

*5 * (5-1)* (5-2) * (5-3) * (5-4)* Or *5*4*3*2*1 = 120*

```
public static int fact(int n)
    {
        if (n == 0)
            return 1;
        else
            return n * fact(n-1);
    }
    public static void main(String [] args){
        System.out.println(fact(5));
    }
}
```

4.2.2 Raising to a power

Here I have included another simple recursion program which takes 2 arguments, the base and the exponent, recursively the program computes ***base***^{Exp}

```
public static int power (int base, int exp) {
           if (exp < 0)
                 throw new IllegalArgumentException("n >= 0");
           if (exp == 0)
                 return 1;
           else
                 return base * power(base, exp - 1);
      }

      public static void main(String[] args) {
           System.out.println(power(2, 8));
      }
```

4.3 Recursion vs Iteration

Iteration is performing the same process whilst certain conditions may persist, this repetition will continue until a goal or answer has been found and therefore achieved. The simplest way of iteration is where there is an action or a sequence of actions that will be repeated a certain number of time.

As you can see with the comparison to recursion, you obtain the same end result, where they differ is where recursion is lot neater and tidier, also certain algorithms may perform better or worse depending on what technique is chosen.

4.3.1 My Strategy (using iteration)

For my solver algorithm, I am using a combination of both recursion and iteration, firstly I am using iteration for the propagation of the grid, so filling in gaps that I know must be one or the other, see below:

For my solver algorithm, I am using a combination of both recursion and iteration, firstly I am using iteration for the propagation of the grid, so filling in gaps that I know must be one or the other, see below:

4.3.1.1 ‘Piggy in the middle’:

Using nested for loops I can iterate through the 2-dimensional array and check for various conditions, one that is looked for is a ‘Piggy in the middle’ this is where a gap is placed between 2 of the same number and thus we know that the gap must the opposite.

1		1	

1	0	1	

4.3.1.2 Finding pairs:

Another strategy I'm using for the propagation solver is to find pairs in the grid, to do this I iterate through the grid looking for 2 spaces that have 2 one way or 2 the other way, then this allows the program to know that that space must be the opposite of the state of the paired cells.

	1	1	

0	1	1	0

Similarly, for pairs facing a wall or only placed next to one empty cell, the same rule must be applied:

		1	1
		0	
		0	

	0	1	1
		1	
		0	
		0	

4.4 Recursive Backtracking

4.4.1 What is Backtracking

Recursive backtracking is a method of solving computational problems, the concept of how is works is that it will take a case, for example a node, and then applies the computation and checks if it is a viable way of solving the problem, if it isn't the algorithm will return i.e. it will backtrack when it discovers this. That case will then be deemed incorrect and the algorithm will try a new one. There are many applications of backtracking in well-known algorithms, including, the N-Queens problem, Map colouring algorithm, Sudoku solving and of course Binary Puzzle solving, most problems that rely on it are constraint satisfaction problems.

A more practical example of an application of backtracking relates to computer language engineering, parsing and syntax analysis, where the left most derivation parser derives syntax from a language if there is a rule that contradicts the language the parsing algorithm must backtrack to go back to a point where the tree is stable, unsurprisingly this is similar to the application of the concept in the N-Queens and Binary Puzzle solving algorithms, see the worked N-Queens example below 4.4.2.2.

4.4.2 Recursive Backtracking Example

Below is an example using recursive backtracking to find a specific node in a tree that has previously been set to true where all the other nodes have been set false.

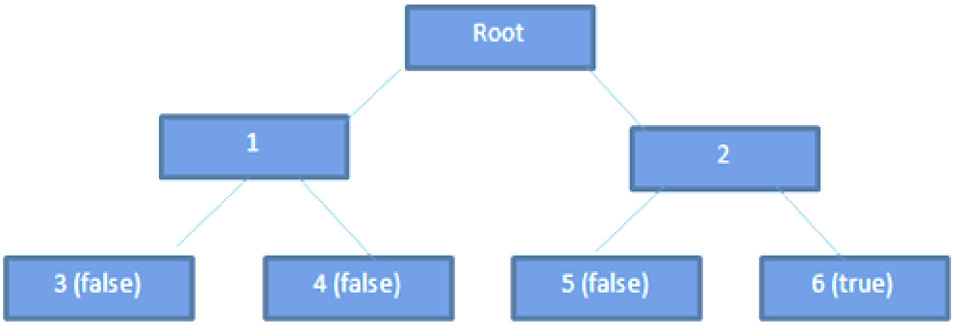
As a more complex example and more relevant to the Binary Puzzle Solution, I will also talk about the N-Queens problem, which uses a recursive backtracking algorithm to place N number of Queens on an N by N grid, without causing conflicts between any of the N queens placed.

4.4.2.1 Searching Algorithm using Backtracking – Simple Example

Here I am going to describe an algorithm for recursively finding a node in a tree.

Outline:

1. Begins at the root
2. Chooses a node to explore
3. Checks if leaf
    a. If leaf check if it is the node it's looking for
    b. If true return true
4. If false explore children and choose one
    a. If leaf check if it is the node it's looking for
    b. If true return true
5. If false return to parent
6. Repeat



Here are the steps for this example (Search for 6):

1. Start at the root, choices = 1 or 2, choose 1 by default

2. At 1 the children are 3 and 4. choose 3. 3 is false so return to 1
3. At 1, 3 is already set to false, so try 4, also returns false so go back to 1
4. All options stemming from 1 have been explored, return to Root
5. At Root, now choose 2
6. At 2 the children are 5 and 6. choose 5. 5 is false so return to 2
7. At 2, 5 is already set to false, so try 6, 6 is true
8. End, found 6

4.4.2.2 N-Queens Problem using Recursive Backtracking – Relevant Example

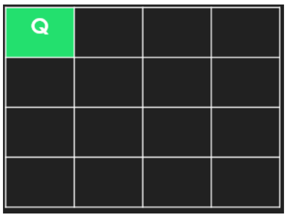
Context of the Problem:

Queens can attack horizontally, vertically, and diagonally. The N-Queens problem questions

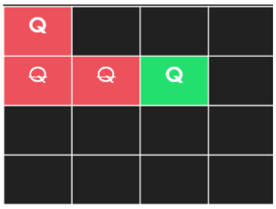
- *"How can N queens be placed on an N x N chessboard so that no two of them attack each other?"*

Worked Example:

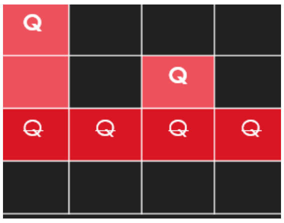
1) We place a Queen in the first cell to begin



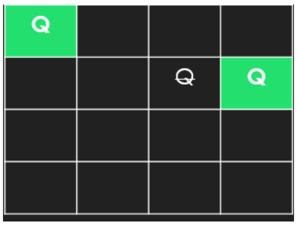
2) We then try the next row, the first 2 cells are compromised by the first Q



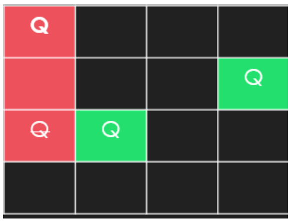
3) This row has no options that can lead to a solution so we need to **backtrack**



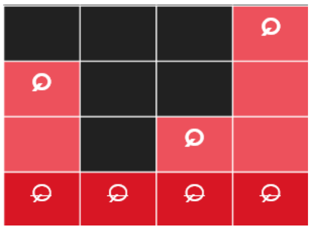
4) We resume from where the Q was placed and proceed – see below.



5) Row is incremented and Q placed on second cell.

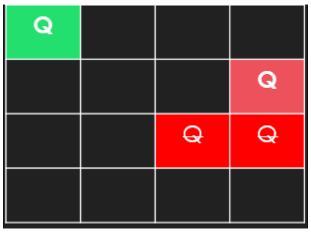


6) All cells result in causing conflicts with other previously placed Qs, so we **backtrack**

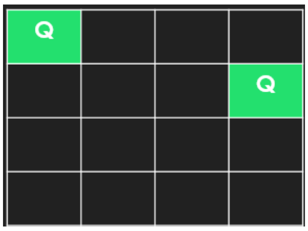


.

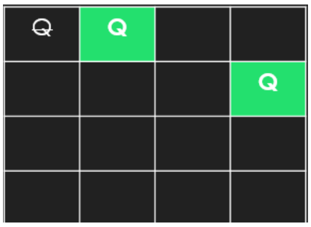
7) We have now backtracked to the previous row so we proceed to check on from where the last Q was placed



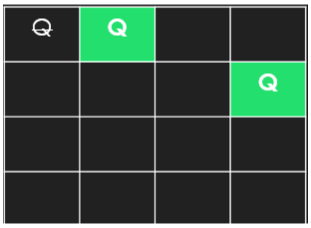
8) Backtrack to the previous row where all have been checked, so we backtrack again



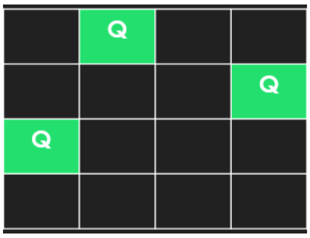
9) We then increment the first queen (column) as we have explored solutions for the first cell.



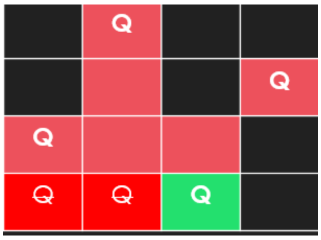
10) All cells in the second row have also been checked and it satisfies the conditions, so it stays.



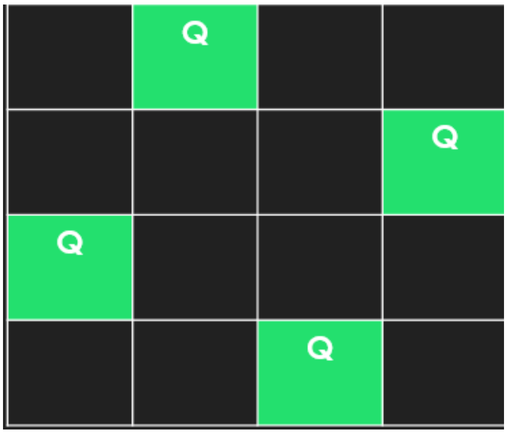
11) We then proceed to the third row, and try the first cell – it satisfies the conditions so a queen is placed



12) Finally, we proceed to the last row to see if this route provides us a solution



13) The route worked and a solution that satisfies all required conditions has been produced.



The algorithm only needs to make N number of instantiations of the queen so the recursion tree is only N deep so has a spatial complexity of $O(n)$ and because of the potentially many recursive calls before reaching the base case the time complexity of this algorithm is exponential $O(c^n)$.

Here in this example I have demonstrated the same backtracking technique used for also solving the Binary Puzzle problem.

There is a slight difference in the algorithm, namely, where the Binary Puzzle solving algorithm will still use the same structure, there are extra validation steps needed firstly to check whether a cell is blank or not, the N-Queens algorithm doesn't have this problem as the row is incremented as soon as a cell is declared as valid and a Queen is placed, this leads to another difference with the algorithms, such that the N-queen's problem only needs to place one on row whereas the Binary puzzle solving algorithm will potentially need a lot more steps as the columns still need to be checked after a blank cell has been set to either a 0 or a 1, additionally, there needs to be a call to another method to check if the entire grid is valid.

However, the N-Queens algorithm doesn't have this step as once the queen is placed on the Nth row the algorithm is complete and the recursion returns to the base case.

Code snippets for N-Queens algorithm

Base Case:

```
if (row == N){
     displayCompletedGrid();
}
```

As mentioned previously the base case for this recursive algorithm is when the row reaches the Nth increment as the whole route to the Nth pass will be valid, we know this because, as demonstrated in the example, if the algorithm reaches the final column without finding a valid cell it will backtrack, and this rule is independent whether it is on any row that is not the 0th row.

Recursive call:

```
for (int columnCtr = 0; columnCtr <= N; columnCtr++){
     if (safeHere(rowCtr, columnCtr)) {
          putQueenHere(rowCtr, columnCtr);
          findAColumn(rowCtr + 1);
          // where the Backtracking occurs
          deleteQueen(rowCtr, columnCtr);
     }
}
```

This snippet shows the recursive call, the algorithm enters a for loop which increments through the columns, as soon as the loop is entered it goes into an If-statement which makes a call to another method checking if the position is safe and passing the 2 coordinate parameters.

If this method returns true, then, the Queen will be placed and the method will make the recursive call, calling itself and incrementing the rowCtr variable, now if we pass this point we need to backtrack, the algorithm then calls the delQueen method which passes the rowCtr variable, which as we have previously put rowCtr + 1 and not rowCtr++ or rowCtr = rowCtr +1, we have the correct value to pass, then the columCtr variable is stored in the stack frame so the algorithm can continue from where it previously was.

Binary Puzzles:

4.4.2.3 Simple Binary Puzzle Solving Algorithm Example

We start with this randomly partially filled grid, after this as previously mentioned in the explanation of how my solution works, the propagation solver then loops through the grid popping in values that are pre-defined in the specification as patterns where the consecutive cell must be set as the inverse bit value.

1		1	
	0		0
0		0	
	1	0	

1	0	1	
	0	1	0
0	1	0	
	1	0	

In the next part of the algorithm we enter the recursion. The method behind this part is to have an initial for-loop within a recursive method, the method will later then call itself with the row being incremented, as we saw with the above N-Queens example.

1	0	1	
	0	1	0
0	1	0	
	1	0	

1	0	1	1
	0	1	0
0	1	0	
	1	0	

1	0	1	0
	0	1	0
0	1	0	
	1	0	

We try a 1 in the first blank cell, this fails, so it breaks from the if block and gets set to the inverse, 0. The 0 doesn’t break any rules so the algorithm continues. The next blank gets set to one and this returns valid

1	0	1	0
	0	1	0
0	1	0	
	1	0	

1	0	1	0
1	0	1	0
0	1	0	
	1	0	

1	0	1	0
1	0	1	0
0	1	0	1
	1	0	

The next blank cell gets set to 1 and this also passes so it is kept, and as there are no more contradictions the algorithm will not need to backtrack.

1	0	1	0
1	0	1	0
0	1	0	1
1	1	0	

1	0	1	0
1	0	1	0
0	1	0	1
0	1	0	

The loop then iterates to the next blank, and the algorithm is set to one, as already demonstrated, when set to one the isValid method will fail as there are 3 ones in the column, so the cell is set to the inverse.

1	0	1	0
1	0	1	0
0	1	0	1
0	1	0	1

The final cell is set to 1 and this completes the algorithm.

Raw Console Output Example from my Algorithm

Here is an example of the console after adding print statements after each move is made, where 9 = a blank cell.

```
1*****GRID****
[9][1][9][1]
[1][9][0][9]
[1][0][9][9]
[9][0][9][9]
```

```
2*****GRID****
[0][1][0][1]
[1][1][0][9]
[1][0][9][9]
[9][0][9][9]

3*****GRID****
[0][1][0][1]
[1][1][0][0]
[1][0][1][9]
[9][0][9][9]

4*****GRID****
[0][1][0][1]
[1][1][0][0]
[1][0][1][0]
[0][0][1][9]

5*****GRID****
[0][1][0][1]
[1][1][0][0]
[1][0][1][0]
[0][0][1][1]
```

Binary Puzzle Solving Algorithm – Example that uses backtracking

Here is a worked example demonstrated in the same in the same fashion as how my algorithm would with added efficiency checks that are not present in my version.

```
static boolean solve(int i, int j, int[][] cells) {
          if (i == Model.n) {
               i = 0;
               if (++j == Model.n)
                    return true;
          }
          for (int h = 0; h < 1; h++) {
               if (cells[i][j] != 9) {
                    return solve(i + 1, j, cells);
               }
          }
          for (int val = 0; val < 2; ++val) {
               cells[i][j] = val;
               if (isValid(cells, i, j)) {
                    cells[i][j] = val;
          if (solve(i + 1, j, cells) && validate(cells) == true)
                         return true;
               }
          }
          cells[i][j] = 9; //Reset to blank in UI
          return false;
```

```
        }
```

1) We start in a linear fashion starting with cell 0, 0.

0			

2) We progress by looping through the row, placing the second element to zero as well, this move proves fine by the isValid() method.

0	0		

3) In the proceeding cell after we attempt to place a zero however, this is invalid

0	0	0	

0	0	1	

4) a zero in the next position is also invalid so a 1 is placed as a result.

0	0	1	1

5) A zero in the next position, this is valid, however the proceeding move (if zero) will cause the rows to be the same bit patterns, this is invalid, so after the two resulting 1's have been placed the algorithm will backtrack and set the second zero to a one and proceed in the usual way.

0	0	1	1
0	0	1	1

0	0	1	1
0	1	0	1

6) A zero is placed on the next row, this is determined as in valid and set to a one, the proceeding cells are then set and a conflict has occurred where 2 zeros and 2 ones lead to the same blank, the algorithm then backtracks to set the previous zero cell to a one and continues.

0	0	1	1
0	1	0	1
0			

0	0	1	1
0	1	0	1
1	0	0	0

0	0	1	1
0	1	0	1
1	0	1	0

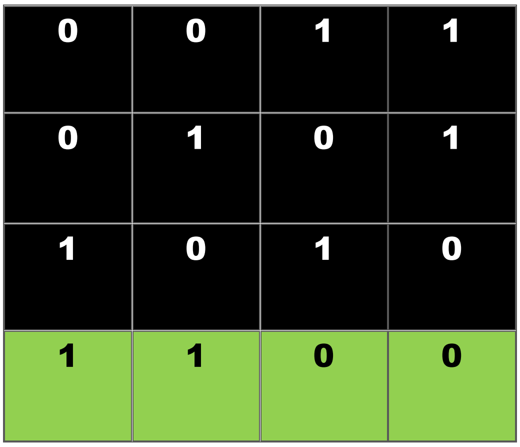
7) A zero is placed on the next row, this is determined as invalid as would cause an unbalanced column error, resultantly, it is set to the inverse, same case is present for cell 2 in the final row.

0	0	1	1
0	1	0	1
1	0	1	0
0			

0	0	1	1
0	1	0	1
1	0	1	0
1			

8) The final pass is carried out successfully and we have reached the last cell with a valid grid using recursive backtracking techniques this algorithm solves the grid efficiently, reaches the base case and returns.

The Final grid is as follows:



0	0	1	1
0	1	0	1
1	0	1	0
1	1	0	0

4.5 N Queens vs Binary Puzzle Solving Algorithms Comparison and Benchmark

The algorithm only needs to make N number of instantiations of the queen so the recursion tree is only N deep so has a spatial complexity of $O(n)$ and because of the potentially many recursive calls before reaching the base case the time complexity of this algorithm is exponential $O(c^n)$.

Here in this example I have demonstrated the same backtracking technique used for also solving the Binary Puzzle problem.

There is a slight difference in the algorithm, namely, where the Binary Puzzle solving algorithm will still use the same structure, there are extra validation steps needed firstly to check whether a cell is blank or not, the N-Queens algorithm doesn't have this problem as the row is incremented as soon as a cell is declared as valid and a

Queen is placed, this leads to another difference with the algorithms, such that the N-queen's problem only needs to place one on row whereas the Binary puzzle solving algorithm will potentially need a lot more steps as the columns still need to be checked after a blank cell has been set to either a 0 or a 1, additionally, there needs to be a call to another method to check if the entire grid is valid.

However, the N-Queens algorithm doesn't have this step as once the queen is placed on the Nth row the algorithm is complete and the recursion returns to the base case **– *See N-Queens section above ^***

My method of solving the binary puzzle builds upon the same idea as the N-Queens algorithm, in the sense that the base case is when the final row is completed the algorithm is complete, obviously using the game specific rule set and the fact that all cells must be assigned a value.

As this is the case it means that the recursion tree is N deep as it goes through row by row before returning on completion, so the spatial complexity is $O(n)$, the time complexity is also exponential, or $O(n^2)$ since each function call calls itself twice located in each of the two for loops, the one to skip filled cells and the other used to set the value of the cell, unless it has reached the end of the 2-dimensional array and has returned a Boolean value.

Execution Benchmarks:

Both tests were conducted using the respective backtracking algorithms, the one for the N-Queens I did not program myself, rather I am using the following code found online:

```
public class Queens {
    public static boolean isConsistent(int[] q, int n) {
        for (int i = 0; i < n; i++) {
            if (q[i] == q[n])             return false;
            // same column
            if ((q[i] - q[n]) == (n - i)) return false;
           // same major diagonal
            if ((q[n] - q[i]) == (n - i)) return false;
           // same minor diagonal
        }
        return true;
    }
    public static void printQueens(int[] q) {
        int n = q.length;
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (q[i] == j) System.out.print("Q ");
                else           System.out.print("* ");
            }
            System.out.println();
        }
        System.out.println();
    }


    public static void enumerate(int n) {
        int[] a = new int[n];
        enumerate(a, 0);
    }
    public static void enumerate(int[] q, int k) {
        int n = q.length;
        if (k == n) printQueens(q);
        else {
            for (int i = 0; i < n; i++) {
                q[k] = i;
                if (isConsistent(q, k)) enumerate(q, k+1);
            }
        }
    }
    public static void main(String[] args) {
      long startTime = System.nanoTime();
      int n = 4;
       enumerate(n);
      long endTime = System.nanoTime();
      long duration = (endTime - startTime);
        System.out.println(duration);
    }
}
```

(Sedgewick and Wayne, 2016)

Results are given in Nanoseconds:

N-Queens Solver	**732032ns**
Binary Puzzle Solver	**115539346ns**

Tested both on a 4x4 grid:

Considerations:

- Binary Puzzle needs to place N squared amount of values whereas N-Queens needs to place N values.
- Binary Puzzle benchmark was tested on my actual product so GUI update time should have been factored in, whereas the N-Queens used a CLI.

Chapter 5: The Software Engineering Process

5.1 Requirements:

Gather requirements / break problem into tasks

5.1.1 Task Slicing:

The process of dividing up a project into tasks and categorising those tasks is called slicing. Slicing is the process of developing a part of the overall system, considering prioritisation and choice of tasks. The 2 types of slicing are horizontal and vertical, and they both accomplish different goals.

Horizontal slicing is used to develop one main part of the system e.g. modules/classes, to gradually build up the core elements of the system, this must be used sometimes with certain types of systems but can lead to 'over-engineering' and it delays feedback from the client which could easily result in developing undeliverable code. Vertical slicing however, is used to develop paths throughout the system which have some functionality even though this can be faked and is usually very limited. The positive aspect form using this method is that it gets the client involved with the development very early on and promotes feedback and frequent releases each with added incremental functionality.

Vertical slicing is the focus of Agile development whereas, the horizontal method is the basis of the, more-or-less redundant traditional method (waterfall) where functionality is only available at towards the end of the project; however, some core components of the system will need to be developed using this process.

The natural solution (Agile development process) would be to use horizontal slicing where required to and using vertical as much as possible to add functionality quickly; this also allows us to spot potential integration issues.

In my project, I have mostly used horizontal slicing as it is an individual project and the way my milestones have been set out it made sense to develop the software components incrementally, even though this isn't my preferred method, However, in a sense I have used horizontal if you consider component interaction, i.e. the Solver needing to use the Validator.

5.1.2 Task Breakdown:

Task: GUI initial prototype
Difficulty: Medium
Importance: Moderate / Very
Time (Start–Fin): Less than 1 week

Task: GUI full implementation
Difficulty: Medium
Importance: Very
Time (Start–Fin): Multiple weeks however work has been scattered between tasks

Task: GameButton class
Difficulty: Medium with difficulties
Importance: Moderate / Very
Time (Start–Fin): less than 1 week

Task: Grid Verifier initial prototype
Difficulty: Medium with difficulties
Importance: Moderate / Very
Time (Start–Fin): 1-3 weeks

Task: Grid Propagation Solver initial prototype
Difficulty: Medium with difficulties
Importance: Moderate / Very
Time (Start–Fin): 3-4 weeks

Task: Setup MVC and re-structure code base
Difficulty: Easy / Medium
Importance: Moderate
Time (Start–Fin): Less than 1 week

Task: Model Class
Difficulty: Easy - Medium
Importance: Not Essential but important to good practice
Time (Start–Fin): Less than 1 week

Task: Controller Class
Difficulty: Easy - Medium
Importance: Not Essential but important to good practice
Time (Start–Fin): Less than 2 days (inc. gaps in work)

Task: Verification Algorithm integration with updated GUI
Difficulty: Medium with difficulties
Importance: Very
Time (Start–Fin): 1-2 weeks

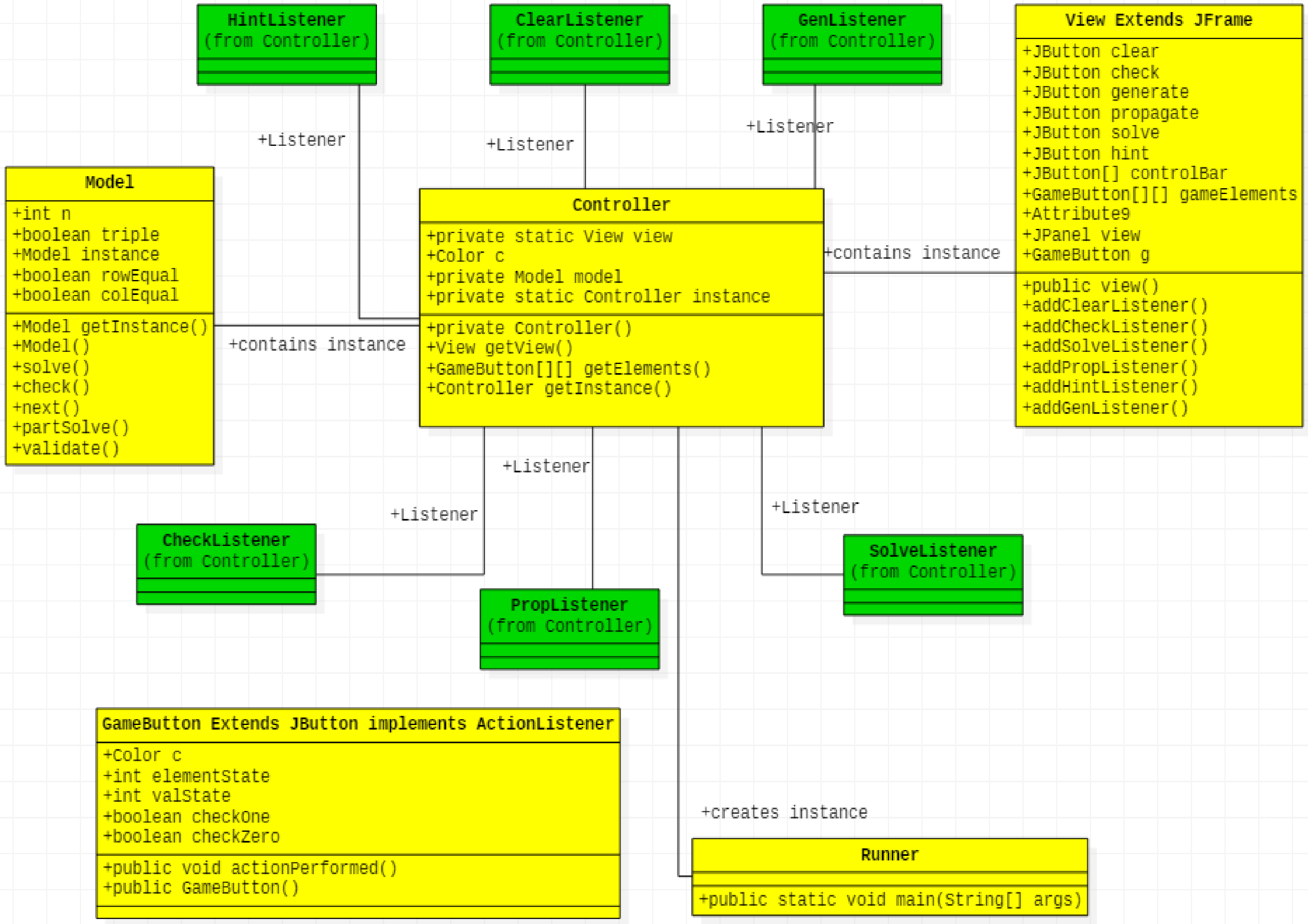
Task: Propagation Algorithm integration with updated GUI
Difficulty: Medium with difficulties
Importance: Not Essential when full solver is complete
Time (Start–Fin): 1-2 weeks

Task: Hint feature implantation
Difficulty: Easy
Importance: Not Essential
Time (Start–Fin): less than 3 weeks

5.2 Design / Analysis

- **UML**

Below is the UML diagram for my program but please note as this was produced before the program was finished some elements may have changed, however the general structure is still the same.


HintListener
(from Controller)
ClearListener
(from Controller)
GenListener
(from Controller)
View Extends JFrame
+JButton clear
+JButton check
+JButton generate
+JButton propagate
+JButton solve
+JButton hint
+JButton[] controlBar
+GameButton[][] gameElements
+Attribute9
+JPanel view
+GameButton g
+public view()
+addClearListener()
+addCheckListener()
+addSolveListener()
+addPropListener()
+addHintListener()
+addGenListener()
+Listener
+Listener
+Listener
Model
+int n
+boolean triple
+Model instance
+boolean rowEqual
+boolean colEqual
+Model getInstance()
+Model()
+solve()
+check()
+next()
+partSolve()
+validate()
Controller
+private static View view
+Color c
+private Model model
+private static Controller instance
+private Controller()
+View getView()
+GameButton[][] getElements()
+Controller getInstance()
+contains instance
+contains instance
+Listener
+Listener
+Listener
CheckListener
(from Controller)
PropListener
(from Controller)
SolveListener
(from Controller)
GameButton Extends JButton implements ActionListener
+Color c
+int elementState
+int valState
+boolean checkOne
+boolean checkZero
+public void actionPerformed()
+public GameButton()
+creates instance
Runner
+public static void main(String[] args)

- **User Stories**

As a game user, I want to be able to load new puzzles so that can play the game.

As a game user, I need to be able to change cell states and have them displayed to me in the grid instantaneously by clicking on the cell so that can play the game.

As a game user, I want to be able to load new puzzles and have the pre-set cells 'locked' so that can play the game with an element of difficulty appose to being able to edit every cell.

As a game user, I want to be able to solve grids that I have loaded to the screen so that I can see the solution if I can work out the puzzle myself.

As a game user, I want to be able to ask the program for hints if I get stuck so that I can solve the puzzle myself without using the solve button.

As a game user, I want to be able to clear the grid so that I can start the game again if I want to.

As a game user, I want to be able change the size of the grid to allow me to play instances of the game of varying difficulty so that I can play with more variation of puzzles.

As a game user, I want to be able to check the current state of the grid and receive feedback from the program to tell me if the grid isn't solved correctly so that I can know if I need to change cell states or start the game over.

As a game user, I want to be able to fill the grid where a loaded grid starts with 2 in a row for example the next blank cell will be set to the inverse number so that solve the grid myself without using the full solver.

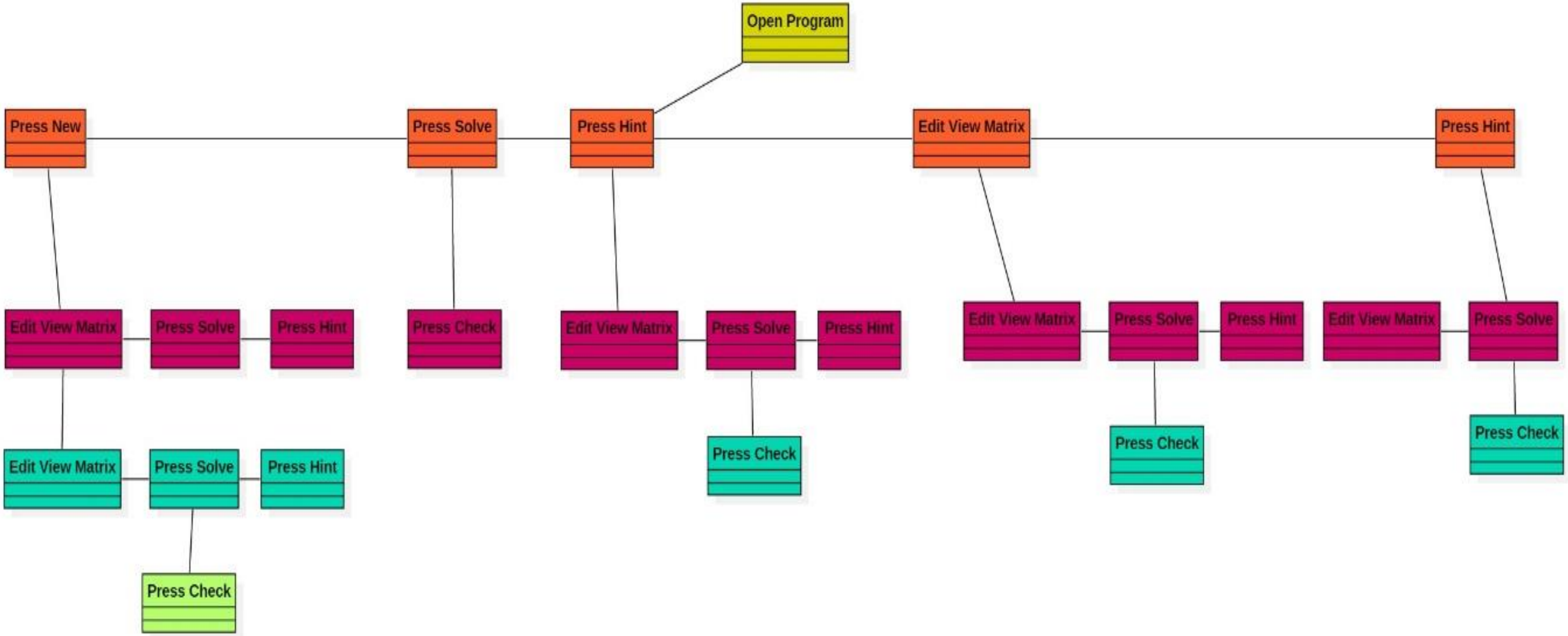
5.2.1 HCI

- **Hierarchical Task Analysis (HTA)**

"A hierarchical task analysis provides an understanding of the tasks users need to perform to achieve certain goals."

(Hornsby, 2010)

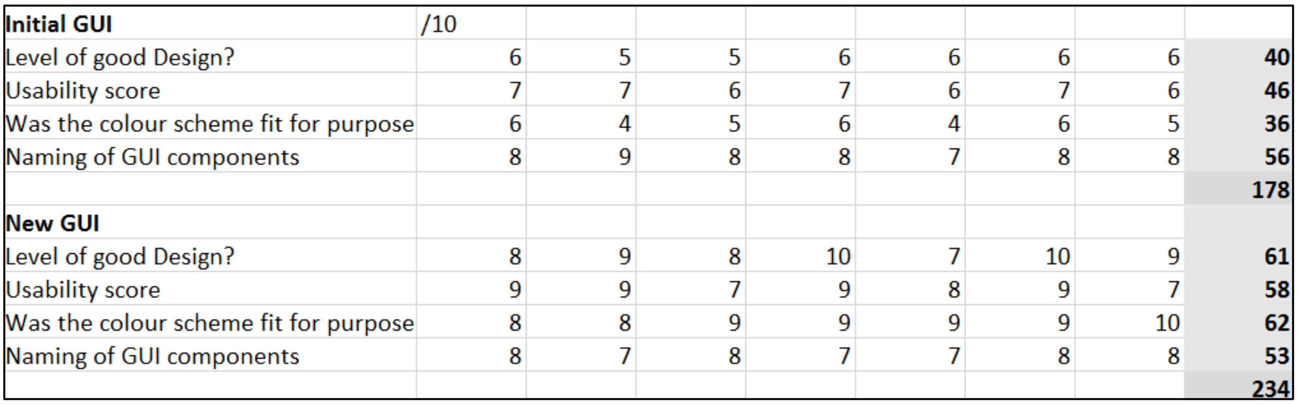
Below Is an image of my HTA diagram produced in StarUML:


Open Program
Press New
Press Solve
Press Hint
Edit View Matrix
Press Hint
Edit View Matrix
Press Solve
Press Hint
Press Check
Edit View Matrix
Press Solve
Press Hint
Edit View Matrix
Press Solve
Press Hint
Edit View Matrix
Press Solve
Edit View Matrix
Press Solve
Press Hint
Press Check
Press Check
Press Check
Press Check

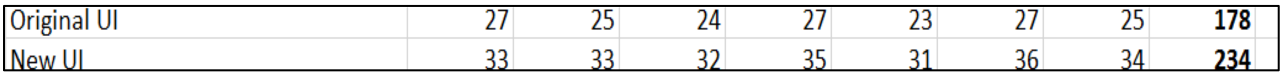
- Statistical Analysis for Variable Design Considerations

Question: A comparison between initially developed GUI compared to the new implementation.

Data Sample



Initial GUI	/10							
Level of good Design?	6	5	5	6	6	6	6	**40**
Usability score	7	7	6	7	6	7	6	**46**
Was the colour scheme fit for purpose	6	4	5	6	4	6	5	**36**
Naming of GUI components	8	9	8	8	7	8	8	**56**
								178
New GUI								
Level of good Design?	8	9	8	10	7	10	9	**61**
Usability score	9	9	7	9	8	9	7	**58**
Was the colour scheme fit for purpose	8	8	9	9	9	9	10	**62**
Naming of GUI components	8	7	8	7	7	8	8	**53**
								234



Original UI	27	25	24	27	23	27	25	**178**
New UI	33	33	32	35	31	36	34	**234**

T-Test

For these data sets I am using a T-Test to compare the overall UX for the interface based on 4 factors that I deem to be important, detailed in the above table.

I chose to use the T-Test because was simply comparing the 2 lists of values in the data sets; in this case I was comparing 7 users against 2 different variant conditions, the variables being the colour scheme and names of the command buttons.

- Hypothesis

The mean(UX_Original) < mean|(UX_New)

If this statement is true, then I should use the new interface for the deployment of the product. However, if the above boolean expression is false and mean(UX_Original) > mean|(UX_New) then the old version of the interface should be used on deployment.

The NULL hypothesis of this experiment is

Mean(UX_Original) == mean(UX_New)

If this null hypothesis is true, then the deployment of the product doesn't matter and either user interface can be used. However, this is independent for the data sample.

- Outcomes:

To analyse the data I have used Python and the SciPy library, using the following script, where the 2 arrays represent the totalled data for each of the categories that I have used to measure user experience and interaction as detailed in the table above.

```
import scipy.stats as stats

oldGUIDataResultSet = [40, 46, 36, 56]

newGUIDataResultSet = [61, 58, 62, 53]

ttest, pValue = stats.ttest_ind(oldGUIDataResultSet, newGUIDataResultSet)

print ('p-value =', pValue)
```

Result Output:

```
p-value = Ttest_indResult(statistic=-2.9192017967990469,
pvalue=0.026658992174787021
```

With this result, we can safely reject the NULL hypothesis of:

The mean(UX_Original) == mean(UX_New) then for the deployment of the product it doesn't matter what user interface that we use, however, this is independent for the data sample.

With a confidence level, greater than **99.9733410078%**

5.3 Implementation

- **Graphical User Interface**

For the GUI, I used the Java Swing library for which I have used I excess the JButton object, extending one of my classes to create a custom button that is fit for purpose for my product. The good thing about using the JButton is the ability to easily set text with one method which makes it simple to change on click with instant appearance change, the same concept applies to the changing of colour and font type settings.

A second point to make about my UI is that I use a combo / drop down box instead of an array of JButtons or a text box, this was pointed out to me by Dave Cohen during my interim presentation feedback that would be a good idea to implement in this way

- **Final Solution Design Patterns used**

- **MVC**

In my final product, I have used the Model, View, Controller pattern, used primarily to de-couple the model and view classes, using the controller to do the work in the middle, the controller and model being implemented through the singleton pattern.

- **Singleton Pattern**

I have used the Singleton pattern a few times through my project, namely the Model and Controller classes.

The purpose of using this design pattern is to restrict instantiation to just one single class and the general purpose of this is to centralise elements of the system, for example the Model component in MVC pattern is usually a Singleton implementation.

The instance created in the Singleton class has a private constructor which allows only one instance to be instantiated and that instance can be accessed from anywhere in the system, however as the constructor is private new instances cannot be created e.g.

Model:

```
/**
* private ctor for Singleton pattern
*/
private Model() { }
```

Controller:

```
/**
* private ctor for Singleton pattern
*/
private Controller() {

		model = Model.getInstance();
		view = new View(model.n);
		view.addClearButtonListener(new ClearListener());
		view.addCheckListener(new CheckListener());
		view.addHintListener(new HintListener());
		view.addActHint(new actHintListener());
		view.addPropListener(new PropListener());
		view.addSolveListener(new SolveListener());
		view.addSizeListener(new SizeListener());
		view.addGenListener(new GenListener());
		hintctr = 0;
	}
```

A downside of using the Singleton design pattern is that it limits the ability to subclass.

- **Test Driven Development**

I made the decision to not use Test Driven Development (TDD) all throughout my code as it wasn't strictly necessary for parts of the development, I have generally used it where I am testing for returns, for example for my preliminary recursive backtracking solver, I wasn't overly familiar with the concept so I have used TDD to aid with that part of the development, see code below:

```
public class PuzzleTest {
	PuzzleSolver ps;
	@Before
```

```
    public void setup() {
        ps = new PuzzleSolver();
    }
@Test
    public void testCheckValidFalse() {
        int[][] solgrid = new int[][]
        { { 1, 0, 9, 0 }, { 0, 9, 0, 1 }, { 9, 1, 0, 1 }, { 1, 0,
    1, 9 } };

        ps.validate(solgrid);
        assertEquals(ps.validate(solgrid), false);
    }
@Test
    public void testCheckValidTrue() {
        int[][] solgrid = new int[][]
        { { 1, 0, 1, 0 }, { 0, 1, 0, 1 }, { 0, 1, 0, 1 }, { 1, 0,
    1, 0 } };

        boolean check = ps.validate(solgrid);
        assertEquals(check, true);
    }
@Test
    public void testIsValid() {
        int[][] solgrid = new int[][]
        { { 1, 0, 1, 0 }, { 0, 1, 0, 1 }, { 0, 1, 0, 1 }, { 1, 0,
    1, 0 } };
        boolean check = ps.isValid(solgrid, 2, 3);
        assertEquals(check, true);
    }
@Test
    public void testAllIsValid() {
        int[][] solgrid = new int[][]
        { { 1, 0, 1, 0 }, { 0, 1, 0, 1 }, { 0, 1, 0, 1 }, { 1, 0,
    1, 0 } };
        boolean check = false;
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < 4; j++) {
                check = ps.isValid(solgrid, i, j);
                assertEquals(check, true);
            }}}
```

Class Structure Used:

- **GameButton**

Extends the JButton to essentially create my own custom button, which is important as for my implementation I use the additional state field integer values to work out the propagation and solving algorithms by converting the array of custom JButtons using the maintained valState integer

- **Model**

The Model is where most of my functionality is stored, including my validate method, the propagation solver, and the global *N* variable which defines the size of the grid.

- **Model2**

Contains the solve method and a variation of the validate method used in the solve method, which works by passing the aforementioned state array to this method, then I check if its reached the last row and column, if not it'll skip cells that are filled, then it enters the setting loop where it sets 0 then checks is valid then 1, whatever comes out true first is set, then later if it turns out to be wrong it backtracks where cells[I][j]=9, i.e. the reset statement.

- **Controller**

The Controller component interprets interactions and communicates these interactions with the View and Model components, which includes all the listeners for the components displayed on the view.

- **Grids**

My Grid library where grids get randomly loaded from. Includes several methods for selecting grids of different sizes and all grid are stored in ArrayLists or their respective sizes.

- **View**

The View quite simply is the UI and controls how system data gets presented to the user. My View consists of a Game Matrix which displays the Grid in its current state, It has a control panel which allows the used to execute the functional commands to play the game and it has a combo box to allow the user to change the size of the grid, I have used the BorderLayout style in the Swing library for Java to produce this element of the product.

5.4 Function Point Analysis (FPA)

Goals of Function Point Analysis (FPA);

1. Measure software by quantifying the functionality requested by and provided to the customer.
2. Measure software development and maintenance independently of technology used for implementation.
3. Measure software development and maintenance consistently across all projects and organizations.

(Alexander, No Date)

The first stage to calculating the FPA is for us to determine the count / measurement required. Here are the types of FPA counts.

1. Development Project FP Count
   - Measures the functions provided to the users with the first installation of the software being delivered.
2. Enhancement Project FP Count
   - Measures the modifications to an existing application.
3. Application FP Count
   - Measures the functionality provided to users in an existing application.

(Alexander, No Date)

Counting Process

Data Functions:

Internal Logical Files (ILF):

The count for the internal files, transactions, control and data.

Type	Count	#DETs
EO	solve, check, propagate, generate	4
EI	View, grid[][], controlPanel, generate, changeSize, clear, GameButton	7
EQ	Hint: JOptionPane.showMessageDialog, Completed Status: JOptionPane.showMessageDialog	2

RETS = [Grid Matrix, Control Panel]

DETS = 13, RETS = 2

RETS	**Data Element Types (DETs)**		
	1-19	20-50	51+
1	L	L	A
2 to 5	L	A	H
6 or more	A	H	H

(Alexander, No Date)

Complexity	**Points**
Low	7
Average	10
High	15

(Alexander, No Date)

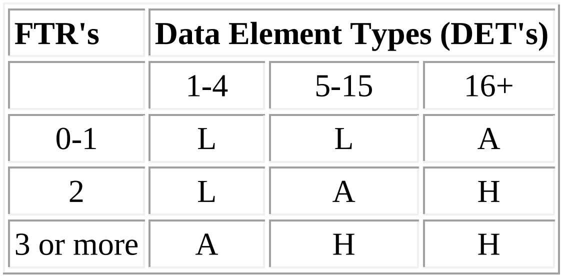
After undertaking this analysis, my application will be in the low section for DETs and the mid-section for RETs, thus concluding to Low complexity and a Function Point score of 7 for the ILFs, I believe this score to be this because I have had no real need to implement a database or any external services, to improve the function score, I could have built upon the idea that this application is a game and included a data base for user accounts and timed scoring, as well as this I could have implemented a download feature that presented the user with a “Puzzle of the day” downloaded from a website.

Transaction Functions:

External Interface (EI):

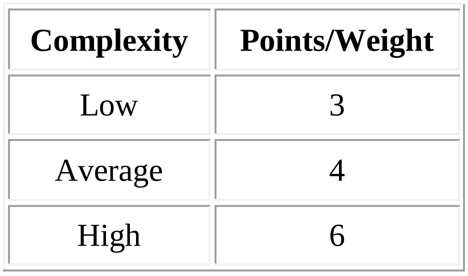
DETS = 9, FTR = 3

- View: DETS = 7, FTR = 2
- ClearListener: DETS = 1, FTR = 0
- Change Size: DETS = 1, FTR = 1



FTR's	Data Element Types (DET's)		
	1-4	5-15	16+
0-1	L	L	A
2	L	A	H
3 or more	A	H	H

(Alexander, No Date)



Complexity	Points/Weight
Low	3
Average	4
High	6

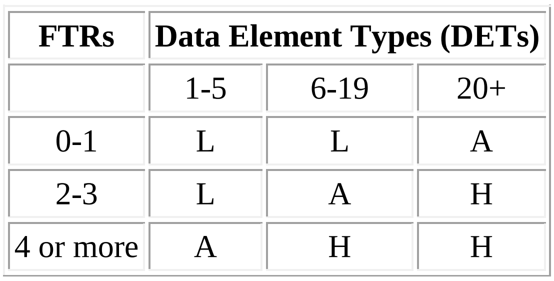
(Alexander, No Date)

High functionality and thus 6 function points per the FP matrix.

External Inquiry (EQ):

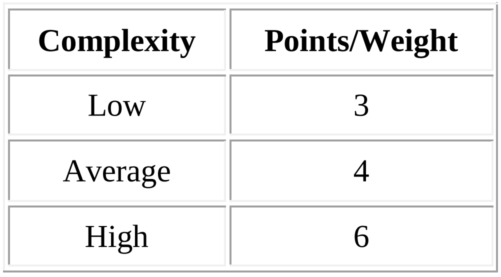
DETS = 2, FTR = 0

- Hints: DETS = 1, FTR = 0
- Completed Status: DETS = 1, FTR = 0



FTRs	Data Element Types (DETs)		
	1-5	6-19	20+
0-1	L	L	A
2-3	L	A	H
4 or more	A	H	H

(Alexander, No Date)



Complexity	Points/Weight
Low	3
Average	4
High	6

(Alexander, No Date)

Low functionality and thus 3 function points per the FP matrix.

External Output (EO):

DETS = 8, FTR = 3

- solve: DETS = 2, FTR = 1
- check: DETS = 2, FTR = 0
- propagate: DETS = 2, FTR = 1
- generate: DETS = 2, FTR = 1

FTR	**Data Element Types (DET)**		
	1-5	6-19	20+
0-1	L	L	A
2-3	L	A	H
4 or more	A	H	H

(Alexander, No Date)

Complexity	**Points/Weight**
Low	4
Average	5
High	7

(Alexander, No Date)

Upon counting this section and consulting the complexity matrix it is concluded that this subset is of Average complexity and acquires 5 points.

Count Analysis

Internal Logical Files:

Model | Complexity = Low = **7**

- hint1
- hint2
- grid_size

View | Complexity = Low = **7**

- ControlPanel
  - ControlBar
    1. Check
    2. Clear
    3. Propagate
    4. Generate
    5. Solve
- ChangeSize

GameButton | Complexity = Low = **7**

Total Internal Logical File Complexity Score = *21*

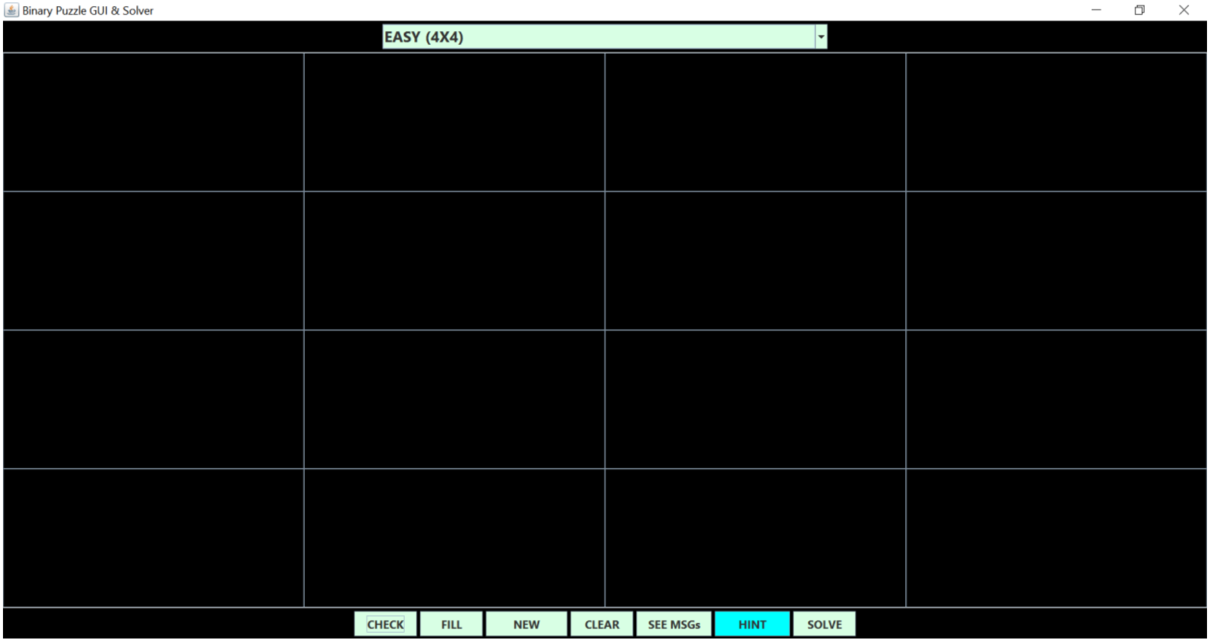
Transaction Functions:

EI

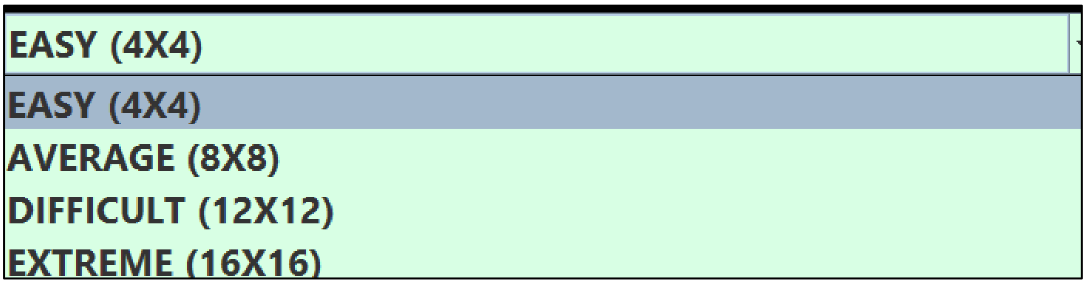
View: DETS = 7, FTR = 2

ClearListener: DETS = 1, FTR = 0

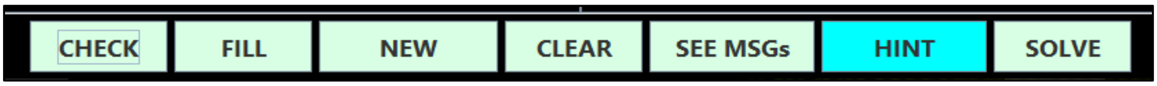
Change Size: DETS = 1, FTR = 1



Screenshot of entire view, shoing a grid that can be changed.



Change size Combo Box.



Control panel, including the Clear button.

Process	#DETS	FTR Names	#FTRs	Complexity	#FPs
Change grid in view	7	GameButton[][], View	2	A	4
Clear	1	n/a	0	L	3
Change Size	1	GameButton[][]	1	L	3
				Total	**10**

EO

DETS = 8, FTR = 3

- solve: DETS = 2, FTR = 1
- check: DETS = 2, FTR = 0
- propagate: DETS = 2, FTR = 1
- Hint: DETS = 2, FTR = 1

- generate: DETS = 2, FTR = 1

Process	#DETS	FTR Names	#FTRs	Complexity	#FPs
Solving process	2	View	1	L	4
Checking process	2	n/a	0	L	4
Propagation procedure	2	View	1	L	4
Generate/Load New Grid process	2	View	1	L	4
Hint System	2	View	1	L	4
				Total	**20**

EQ

DETS = 2, FTR = 0

- Hints: DETS = 1, FTR = 0
- Completed Status: DETS = 1, FTR = 0

Process	#DETS	#FTRs	Complexity	#FPs
Hint MSG System	1	0	L	3
Clear	1	0	L	3
			Total	**6**

Unadjusted Function Point Count

Function Type	Complexity	Multiplier	Line Item Sub-Total	Section Total
ILF	1 Low	x 7 =	7	
	0 Average	x 10 =	0	
	0 High	x 15 =	0	7
EIF	0 Low	x 5 =	0	
	0 Average	x 7 =	0	
	0 High	x 10 =	0	0
EI	2 Low	x 3 =	6	
	1 Average	x 4 =	4	
	0 High	x 6 =	0	10

EQ	2 Low	x 3 =	6	
	0 Average	x 4 =	0	
	0 High	x 6 =	0	6
EO	5 Low	x 4 =	20	
	0 Average	x 5 =	0	
	0 High	x 7 =	0	20
			Unadjusted Function Point Count:	**42**

Value Adjustment Factor (VAF)

- Data Communication: 0
- Distributed data processing: 0
- Performance: 4
- Heavily used configuration: 2
- Transaction rate: 0
- Online data entry: 0
- End user efficiency; 4
- Online update: 0
- Complex processing: 4
- Reusability: 4
- Installation ease: 5
- Operational ease: 4
- Multiple sites: 0
- Facilitate change: 3

TDI = 30

VAF = (TDI * 0.01) + 0.65 = 0.95

Adjusted Function Point Count

Adjusted FP Count = Unadjusted FP Count * VAF

= 42 * 0.95 = **39.9**

Formulas

Work Months = 300 / 42.85 = 7.001

Project Duration = 2.5 * 1.913 (cubeRoot(7.001)) = 4.7

FP Analysis Conclusion

The outcome of this formula in the function point analysis suggests that the project, based on the functionality that I have implemented should have taken 4.7 months working at maximum rate, however, which also suggests that having taken 7 months to complete the project that extra functionality should have been present, however, if you account for the time being allocated for writing reports / the interim presentation and review as well as undertaking coursework for other courses the 4.7 month estimation appears to be appropriate and the functionality implemented to be sufficient for the project time allocation.

5.4 Testing Phase

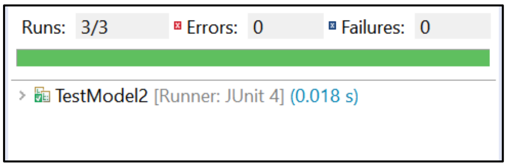
5.4.1 Unit Testing

TDD was not used entirely through out, however the elements of the program that have been written using TDD, mainly the solving and validating algorithms, all tests passed.

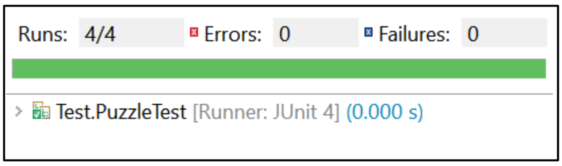
```
/**
 * Test method for {@link Model2#validate(int[][])}.
 */
@Test
public void testValidate() {
	m2.n= 4;
	Model2.validate(test);
	assertEquals(false, false);
}

/**
 * Test method for {@link Model2#solve(int, int, int[][])}.
 */
@Test
public void testSolve() {
	m2.n= 4;
	m2.solve(0, 0, test);
	assertEquals(true, m2.solve(0, 0, test));
}

/**
 * Test method for {@link Model2#displayGrid(int[][])}.
 */
@Test
public void testDisplayGrid() {
	m2.displayGrid(test);
}
```



```
Runs: 3/3    Errors: 0    Failures: 0
> TestModel2 [Runner: JUnit 4] (0.018 s)
```



```
Runs: 4/4    Errors: 0    Failures: 0
> Test.PuzzleTest [Runner: JUnit 4] (0.000 s)
```

5.4.2 Integration Testing

As can be seen from my plan, modules have been developed mostly independently throughout the project, after each module has been developed as essentially a proof console output program, I have immediately integrated that particular module in with the stable components of the code, this usually consists of taking one of my elements of functionality and integrating with the GUI which was stable from the start.

Test1: Combine Validation algorithm written with GUI.
Result: Pass
Notes: Converted proof of concept program into single method, added to listener and assigned button.
Steps: entered a correct grid and ran the validator

Test2: Combine Propagation algorithm written with GUI.
Result: Pass
Notes: Converted proof of concept program into single method, added to listener and assigned button.
Steps: entered a grid that contains known patterns as previously described, and ran the propagator.

Test3: Combine Solving algorithm written with GUI.
Result: Pass
Notes: Converted proof of concept program into single method, added to listener and assigned button.
Steps: entered a solvable grid and ran the solver

Test4: Add Message hints algorithm written with GUI.
Result: Pass w/ bug
Notes: Added 2 new strings to the model, and setting them with messages containing the equality counts of the incorrect rows / columns.
Steps: entered an incorrect grid and requested hint message (See MSGs)

Test5: Add Hints to program
Result: Pass
Notes: blindly solved the grid and set a cell to visible on-click.
Steps: loaded a new grid, requested hint, a blank cell was filled blue.

Test6: Load new grids
Result: Pass
Notes: Load pre-defined grid at random from library

Steps: opened program, clicked new, a grid was displayed and was solvable.

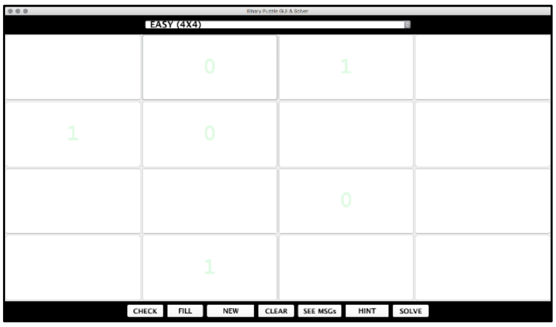
Bug List:

- Propagation can be repeatedly pressed and produce unsolvable grids.
- Validator has some instances where a blank is left and outputs solved, very rare
- Message hints appear several times, intended functionality is just once.

5.4.3 System Testing

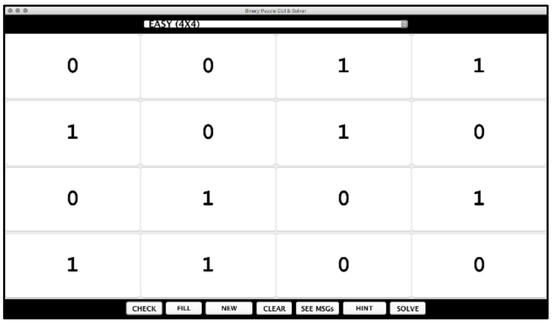
After testing through usage for system testing of the product I have found the following known issues:

- Propagation can be repeatedly pressed and produce unsolvable grids.
- Colour Scheme does not work on MacOS however, runs fine on Windows and standard Linux distributions.
- Validator has some instances where a blank is left and outputs solved, very rare
- If many operations have been carried out during an instance of the game, the program can start to fail by repeatedly flashing and not executing the requested command immediately, seems to get overloaded with instructions.
- Message hints appear several times, intended functionality is just once.



EASY (4X4)

	0	1	
1	0		
		0	
	1		



EASY (4X4)

0	0	1	1
1	0	1	0
0	1	0	1
1	1	0	0

- Running on MacOS ^

Chapter 6: Open & Closed Source Software / Patents (Report)

Open source software is software that is available to anyone and open to be modified or adapted by any developer. Whereas Closed source software is the opposite, rather; closed source software is owned by the author of the software.

6.1 General Advantages and Disadvantages of Open Source Software:

Advantages

Security issues will generally be found quicker than an internally developed system.

Software is examined and edited by a large team of developers this generally means that software will be made more secure than a system being developed internally by a smaller team of developers, obviously, this is situation dependent, but as a general rule this is the case.

Disadvantages

Although a cheap alternative to proprietary software, Open Source is widely available to developers which is good because the organisation can utilize software without actually paying to develop the same system, however, this means it is also available to attackers or anyone who may want to break the software who could potentially find or plant flaws within the system.

Conflictingly to advantage 1, the actual software vulnerabilities found may take time to actually be patched, also, this gives a window of opportunity for attackers to exploit these vulnerabilities.

Having software that is developed by a large community of developers may result in the manager or director of the organisation becoming complacent and not taking the still necessary security precautions required.

6.2 Advantages and Disadvantages of Closed Source Software:

Advantages

An advantage for closed source software is that the owners have complete access and control over the software and, thus, the potential vulnerabilities, this also includes the control over version releases, contrary to open source projects.

Proprietary software (closed source) is generally seen as more secure as the initial development of the software happens in a controlled environment with usually no external input, this makes the first source code release statistically more secure than that of an open source project.

By using closed source software there are other factors that also affect elements of the system in a positive way; firstly, the cost of the software, the expression “you get what you pay for” comes to mind, with proprietary software (closed source) usually costing a great deal you are more likely to get a system that is of higher quality and efficiency – and this includes the security of the system! Also, the ability, or non-ability, to edit the source code directly ensures the security of the software as it would have had to be thoroughly tested before release.

Disadvantages

A potential disadvantage to using closed source software would be that there is a smaller set of abilities within the development team compared to open source and this means that within the development team they need to be very competent with writing efficient, reusable and secure software, not only that, they will need to also be competent with analysing and fixing potential bugs within the developed software after first release or during testing.

As mentioned previously, the cost of obtaining well developed, secure systems can be an issue for some organisations and for organisation that handle valuable information it is essential that the systems they have in place are well secured and this comes at a cost.

6.3 Can my program can be open-source and patent-free?

Yes, it can be open source, if I were to store my source code in a public GitHub account, I have not plagiarised any code so wouldn’t be violating any legal or ethical rules

The Binary Puzzle is a Known problem, and the idea has been around for a long time and there are many different implementations of the game including binarypuzzle.com, binarypuzzle.nl and the Takuzu game on the Microsoft Windows Store. So to conclude, it is not patentable.

Chapter 7: Professional Issues

As with any Software Engineering project it is imperative to conduct all undertaken work properly and adhere to standardized regulations as well as internal organisational regulations regarding the code you produce.

If we refer to the ACM Software Engineering Code of Ethics and Professional Practice, it states the following:

> *1. PUBLIC - Software engineers shall act consistently with the public interest.*
>
> *2. CLIENT AND EMPLOYER - Software engineers shall act in a manner that is in the best interests of their client and employer consistent with the public interest.*
>
> *3. PRODUCT - Software engineers shall ensure that their products and related modifications meet the highest professional standards possible.*
>
> *4. JUDGMENT - Software engineers shall maintain integrity and independence in their professional judgment.*
>
> *5. MANAGEMENT - Software engineering managers and leaders shall subscribe to and promote an ethical approach to the management of software development and maintenance.*
>
> *6. PROFESSION - Software engineers shall advance the integrity and reputation of the profession consistent with the public interest.*
>
> *7. COLLEAGUES - Software engineers shall be fair to and supportive of their colleagues.*
>
> *8. SELF - Software engineers shall participate in lifelong learning regarding the practice of their profession and shall promote an ethical approach to the practice of the profession.*

(Acm.org, 2017)

The most relevant professional issue to my project relate to the plagiarism of code, obviously, an unethical method of producing software however, the code is designed to be reused and if part of an overall system has been taken directly from another source then it should be properly referenced / cited in the source, so for example in java that is done using the @author tag. E.g. @author -Author name

```
/**
 * @author Bradley Gauntlett
 */
```

This problem does not only apply to source code independently, it also related to other software components too such as graphical objects, flat files, databases and any other content that has been produced by someone else.

The second ethical issue that we must consider for the type of application that I am developing is that, in essence, it is intended to be played as a game for the user, there are many issues with games is the addictive quality they possess, Mostly, this is the case with games that use a level based system with no defined end to the game to contest this I have implemented a feature that changes the colour of the screen when a grid gets completed clearly defining the end of that routine also which is good as with my application each puzzle will be solvable and also the main part of the application, the solving algorithm, will always be active for the user to apply acting as a give-up feature. However, I do hope to implement a difficulty feature I will not use a level 'unlocking' system which is proven to be one of the most addictive qualities of a game, according to (Conrad, 2017).

Chapter 8: Self Evaluation

8.1 Organisation

Planning

During the end summer period before commencing the project I worked on producing a plan to follow throughout the project. Mostly I have tried to work alongside my defined guidelines for when to produce the work contributing to my overall project however, obviously throughout the course of the year certain circumstances change and as influenced by the Agile methodology of producing software task needed to occasionally be re-organised to prioritise other tasks of higher importance, also as response to my feedback I altered my plan to switch the priority of 2 tasks, in hindsight, this was an error of judgement on my part –

Initial Feedback:

"This plan has most expected features, but there are some clear weaknesses in it. In particular, it is evident that the plan has been written without a sufficiently clear understanding of some of the work that will go the algorithms and the program. As an example of this, the plan proposes to finish the proof of concept program for solving a grid before the program that verifies whether a proposed solution is correct. In fact, the former program will need to build upon the solutions for the latter program. Similarly, the risks and mitigations are mostly generic, e.g., getting behind / having problems with the code, and do not show a clear understanding of problem-specific obstacles or specific expected difficulties."

Meetings

I have regularly attended meetings with my project supervisor (Magnus Wahlström) to present current progress and discuss specific problems that I have faced, these meetings regularly occurred at a rate of once every 2 weeks.

8.2 Practice

Design

I believe my software has reasonable design and have conducted basic HCI tests and statistical analysis for proof of design proposals, all documented in Chapter 5: Software Engineering.

Feedback:

The feedback for my original (Proof of Concept) design: *"The GUI is functional and compactly written*" confirmed that my initial design was acceptable.

However, I wanted to build upon this because; firstly, to improve my GUI programming ability because I am starting a job as Front-end developer working with UX designers after I graduate which is partly why I have conducted the amount of work that I have in relation to the design of my product, secondly; I believe the concept (game logic) of this project is fairly simple to the end user (even though the implementation isn't) so a pleasant and easy to use GUI was something that felt was important.

Code

For the first stage of the project (Interim Review) I felt some aspects of the project went well however, I faced some large problems, Firstly the development of the GUI I found fairly simple so this was produced almost fully, the grid validator I produced almost fully aside from problems with reuse, and the solver unfortunately after much trying was not able to successfully implement the backtracking algorithm for solving, instead, produced an iterative approach that used known patterns and filled empty cells respectively of the pre-specified patterns.

Feedback:

"Three proof of concept programs are provided: A GUI, a "checker" (that checks whether a randomly filled grid is correct), and a "partial solver". The GUI is functional and compactly written. The checker does its job, although it is not written in a "maximally reusable" manner (instead it's a single giant main function). The "partial solver" contains code to do a single pass of the "piggy in the middle" and "finding pairs" rules from the report. Unfortunately, an actual complete solver is missing, and the proof of concept programs contain no implementation of any backtracking-style algorithm. Also, a remark on coding style: Please don't rely on "try / catch" mechanisms in the normal operation of your code! The partial solver in particular should have been written so as to never trigger an out-of-bounds exception in the first place. So in summary, what's present is functional (if not too carefully designed), but a significant, central part of the project is still absent."

For the second part of the development I feel that I have made major improvements: I have taken on board the December feedback and implemented a more suitable 'product', I have used my code for my proof of concept programs and refined and improved the implementation, firstly, adjusting the specific loop variable to remove un-needed try-catch blocks (as specified in the feedback), I have improved on my design for the appearance of the product and I have managed to implement the backtracking solver feature that works with a reasonable level of efficiency (solving 12x12 grids in under 3 seconds and my program now loads new grids at random and displays to the user with locked cells so the user cannot edit the initial grid starting pattern. I feel this is significant progress since the interim review and I am quite pleased with the state of the project.

8.3 Achievements

I have gained a lot after undertaking this project namely; Re-enforcing my understanding with Swing and Java GUI development, research and report writing from the several reports that I have produced as part of this document, use of SVN, understanding and knowledge of recursion and in particular how the use of recursive backtracking can be used to solve a problem like this and finally presentation skills are something I (and many other Computer Science student seem to struggle with) and I found the interim presentation very useful in developing this skill, which I received an A grade for:

Feedback: *"This student presented his work on the Binary Puzzle. Excellent presentation. Very good style. Clear and well structured. There was not enough technical detail presented, but the student answered questions intelligently. Hence this is graded as a Very Good presentation."*

During a meeting with my project supervisor it was pointed out to me that when a grid is loaded into the GUI the cells do not lock, this was something I wasn't sure how to fix but worked out how to do it by breaking out of the switch statement in my GameButton class used to change the number shown in the cell on click.

8.4 Difficulties

I have had many difficulties during the course of this project, firstly with my planning phase I hadn't fully understood the intricacies of the project and thus made mistakes in this phase.

Secondly, my main difficulty, was implementing the recursive backtracking feature to my solver, after my interim report feedback I was instructed that I hadn't properly understood the main concepts concerning my project relating to recursion.

Feedback:

"It also contains a section on recursive algorithms, including a description of the backtracking algorithm. Unfortunately, the description of recursion is at a simple level, and the description of backtracking, while not incorrect, is not really concrete enough to be turned directly into an algorithm. There is also no connection made between the backtracking algorithm and how a computer can solve binary puzzles"

To fix this I undertook more reading and repetitively practiced the programming of this element of my project until it finally worked, which also helped my produce a better section in my report that I was told through verbal feedback was a significant improvement after the draft hand-in in February.

Chapter 9: Planning and Timescales

9.1 Initial Work outline (Taken from original plan)

Term 1:

	Week	1	2	3	4	5	6	7	8	9	10	11	12
Task	-	-	-	-	-	-	-	-	-	-	-		
(1)	-						X	X	X				
(2)	-				X	X	X						
(3)	-								X				
(4)	-		X	X		X	X	X					
(5)	-			X	X	X							

- Proof of concept programs which tests if a grid is correct and complete (1)

Due: 25/11/16

For this particular program, I was thinking of rather than trying to implement the GUI to work with the algorithm I would simply use a 2-dimensional integer array and get the algorithm to check grids to work with that, just for the proof of concept that is then for the actual project deliverable it will be a lot easier to modify the algorithm to rather than perform the operations on integers it will check the state of elements within the grid on the GUI.

- Proof of concept program which gives a solution to a grid (2)

Due: 11/11/16

Again, same as above; I will initially for the proof of concept program use a 2-dimensional integer array to store the 1s and 0s then the algorithm, in this case to give correct solutions, can be applied to this then later the code can be refactored to work with the GUI.

- Proof of concept GUI design (3)

Due: 25/11/16

For this I have thought of a few ways in which it could be implemented and currently I am thinking of using a 2D array of buttons, a separate class will be used for this which will maintain a state of what the button is currently set to and on the listener will be set as %= 3 so each button can have a different text set to it and will be blank, 0 or 1, this will make it nice to convert the proof of concept program, I will also use a switch statement to allow the user to change any of the buttons by clicking them. Also using this will make it easier for the algorithm to change it by simply setting a particular button object's state number to a different one where necessary.

Also for this proof of concept design I will try to utilise the skills that I acquired from last year's HCI course to help me eventually develop a 'good' GUI for the final deliverable, and for the proof of concept will utilise more basic HCI skills to produce a satisfactory one with the time frame that I have. The project will be programmed in Java as it is the language that I am most competent in, however, my experience with Swing or JavaFX is limited so I will be using both books and web resources to aid me with producing the best design possible.

- Report on design patterns and code organisation (4)

Due: 21/10/16

When it comes to writing the report on Design patterns and Code organisation I will examine the project more closely to choose design patterns and will make use of packages for code organisation.

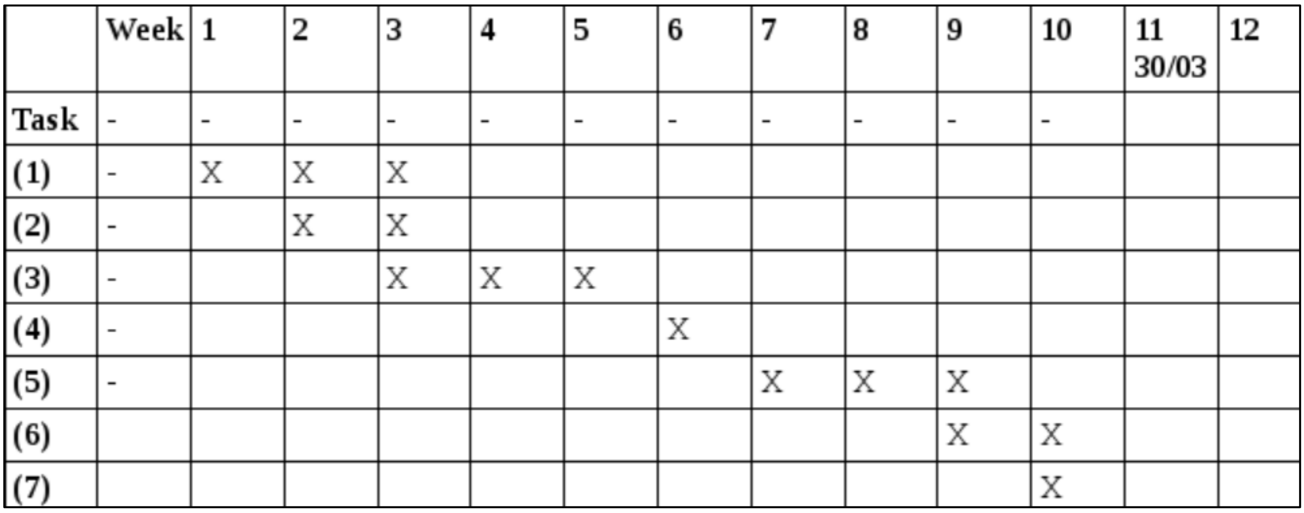
For the design patterns, I will revise how we used them for the team project last term and improve my more theoretical knowledge of design patterns from my notes from our software engineering module last year, I will conduct all this in the first few days of the allocated time slot for report detailed on the timeline.

- Report on Backtracking and Recursion (5)

Due; 04/11/16

I plan to write a report about backtracking and recursion and how I can use them in my solution, to do this I am using both web resources and books as detailed in my bibliography.

Term 2:



	Week	1	2	3	4	5	6	7	8	9	10	11 30/03	12
Task	-	-	-	-	-	-	-	-	-	-	-		
(1)	-	X	X	X									
(2)	-		X	X									
(3)	-			X	X	X							
(4)	-						X						
(5)	-							X	X	X			
(6)										X	X		
(7)											X		

- Improve algorithms and get working in conjunction with GUI (1)

  Take work from the proof of concept programs completed in first term and work to get them working in conjunction with each other. To do this I will have to move the code in the proof of concept algorithms to work on an event for a JButton in the GUI and also edit the code so that it will reference the state of the grid element rather than just an integer.

- Add features for resizing grids, filling by user input and apply the checking algorithm (2)

To do this I will add 2 text boxes and a submit button which will change the values in the code that are used to build the grid.

- Add feature to generate 1 solution only grids (3)

This is an aspect of the project that I am yet to work out and understanding of, I will be utilising my free time over the Christmas break to do some independent research into this and hopefully plan on how to accomplish this feature.

- Software Engineering process report section (4)

Throughout the project, I will be using Software Engineering techniques that I have acquired from last year's modules in Software Engineering and the Team project, I will make sure to use TDD appropriately, good use of code repositories and version control software as well as including work from the preliminary report done in term

- Backtracking and recursion second report section (5)

Final report section showing how I used backtracking and recursion in my solution to solve the problem, I will have done preliminary work on this in the first term report as well.

- Data structures report section (6)

In this report, I will write about all the various data structures that I am going to use throughout the program, and justify why I have used them, for instance I am planning on using a 2D array for the grid because it will display a grid like what the requirements state and they are easily manipulated.

- Open source report section (7)

The report will discuss if the program can be open-source and patent-free.

9.1.1 Reflective action to original work plan

After the feedback that I received on my proposed dates for undertaking project work it was pointed out to me that some of my deadlines needed adjusting for example completing research via the reports in line with the development of related techniques also the order in which I was carrying out the development, hopefully from my work diary you can see I made the appropriate adjustments.

Bibliography

1. **Binarypuzzle.nl. (2016)**. Binary Puzzle Solver. [online] Available at: https://binarypuzzle.nl [Accessed 29 Sep. 2016]
    a. Used to help me with understanding the work entailed with the proof of concept programs

2. **Binarypuzzle.com. (2016)**. Binary puzzles, solve online or print - BinaryPuzzle.com. [online] Available at: http://www.binarypuzzle.com/ [Accessed 29 Sep. 2016].
    a. Used to help me with understanding the work entailed with the proof of concept programs

3. **Drozdek, A. (2001)**. Data structures and algorithms in Java. Pacific Grove, CA: Brooks/Cole Pub. Java
    a. Used to give me an understanding of backtracking and help write the report on Back tracking and recursion in both the first term and second terms.

4. S**ommerville, I. (2016).** Software engineering. Harlow: Pearson. -
    a. Used to refresh myself on good software practices as well as write the report on Design patterns and Code Organisation, also the second term report section on the Software Engineering process.

5. **Projects.cs.rhul.ac.uk. (2016)** [online] Available at: https://projects.cs.rhul.ac.uk/List.php?PROJECT-TYPE=Full [Accessed 27 Nov. 2016].

6. **David G.Sullivan PhD, (n.d.).** Recursion and Recursive Backtracking. [online] www.fas.harvard.edu. Available at: http://www.fas.harvard.edu/~cscie119/lectures/recursion.pdf [Accessed 2 Jan. 2017].

7. **Alexander, A. (n.d.).** An introduction (tutorial) to Function Point Analysis. [online] Alvinalexander.com. Available at: http://alvinalexander.com/FunctionPoints/FunctionPoints.shtml [Accessed 25 Feb. 2017].

8. **Acm.org. (2017).** Software Engineering Code of Ethics and Professional Practice — Association for Computing Machinery. [online] Available at: http://www.acm.org/about/se-code [Accessed 11 Mar. 2017].

9. **Conrad, B. Dr (n.d).** 15 Reasons & Theories on Why Video Games Are Addictive - TechAddiction. [online] Techaddiction.ca. Available at:

http://www.techaddiction.ca/why_are_video_games_addictive.html [Accessed 11 Mar. 2017].

10. **Codejava.net. (2017).** JComboBox basic tutorial and examples. [online] Available at: http://www.codejava.net/java-se/swing/jcombobox-basic-tutorial-and-examples [Accessed 12 Mar. 2017].

11. **P. Utomo and R. Pellikaan**, On The Rate of Constrained Arrays. Proc. IndoMS International, Conference on Mathematics and Its Applications (IICMA) 2015, 3 November 2015, http://www.win.tue.nl/~ruudp/paper/75.pdf

12. **Sedgewick, R. and Wayne, K. (2016).** Queens.java. [online] Introcs.cs.princeton.edu. Available at: http://introcs.cs.princeton.edu/java/23recursion/Queens.java.html [Accessed 28 Mar. 2017

13. **Peter Hornsby, (2010)** Hierarchical Task Analysis :: UXmatters. [online] Uxmatters.com. Available at: http://www.uxmatters.com/mt/archives/2010/02/hierarchical-task-analysis.php [Accessed 30 Mar. 2017].

Appendix 1: Work Diary

Date: June 2016
Task: GUI Design Proof of concept
Description of Achievements: During the summer, I started some for the work for my GUI design, during this month I successfully programmed a draft GUI for my program using the Swing library in Java.
Difficulties faced: During the team project, last year we used WindowBuilder for our GUIs, for this project I didn't want to have generated code like this method does, so wanted to code the GUI myself, however after not using Swing since December 2014 I needed to use tutorials and a Java book for reference.

Date: September 2016
Task: Project Plan
Description of Achievements: Wrote a plan including timescales of when I was going to complete work, found various readings and web resources in advance to aid me during the project.
Difficulties faced: Finding resources specifically for Binary puzzles that weren't solely web based.

Date: October 2016
Task: Set up SVN repository for Project
Description of Achievements: Shared project with my SVN repository and provided supervisor (Magnus Wahlström) with the repository link.

Date: October -November 2016
Task: Properly started working on the grid verifier
Description of Achievements: Successfully produced an iterative solution that verifies if a grid is complete or incomplete, as a proof of concept used a 2-dimensional array rather than the GUI.
Difficulties faced: In early trials of the algorithm implementation it would not detect all conditions causing an incomplete solution and would output complete, similarly it would output incomplete when a complete grid was generated.

Date: October 2016 – November 2016
Task: Started writing reports
Description of Achievements: initially wrote about a selection of potential design patterns that I could use and started reading up on recursive backtracking.

Date: October 2016 - November 2016
Task: Report writing

Description of Achievements: Started writing properly on Recursion starting with the basic concepts and continuing researching backtracking
Difficulties faced: Struggling to understand the recursive backtracking concept

Date: November 2016 (early)
Task: Properly started working on the grid solver
Description of Achievements: Started to work on the solver, initially using propagation techniques for known blank cells, for example, having a blank between 2 1s the blank must be set to a 0.
Difficulties faced: won't detect all scenarios where a blank must be a 1 or 0 based on neighbouring cells.

Date: November 2016 (early-mid)
Task: Interim Report
Description of Achievements: Combined work done for each report (both still unfinished) and formatted into the project report outline
Difficulties faced: Not using MS Word or Latex so having to use Google Docs which caused formatting issues, now solved.

Date: November 2016 (mid)
Task: Grid solver
Description: Still working on solver, work proving difficult. Suggested by supervisor to try a different recursive backtracking problem like N-Queens for proof of concept.

Date: November 2016 (mid)
Task: Reports
Description of Achievements: Recursion and Backtracking report nearly complete.

Date: November 2016 (mid-late)
Task: Reports
Description of Achievements: Design patterns and Code organisation report nearly complete.

Date: December 2016 (mid)
Task: GUI and algorithm integration
Description of Achievements: worked on GUI adding control bar, and adding functionality to the check grid button.

Date: December 2016 (mid)
Task: GUI
Description of Achievements: worked on GUI adding control bar buttons and listeners, and adding functionality to the clear grid button.

Date: December 2016 (late)
Task: GUI and algorithm integration
Description of Achievements: worked on GUI adding control bar, and adding functionality to the partial / propagation solve grid button.

Date: Jan 2017 (early)
Task: GUI
Description of Achievements: improved colour scheme for GUI.

Date: Jan 2017 (early-mid)
Task: GUI and hints
Description of Achievements: added button to the control bar and added functionality for the program to give hints to the user, this feature works occasionally, however, sometimes gives wrong messages to the user.

Date: Jan 2017 (early-mid)
Task: Algorithm (solver)
Description of Achievements: worked on implementing an algorithm that used an iterative approach, whilst still using backtracking

Date: Jan 2017 (mid)
Task: Reports
Description of Achievements: started formatting final report, taking work from interim and moving across.

Date: Jan (late)
Task: Reports
Description of Achievements: upon meeting with supervisor, decided to write section on the N-Queens algorithm, with full worked example of the algorithm.

Date: Jan 2017 (late)
Task: Solving algorithm
Description of Achievements: after being advised that the above proposed solution would not be feasible, started doing more research into the recursive solution with backtracking.

Date: Feb 2017 (early)
Task: Solving algorithm
Description of Achievements: produced a solution that uses recursion and backtracking, however, as I was using work I did when researching the N-Queens problem there are slight flaws in my implementation, although the core functionality and concepts are present.

Date: Feb 17 (mid 12/02/17)
Task: Reports
Description of Achievements: Further work on report, as I am aiming to get as many sections complete before the draft hand in on the 17th of February.

Date: Mar 17 (early)
Task: Solving Algorithm
Description of Achievements: Solver fully works and solves 12x12 in approx. 3 seconds.

Date: Mar 17 (early)

Task: Reports
Description of Achievements: Continued work on report

Date: Mar 17 (early)
Task: Loading grids
Description of Achievements: Back to programming, I have managed to get my program to load new grids to the user at random from my grid library.

Date: Mar 17 (mid)
Task: Reports
Description of Achievements: Continued work on report

Date: Mar 17 (mid-late)
Task: Reports
Description of Achievements: Continued work on report – Software eng process, Recursion / Backtracking

Date: Mar 17 (mid-late)
Task: Reports
Description of Achievements: Continued work on report – Open Source, Lit Review,

Date: Mar 17 (late)
Task: Reports
Description of Achievements: Continued work on report – FPA

Date: Mar 17 (Date of submission)
Task: Reports
Description of Achievements: Tidied up report and proof read and prepared.

Date: Mar 17 (Date of submission)
Task: Code
Description of Achievements: Checked version was still stable before submission and prepared.

Appendix 2: User Manual

Installation and Running:

Windows:

Ensure that you have Java installed

Open the .JAR file by double-clicking

Linux:

Ensure that you have Java installed

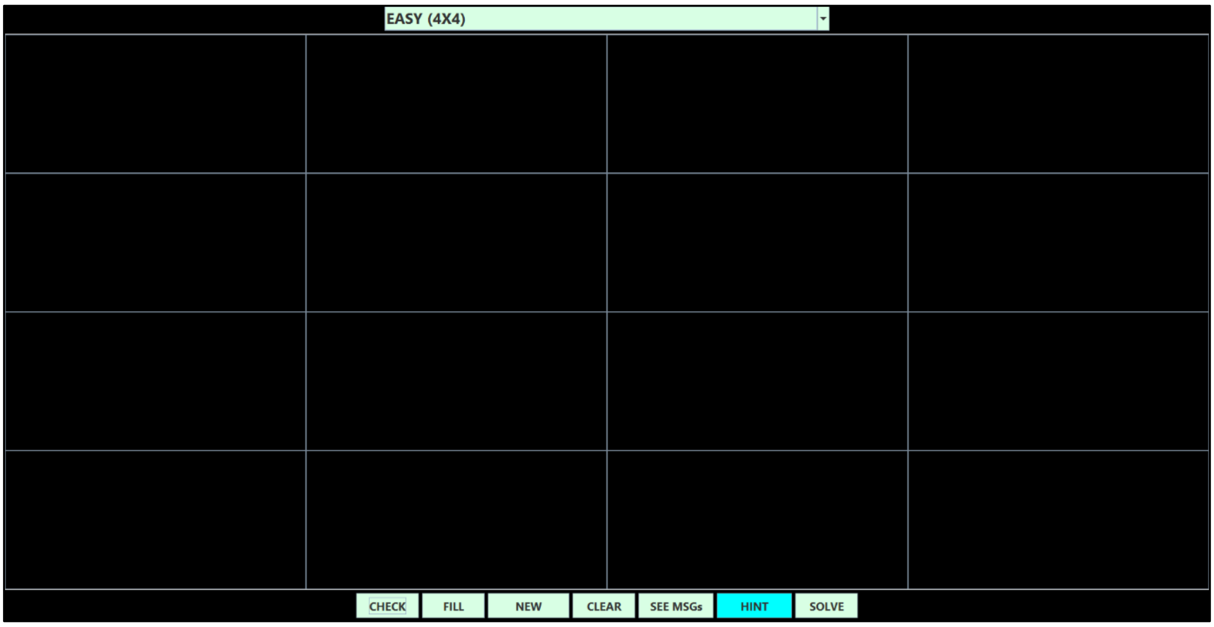
Open the .JAR file by opening a new Terminal window and navicating to the file location

Run by entering the following command:

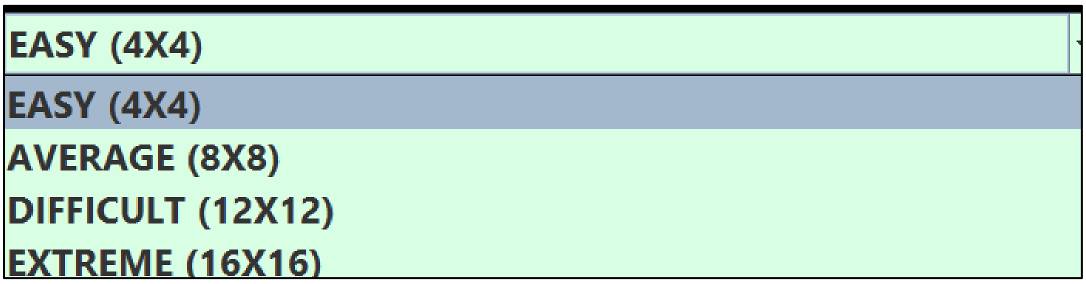
Java -jar <filename>.jar

Usability:

Once opened you will see the following screen:

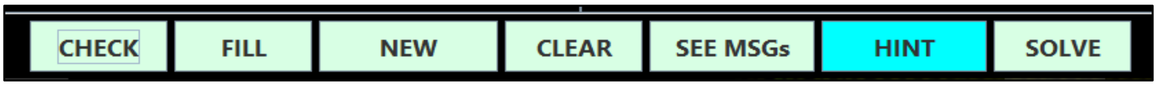


Use Dropdown box at the top to set the size / difficulty of the grid.



Once set, look at the control panel at the bottom of the GUI.

This encapsulates all the game functionality:



Check – *checks if the currently displayed grid is a correct solution or not*, it will turn green if it is and red if not.

Fill – *the Fill function is an extra function that is included that runs the propagation algorithm and will fill blanks that are in known patterns.*

New – *Will load and display a new grid to the user at random from the grid library (click this first).*

Clear– *Will clear all values stored in the grid, allowing the user to re-start.*

See MSGs – *after the check command has been run it will store an indicator as to where the grid is wrong, if wrong at all,* ***be aware*** *though the longer the game runs it appears that the number of msgs increases and this can be annoying for you as the user.*

Hint– *The Hint function will allow the user to peek at one black cell and see the correct value for that cell, the cell will turn Cyan and can only be used once per game instance.*

Solve – *The solve function will take the currently displayed grid and run it through the solving algorithm and display back the solved grid to the user, if when you click the button it does nothing, this is an indicator that the currently displayed grid is unsolvable.*