

# Using Gamification to Create an Educational Tool based on Natural Deduction Logic

Submitted by: James Farthing

## **COPYRIGHT**

Attention is drawn to the fact that copyright of this dissertation rests with its author. The Intellectual Property Rights of the products produced as part of the project belong to the author unless otherwise specified below, in accordance with the University of Bath's policy on intellectual property (see <http://www.bath.ac.uk/ordinances/22.pdf>).

This copy of the dissertation has been supplied on condition that anyone who consults it is understood to recognise that its copyright rests with its author and that no quotation from the dissertation and no information derived from it may be published without the prior written consent of the author.

## **Declaration**

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Signed:

### **Abstract**

Your abstract should appear here. An abstract is a short paragraph describing the aims of the project, what was achieved and what contributions it has made.

# Using Gamification to Create an Educational Tool based on Natural Deduction Logic

James Farthing

Bachelor of Science in Computer Science and Mathematics with  
Honours  
The University of Bath  
April 2016

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Literature Review</b>	<b>4</b>
2.1	Project summary . . . . .	4
2.2	Gamification . . . . .	4
2.2.1	Uses . . . . .	5
2.2.2	Advantages . . . . .	6
2.2.3	Current Educational Games . . . . .	8
2.3	Education . . . . .	9
2.3.1	Different learning styles . . . . .	10
2.3.2	Current Natural Deduction Educational Resources . .	11
2.3.3	Online Natural Deduction Resources . . . . .	12
2.4	Natural Deduction . . . . .	13
2.4.1	History . . . . .	13
2.4.2	Why Natural Deduction is Suitable to Make a Game for	14
2.4.3	Mathematics . . . . .	15
2.5	Concluding remarks . . . . .	18
<b>3</b>	<b>Requirements Analysis and Requirements Specification</b>	<b>20</b>
3.1	Target Audience . . . . .	20
3.2	Project Goals and Targets . . . . .	21
3.3	Software Requirements . . . . .	21
3.3.1	Functional Requirements . . . . .	22
3.3.2	Non Functional Requirements . . . . .	23
3.4	Hardware Requirements . . . . .	24
3.5	Programming Language . . . . .	25
3.6	Project Limitations and Changes to Existing Systems . . . .	26
<b>4</b>	<b>Design</b>	<b>29</b>
4.1	Data Design . . . . .	29
4.2	Data Flow Diagram . . . . .	31
4.3	Architecture Design . . . . .	33
4.4	Interface Design . . . . .	35

4.4.1	Design Principles . . . . .	35
4.4.2	User Designs . . . . .	36
<b>5</b>	<b>Implementation</b>	<b>40</b>
5.1	Beginning the project . . . . .	40
5.2	Drag and Drop . . . . .	41
5.2.1	Original Colour System . . . . .	42
5.2.2	Dots System . . . . .	43
5.3	Data Structures . . . . .	45
5.4	Unification Algorithm . . . . .	46
5.5	Drawing and rendering algorithms . . . . .	47
5.6	Levels and how the Software was turned into a Game . . . . .	48
<b>6</b>	<b>System Testing</b>	<b>50</b>
<b>7</b>	<b>Results</b>	<b>51</b>
<b>8</b>	<b>Conclusion</b>	<b>52</b>

## Chapter 1

# Introduction

**Hello world!**

Hello, here is some text without a meaning. This...

## Chapter 2

# Literature Review

### 2.1 Project summary

The idea behind this project is to take the Natural Deduction logic in mathematics and try to make it easier for older secondary school and university students to understand. In my opinion, an effective way of doing this is through gamification (making a game of it) which will hopefully engage students in a new and different way. As education is not usually presented in this way, research will be undertaken to see how gamifying educational concepts can be helpful for the user. Whilst primarily a software project, where a game will be created, research needs to be done to find the most effective way of communicating via games. Research also needs to be done to fully understand the topic of Natural Deduction, so the concepts can be presented correctly in the game.

### 2.2 Gamification

Gamification is the idea of making a task or situation use game mechanics that does not naturally use this context. It is theoretically possible to make many tasks into a game using game-like concepts and this is why gamification has become business in a wide variety of topic areas Muntean (2011).

It is important to understand the characteristics of a game to fully understand how a task can take on game like properties. A game is defined by a system in which players engage in abstract challenge, defined by rules, interactivity, and feedback, that results in a quantifiable outcome often eliciting an emotional reaction. Koster (2013) Gamification of this in my educational context would be to set rules for a user to follow, challenge them with

different Natural Deduction concepts and give them constructive, instant feedback in the task which is not usually presented in this format.

### 2.2.1 Uses

Gamification has been used in a variety of different situations in a number of industries and environments. Whilst it is important to focus on how gamification has been adapted and used for educational purposes, it has to be noted which other areas have used gamification to their advantage and what key concepts they have used to make gaming a successful platform for their product or service.

Marketing by companies has been a successful way of using gamification Zichermann & Cunningham (2011). They have been able to engage users in new ways by providing apps and websites with games and activities on and consequently consumers are spending more time around that brand. It is important to harness this way of being able to interest people using gaming principles when trying to teach my educational concepts. I think that people can enjoy and learn more effectively if they are engaged in the thing they are doing. Games are a good way of achieving this.

Industry has also used gamification. Studies have investigated whether gaming concepts can increase motivation and morale in the workforce, leading to higher productivity. An article in *Frontiers in psychology* was positive stated in conclusion that there's "emerging base of evidence that suggests gamification as a promising strategy for promoting loyalty, productivity, and wellbeing in the workplace" Oprescu et al. (2014).

Whilst education and industry are two different things, there are qualities and skills that are required in both areas. Increasing motivation to learn a particular mathematical formula, for example, could be very beneficial for some students. A higher rate of productivity would also be advantageous, as students often have many tasks ongoing at any one time. An increase in these skills would overall increase the enjoyment of learning and reinforce understanding so students can get a better handle on the key ideas.

Gamification has also been embraced by the educational sector. This is by far the most important sector for this project. It is important to understand how gamification has helped improve education and helped student to learn. This is the core to the project, that gamification of educational topics will aid students to learn the topics in an easier way. Above all else, helping people being able to learn the Natural Deduction logic in an easier way is the main goal of this project.



### 2.2.2 Advantages

There is an abundance of educational games on the Internet. Educational games have many benefits that will help students learn. Instant feedback, social aspects, competitiveness, games being fun and giving the user hints and tips are all advantages that make games a better way of learning than other formats. These are explained in detail below.

#### Instant Feedback

Games offer instant feedback; the user can immediately know when they have done something right or wrong. This is a big advantage with games, it can offer advice and tips much quicker than a teacher can. My game needs to use this feature to its advantage, because giving someone instant feedback can be rewarding and be much more helpful than waiting weeks for feedback, for example, after handing in a question sheet.

Research has been undertaken to examine whether giving students instant feedback has improved learning performance Wu et al. (2012). This study, which was published in the British Journal of Educational Technology, found that the enhanced way of presenting their concept maps which gave the students instant feedback "significantly improved the learning achievements of the students". They recognised that giving instant feedback meant that the topic was fresh in the students' minds and therefore could organise, collate and adapt to the information that was given in the feedback.

The paper also suggested that students which used the new system "gave significantly higher ratings" than those who used the standard approach. This questionnaire was asking about learning attitudes towards the course that they were undertaking. From this information, it can be deduced that the students who were questioned seemed to be enjoying their course more due to the enhanced system that was introduced in this study. It is important to recognise that instant feedback was not the only improvement to the enhanced system they were using in the study, but this was a key part of the new system, so must have contributed at least partly to the students' improved learning performance and improved learning attitudes.

Instant feedback can be key motivation for the project, giving that it can improve both student enjoyment and performance. As mentioned by Dempsey et al, instant feedback can more effective than delayed feedback when using "classroom quizzes and (other) materials" Dempsey et al. (1993). He goes on to conclude that "immediate feedback should be prescribed unless the feedback is delayed systematically for a specialised purpose".

It is crucial to bear this in mind when creating the game that giving a

user instant feedback can be a powerful tool. Because of this, mathematical notation needs to be included so that students can start to learn the Natural Deduction logic concepts. If the game was too abstract, with no relation to the mathematical representation, part of the learning process would be lost.

One way of offering this constructive feedback would be to offer a scoring system. You could get a certain number of points for getting parts of a question correct and bonus points for flawless levels completed. This would instantly tell the player of the game whether they were doing well or not. Players could repeat plays of the game to get higher scores which would increase understanding. This would be because doing a repetitive action multiple times reinforces your learning.

### **Social and Competitive Aspects**

Another advantage to Gamification is the ability to offer a social or competitive aspect of your subject matter. This could be through online leaderboards displaying top scores, multiplayer aspects or social media connectivity. Keeping fellow users of the game connected in some way can be important for increasing enjoyment of gameplay and also the amount of time they spend playing the game.

In the current age where 90% of 16-24 year olds in the UK have smart phones, and being used for at least two hours a day, it can be said that people find that communicating via online methods important Ofcom (2015). In the same article, it is said that "around half (49%) of young people aged 18-24 check their phones within five minutes of waking up". This just backs up my point that social interaction and social media via online devices is in the forefront of people's minds throughout their lives. By incorporating this into my game, by offering some kind of social aspect, people will enjoy it more.

A study undertaken which has looked into how to create engaging games in the health care sector discovered that communication was important for 13 to 16 year olds, which was their target market Suhonen et al. (2008). It also discovered that being in touch with others and interacting with them while playing games was also important to them. The report states that "The company of friends motivates to play games and makes gaming fun". These are two massively important points that should be not underestimated when creating my own game. Being able to socialize whilst playing a game may increase participation and fun. Students can also talk to each and discuss ideas about what is the correct way to go about solving a problem.

## Fun

Games being fun is one of the main reasons that people want to make non-gaming concepts into games. If people will enjoy what they are doing more when they are playing games, then games are a viable solution for all sorts of media, including educational resources. A game being fun will capture an audience more and they will be more likely to play again because of the enjoyment they have got whilst playing it.

Research published by the UKIE stated that the UK games market is worth £3.944 billion in 2014 UKIE (2014). This much money would not have been spent on games if the consumer did not enjoy playing games. As mentioned by Rosemary Garris, Robert Ahlers and James E. Driskell, users make a clear cut choice to whether they are enjoying a game based on their own feelings Garris et al. (2002).

## Hints and Tips

The final advantage of gamifying education that will be discussed is the ability to be able to give users hints and tips. Students learning a topic can often get stuck, and sometimes with nowhere to turn, may give up and try again another time. What games can offer to educational topics that more traditional methods can not offer is that they can give the player hints and tips at that moment whilst playing. If a player is struggling with a particular topic, a prompt could be the only motivation a student needs to carry on. This hint could help the student continue working through the problem instead of stopping. This can aid further learning and a longer time will be spent playing the game because the user will be able to complete the levels.

Chen discussed this in his 2009 paper on giving users prompts to online learners Chen et al. (2009). He found that "the main factor affecting reflection levels is high levels prompts". He went on to say that "We believe that effective reflection can be achieved by reflection prompts". This means that prompts can be an effective tool for promoting learning performance through a process of reflection.

### 2.2.3 Current Educational Games

MyMaths is a popular subscription website that offers a number of mathematical puzzles, challenges and games for a student to work through. *My-Maths Website* (n.d.) From personal experience of using this platform when I was in secondary school, it was a much more exciting way of experiencing

mathematics because it was more interactive than just completing questions, it was understanding mathematics in a completely different way. The key element to this platform being successful is that it is fun. It keeps students engaged by making a more fun way to learn.

A reason to me why MyMaths was a good interactive resource is that it was available anywhere there was a computer and internet connection, which are becoming more easily available resources these days. This means that the tasks on the website could be completed during dedicated lesson times, but also could be done during lunchtimes or at home. It was easily accessible and this will be one of the advantage of creating an online game.

A highly successful website that targets a wider audience to teach them how to code is Codecademy. Sims & Bubinski (2013) Codecademy is a website that provides the ability to learn the basics of certain programming languages. It does this by creating a variety of different levels with goals at every stage of the process. Some of the levels can contribute to the creation of a larger program. Using bite size chunks of code as levels is an idea that could be incorporated into this project. This splits the task into manageable pieces and makes it easier to understand.

Another concept that Codecademy uses is badges for completing levels, challenges and for having a streak of days where you complete challenges. Receiving badges makes the user gain a sense of achievement and will motivate them to continue with the website. Again, this would be something that could be considered for the project, as it encourages people to continue playing your game.

Codecademy makes levels progressively harder with harder coding concepts as the user goes through the levels. The user needs to be eased in to very basic concepts so they will continue playing the game through the levels and not give up. Another reason to introduce the basic concepts first is that it is common that basic concepts will need to be used in conjunction with the more complicated topics, which can then be introduced in later levels.

## 2.3 Education

Learning is a process that comes naturally throughout life. Throughout growing up, many skills are learnt without even realising we are learning them. When students are in education, they are consciously trying to learn new things to benefit themselves and to improve themselves. Learning could be defined as "A change in behaviour as a result of experience or practice" amongst other things. Pritchard (2013) Trying to learn things in the best way possible is our goal and there of many different ways of learning material.

Similarly there are many different ways of teaching material. In this section some advantages of certain learning and teaching methods will be outlined. Gathering these different learning and teaching techniques will hopefully lead to the creation a more valuable game which can be used as an effective learning resource.

As summarised in Simulation and Gaming, games are perceived as more interesting than other ways of delivering material Garris et al. (2002). In a study by Cohen, also mentioned in Simulation and Gaming, he found that 87% of students were more interested in educational games than other classroom approaches. This highlights a want for educational games by students and an enthusiasm that might not be there for other approaches to teaching.

### 2.3.1 Different learning styles

Every person is different in which way they learn best and it would be rare that a person can only learn from one style, but the different ways of learning below are broadly the main ways people can take in and understand information.

- Visual Learning

Visual learners like pictures and graphs. They'd prefer to see something written down in front of them rather than being said to them. Maps and other visual learning tools can also be effective for these kind of learners.

- Auditory Learners

Auditory learners learn best by listening. They can take in information well from lectures and reading out loud to themselves. Good techniques for Auditory learners would be to record themselves saying information or hearing the information being said by others.

- Kinaesthetic Learners

Kinaesthetic learning learn best by doing stuff. They like to touch and feel objects. If information is presented to them they learn best by writing it down. Playing a game where they were actively playing it would suit them well.

McDonald & Hull (2012) Montemayor et al. (2009)

From these different learning styles, we can already see that different people learn in different ways. Although Kinaesthetic learners may benefit the

most from playing the game, aspects of other learning styles could be included in the game so that all different types of learners can benefit from the game. For Visual learners, I have an idea that a level summary that could be displayed after a certain number of levels. This could be a range of information summarising what techniques have been used by the player. If this information were available to save and print, then it may be of great benefit to a visual learner.

Auditory learners could also benefit from playing a game. An idea to help auditory learners learn effectively would be include audio clips within the game. This could be after the levels, in the post game summary. There could also be videos to accompany the game. This could make the game more interesting, but it could also have a increased impact with auditory learners if the audio content was relevant and educational.

### **2.3.2 Current Natural Deduction Educational Resources**

Natural Deduction has been taught in schools and universities for a long period now. With the more recent creation of the internet, there is now a number of different resources online which teach people the basics of natural deduction. All of them teach it in slightly different ways and each method of teaching the material will benefit a different group of learners better. In my game I will try and grasp some of the better characteristics of these ways of learning material and wrap them all up within my game.

- Notes

Notes for Natural Deduction are widely available online written by different groups of people. Enthusiasts and academics all have notes available which help for different levels of understanding Natural Deduction. One of the main advantages of notes are that they are clearly laid out and organised. Many sets of notes are written in LaTeX which gives them a professional and good aesthetic appearance. Laboreo (2005) Notes will be effective for visual learners who can see things written down and learn from them. One of the main advantages that I would like to incorporate into my game would be having easily displayed information available to view and also to print off for reference.

- Lectures

Lectures can be an invaluable resource for auditory learners. Academics in their field can really convey information effectively and explain it well because they are experts in their field. Lectures typically include a visual aid, which could be a slide show or writing on a black-

board to help students understand the concepts. npTELhrd (2015) This will benefit visual learners too, as they can see what is being explained. The important thing about lectures that could be included in the game is the audio. Audio is important to develop understanding and to explain concepts in a different way to the visual material.

- Videos

Online videos can be found online for teaching Natural Deduction, such on websites like YouTube. PhilHelper (2013) These are more interactive than tutorials online which are just powerpoint slides because they have audio over the top of them explaining what is happening. This particular video, uploaded by the user PhilHelper on YouTube, uses graphics and examples to clarify some of the mathematical points. The mathematical notation is written next to the example, which keeps the mathematical notation in the forefront of the viewer's mind. This will appeal to visual learners. He is also speaking through what is on screen, so will benefit auditory viewers also. Some of the materials that he has in his videos would be effective in the project between levels of the game, to reinforce the learning.

From these common techniques of delivering material, there is a trend that visual and auditory learners are well catered for through Notes and Lectures, as well as Videos catering for both. Kinesthetic learners may miss out somewhat by these traditional ways of learning.

### 2.3.3 Online Natural Deduction Resources

Whilst not strictly an educational resource, Automated Theorem Provers are computer driven systems that can automatically work out a proof tree from a given end point Sutcliffe (n.d.). These have their advantages as they automatically create a proof, so once the theory has been learnt by an individual, it can be checked by the automated theorem prover. What this project wants to achieve would be to have an automated theorem prover which can guide a user through creating a proof for themselves. The game will have all of the logic ingrained in the code, but the objective is to make the user learn as they go along.

One similar project that has been undertaken is a Proof Assistant for Natural Deduction logic Gasquet et al. (2011). This tool has many good points that are worth highlighting. The notation is in keeping for what will be included in this project. It is also a very interactive system that will definitely benefit kinaesthetic learners. Things that will be focused on in this project is the aspect of becoming a game. Utilising levelling systems, fun and giving hints

and tips to benefit the user. Additional features of some sort of social aspect would also improve engagement in the game.

By creating this project, it aims to target this type of learner through the interactive process of getting the player to 'do' things. If Natural Deduction can be taught effectively as a game, all the different learning types will have some benefit from playing, but especially kinaesthetic learners.

## 2.4 Natural Deduction

Natural deduction is a mathematical logic system. It is a system for proving statements of formal logic. It is used to show if certain statements are valid and to prove them. Pelletier (1999) Natural Deduction has a typical mathematical notation system that could create a barrier between the majority of general public understand how Natural Deduction works. Through this project, I hope to break down this barriers and make Natural Deduction more accessible to a wider range of people.

Natural Deduction is used in a way that a large problem that needs to be proved is broken down into singular steps which are easily manageable. Each individual step is proved, and then that information can be used going forward in the proof. Halbach (n.d.) These easier to understand proofs can then contribute to a larger Natural Deduction proof which was not trivial at the start. This lends itself well to leveling up system in my game, where the user could complete easier tasks in early levels and end up joining all of their findings together to prove a more complex theorem.

### 2.4.1 History

Natural Deduction as an idea formally came to be in 1934/35 when Gerhard Gentzen and Stanisław Jaśkowski both produced papers independently of each other. Jaśkowski (1934) Gentzen (1964) No content had been published on this topic before and this led to the start of what is now known as the Natural Deduction Logic system. Both papers published in the same year had a very similar topic area of taking basic logic as a given- assumed to be true- and then creating a method of exploring what can be done with these basic axioms to prove more complicated theorems. Pelletier (1999)

Whilst this was the true start of where the ideas of Natural Deduction came from, it did not pick up widespread appeal until educational textbooks begun to publish this logical system in the 1950s. Pelletier (1999) At this point the Natural Deduction language was shaped into what it was today. Features and extra bits were added to the language that were not previously available



before and more and more people began to learn about the concepts of Natural Deduction logic. People were taught how to solve practical problems using logic that had discovered and taught to them.

Not only is Natural Deduction important for practical uses in solving proofs. Over the 1960's and even continuing now, Natural Deduction has been used as a theoretical tool, and lots of research has gone into it to improve how Natural Deduction is applied. The aim of this is to make practical Natural Deduction problems easier to solve and quicker, possibly using computing resources to do so. The way Natural Deduction proofs are used has expanded, partly due to the work of Prawitz and Raggio on Normal Proofs. Indrzejczak (n.d.) More work continues to be done on many streams of Natural Deduction today to improve the way people use the logic system.

#### **2.4.2 Why Natural Deduction is Suitable to Make a Game for**

Natural Deduction is a logical proof system which is thought of as closely aligned to people's natural reasoning patterns. Arthur (2011) It should, in theory, be easy to teach a system that is closely related to people's intuitive reasoning patterns. In reality, it might be that the mathematical notation could be one of the main things standing in the way of people learning this logical system. By creating something that is easier to grasp and understand, users should be able to connect with their intuition once again and understand the concepts behind Natural Deduction.

Because of its close connection with being taught since the 1950s, this way of teaching logic has been one of the primary methods for many years now. Its ongoing reputation as a primary way of teaching mathematical logic in both mathematics and philosophy means that it is still very popular today. An easily accessible way to learn this new 'language' for the first time can only be beneficial in my opinion.

Natural Deduction is a core system of logic that has been used regularly over the last fifty years. This importance has been shown through the amount of educational resources produced and the amount of material that has been written on the topic of Natural Deduction. The lacking of a fun, easy way to learn Natural Deduction is a shame and is why I feel it is necessary to create a game based on this logical system.

This project provides academic merit because of the alternative way that Natural Deduction logic will be taught in. Whilst primarily benefiting users and students who use the software for their own educational benefit, many academic experts and educational leaders in logic fields should find this

paper interesting due to different ways of conveying traditional logical information to people who are not familiar with the concepts.

### 2.4.3 Mathematics

This section will cover the basics of mathematics that will be implemented into the project. Most textbooks use similar notation but Fitch notation written in lines is sometimes used. For the purpose of this project, tree-like presentations will be used.

#### Notation

- And (Conjunction)  $\wedge$

A and B both have to be true for  $A \wedge B$  to be true.

- Or (Disjunction)  $\vee$

Either A has to be true or B has to be true or both for  $A \vee B$  to be true.

- Implies  $\Rightarrow$

The statement  $A \Rightarrow B$  says that whatever A is, it must also hold for B as well. This shows consequence. You cannot however deduce A from B with this notation.

- For all  $\forall$

All elements in a set make a statement true then the for all symbol can be used appropriately.  $S = \{1, 2, 3\} \forall x \in S, x < 4$ .

- There exists  $\exists$

This means that at least one item in a particular set exists and holds true.  $S = \{1, 4, 9\} \exists x \in S, \text{ s.t } x > 7$ .

- If and only if  $\Leftrightarrow$

$A \Leftrightarrow B$  means that we can deduce A from B and we can also deduce B from A.

- Not  $\neg$

The opposite of being true. A being false implies  $\neg A$  is true.

Laboreo (2005)

## Rules

Everything on the top line is assumed to be true, with each statement separated by a comma. What is deduced is below the line. In proofs, there will be multiple lines on top of each other.

**Conjunction introduction** Introducing the 'and' symbol into the proof means that given two things that are deemed to be true, we can conclude both together are true. In formal mathematical notation, it would be written as shown in equation 1 below.

$$\frac{A \quad B}{A \wedge B} \wedge I$$

A real life example of this would go as follows:

1. It is sunny.
2. It is hot.
3. Therefore it can be concluded: It is sunny and hot.

**Conjunction elimination** Whilst the introduction takes two pieces of separate pieces of information and makes it into one new piece of information, conjunction elimination takes one piece of already assumed information and from this we can deduce two separate pieces of information. Equations 2 and 3 show what can be achieved.

$$\frac{A \wedge B}{A} \wedge E1$$

$$\frac{A \wedge B}{B} \wedge E2$$

Similarly to the previous example we use conjunction, but instead of deducing the conjunction, we already have it, so we can deduce the individual components.

1. It is sunny and hot.
2. Therefore it can be concluded: It is sunny.
3. It can also be concluded: It is hot.

**Implication introduction** This takes two statements. The first of which is assumed to be true. Now if we can deduce from this another element is true then trivially one must imply the other.

$$\frac{[x : A] \quad B}{A \rightarrow B} \rightarrow I/x$$

**Implication elimination (Modus ponens)** In implication elimination we are given an implication and (at least) one other piece of information and from this we can eliminate the implication to conclude that another element is also true. In Equation 4 below, we see that  $A \rightarrow B$  is true but this is not enough to conclude that  $A$  or  $B$  is individually true. By being given the extra information that  $A$  is true, we can then use this rule to deduce  $B$  must also be true.

$$\frac{A \rightarrow B \quad A}{B} \rightarrow E$$

A practical example of this rule being used may be clearer to understand:

1. James eating chocolate causes James to be happy.
2. James is eating chocolate.
3. It can be concluded: James is happy.

In relation to Equation 4 above, James eating chocolate is  $A$  and James being happy is  $B$ . When the fact is given that James *is* eating chocolate, then it can be concluded that James is happy and this is true on its own because of the other information we have.

**Disjunction introduction** This introduces the 'or' symbol into an equation based on the fact that one of the variables is true. This is because only one of the variables need to hold true for the disjunction to hold true. This is shown in Equation 5 and then an example is given in text.

$$\frac{A}{A \vee B} \vee I$$

1. Football is a sport.
2. Football is a sport or A Lemon is a sport.

The second variable does not have to be true because the first variable, in this case 'Football', makes the statement hold true. The disjunction would still hold true if both the variables held true.

**Disjunction elimination** This rule is very slightly more complex and gives a flavour of what complexity is possible with natural deduction proofs. This is still a simple rule and this list is not exhaustive of rules. Disjunction elimination manages to remove the disjunction because of other information given. There are two implication statements, both implying the same thing. Both these implications hold true. There is then a third true statement, which is a disjunction between the first variables of each implication. Because of the disjunction, it holds that at least one of the variables is true, meaning that the third variable that is implied is true. We conclude that the implied variable is true. Displayed in Equation 6 is a much simpler way to explain the rule using formal mathematical notation.

$$\frac{A \rightarrow C, B \rightarrow C, A \vee B}{C} \vee E$$

An even easier way to understand this is to give a simple example.

1. James eating chocolate causes James to be happy.
2. James eating pizza causes James to be happy.
3. James is eating chocolate or James is eating pizza (*or both*).
4. It can be concluded: James is happy.

This is exactly the same to what is displayed above but using examples instead of complicated mathematical notation.

Halbach (n.d.) Laboreo (2005) Indrzejczak (n.d.)

## 2.5 Concluding remarks

The three sections in this literature review: Gamification, Education and Natural Deduction have all touched on areas that will be brought together in this project.

The benefits of games and what makes people play them have been discussed. The advantages that a game could bring to an educational resource has also been researched in detail. Current games that are successful in the real world show what games can do if created in the correct way. I conclude that a game can be a valuable resource for educational purposes where people can learn better. This, in my opinion, justifies why a game should be made of Natural Deduction.

Education, the second section, talked about different learning styles. There is a lack of kinaesthetic leaning in mainstream Mathematics and this is

exactly what a game can provide. I looked at similar resources and projects to my own and how I feel they can be improved upon. Due to the lack of similar, helpful resources there is definitely a need for this kind of online platform that can help educate.

Finally, an introduction to Natural Deduction. This gave a background to the history and to reasons why Natural Deduction should be used to make a game. Natural deduction is still one of the core logic concepts and it is important to make sure people know about it.

All these things lead me to believe that a project in this area would be extremely worthwhile which will have value to many different groups. Academics and Students alike should find benefit in a resource of this kind. This project, whilst mainly appealing to Mathematicians due to the content, should also appeal to other key departments such as Computer Science where Natural deduction is used extensively. Philosophers should also find this project interesting due to the ongoing research into whether gaming is an effective teaching resource.

## Chapter 3

# Requirements Analysis and Requirements Specification

### 3.1 Target Audience

It is important to recognise that the difficulty of game that will be produced will depend on what audience the game will be targeted at. A game for academics or for university students would have to be far more technical and complete in Natural Deduction theory for solving mathematical proofs than a game which would be made for primary school children, for example.

When deciding on what would be the suitable age group for the educational game, considering the age that someone would learn Natural Deduction logic highly influenced the process of picking the target audience. Whilst most people are not aware of Natural Deduction before 18 years of age during Secondary Education, most people are introduced to First Order Logic and Natural Deduction theory during an Undergraduate Degree.

A game that introduces you to Natural Deduction concepts at a stage in life before you learn it formally in education seemed like the most favourable option as this Sixth Form/Collage (16-18 years old) age is the age that a different style of learning may be most beneficial. At this age, education is delivered traditionally in a School environment which can be very different to University education. At University, you get taught the basics but you have to research the concepts in further detail and try examples for yourself, whereas in School, this is all given to you during lessons.

A tool that tries to bridge this gap by giving you basic examples of the Nat-

ural Deduction theory should help in making practise easier and therefore helps you to to learn quicker. This game will provide a more entertaining way to learn than traditional methods of learning. This target age group of around 18 years old spends more time on computers and mobile devices than previous generations, \*\*\*\*\*CITE HERE\*\*\*\*\* so using computers and gamification to help students learn could be very effective, as on computers is where a lot of time is spent currently.

## 3.2 Project Goals and Targets

When undertaking this project, the time researching literature associated with this topic has shown there are a number of things that this project must broadly achieve in order for it to be successful. Without a number of the following being met, the project will not be effective for the purpose it has been created for. Delivering the majority of these goals however, will ensure a project which displays ingenuity and has real use and value to the target audience. The goals and targets of the project are set out below:

- Create a new piece of software which successfully integrates Natural Deduction theory and gamification concepts.
- Give users an engaging experience where they benefit educationally.
- Support users and give them help and advice when stuck in the game.
- Make the software accessable so no prior knowledge of Natural Deduction is known.
- Use proper Natural Deduction notation so users can effectively transfer knowledge learnt in the game to the acedemic environment.
- Deliver a usable software project on or before 29th April 2016.
- Use Kinesthetic Learning techniques to benefit users who learn in this style.
- Take current attributes in educational games to make sure my project encapsulates qualities of what makes games effective learning tools.

## 3.3 Software Requirements

Effective requirements set before implementation of software leads to a focused design process and a structured framework. There are functional



requirements which is what the software should do and non-functional requirements which describes how the system is expected to work. Within the functional requirements there are Gaming Elements. These are specific functional requirements that will make my software play like a game, rather than just a drag and drop tool.

### 3.3.1 Functional Requirements

- Provide an intuitive drag and drop interface.
  - Making it clear what to do and how to do it. Making sure that the user knows how they can drag and drop items. An algorithm will have to be implemented where users can drag and drop onto certain elements of a proof. This could be that a user can drop onto a certain area of a proof, or could drop onto particular elements of a proof. This will need to be investigated in the design of the program.
- Making all elements of the game drag and drop.
  - This would mean no part of the game would need to be controlled by the keyboard. Everything being able to be dragged and dropped makes it nicer to use.
- Have proofs rendered nicely so they fit on the screen.
  - Proof trees can get very big so this would be a way of nicely scaling proofs on screen so a user can easily tell what the proof is. This will have a function that will render the proof tree.
- Allowing the user to experiment with proofs and possibly create their own levels.
  - Creating a flexible system that can be expanded upon is important for future development of the system.
- An algorithm or a system that can successfully recognise whether a solution to a proof is correct.
  - In an ideal world an algorithm would be automatically be able to determine whether a solution to a problem is a correct solution. If this is not feasible during the time period then a hardcoded system should implemented.

### Gaming Elements

- Provide hints and tips for the user.

- Hints and tips enhance a user experience as if they get stuck, then hints and tips will prevent them from giving up.
- Have levels that will increase in difficulty and guide the user through the learning process.
  - A gradual build up in difficulty can guide a user through the game and keep them entertained. The process of a user being able to progress through the levels is an important gaming element that this project should implement.
- Create a reward, scoring or recognition system for users to tell how they are doing.
  - This could be the form of a score counter, letting the user how many levels the user has got correct, or could be as simple as just letting the user know whether that individual level is correct or not.
- It should be more fun than learning Natural Deduction logic in a traditional way.
  - The key factor for this game to be successful is that users can learn the required material in a more entertaining and light-hearted way than they would learn it in an educational environment.

### 3.3.2 Non Functional Requirements

- Testing must be undertaken to make sure the software is working as desired.
  - Continuous testing must be done by the programmer to make sure that the code is continuing to react as expected and there are no obvious bugs present.
  - User testing must be completed before the end of the project to get feedback on improvements to the system.
  - Some unit testing should be completed to check that the code is solid and nothing unusual should take place when in use.
- The system must react consistently with 100% reliability.
  - Every time elements are dragged and dropped, the system must react the same so everything happens as expected.
  - Algorithms in place are reliable and executed as expected every time.

- The system must be secure.
  - The system must be able to cope and should not be able to be changed permernantly by outside influences.
- The code should be witten in a way which is easy to maintain.
  - The code should be written in small functions which can easily be read by other developers.
  - The code should be sufficiently commented so that other developers can easily understand what is happening in the algorithms.
  - All function and variable names should be sensibly named so that other developers can easily grasp the gist of the code.
- The program must be written in a way in which it could be scaled in the future.
  - The code must be written in such as way that it could be expanded on in future. This can be achieved with a clear layout of the code, where the maintainability requirement is also achieved with similar approach to how the code is written.
  - The software will be open source to allow for the continual development of the platform

Scalibility Usablity

How the system is supposed to be.

### 3.4 Hardware Requirements

All software projects require hardware for them to run on. It is important to outline what the capabilities of the software should be so it can run on the correct hardware.

- All code should run by the client.
  - Using a language which uses client side increases the speed of load times, reduces the reliance of a server.
  - By deciding to run the code client side, it is important that the code is not too computationally demanding so that it runs on a variety of architecures.
- The software should be able to run on a web browser and on a variety of browsers.

- The language chosen should be able to easily intergrate into a web browser.
- It must be able to work on a variety of web browsers so that the software is available to use by a wide range of users across different platforms.
- The software should run comfortably on a PC or Laptop and be played with a mouse.
  - This software project is designed for use on a PC or Laptop. It shouldn't neccesarily be made to work on mobiles or tablets with a touchscreen due to time restrictions.

### 3.5 Programming Language

When analysing the structure of this project, thought need to put in about the format, how the game would be accessed and how the game would be played. These factors directly influence what language should be used to write the software, as different programming languages have different strengths. The Software and Hardware requirements were also needed to be taken into consideration to assess which language would be most suitable for my needs.

The first thought is that this project could be written in Java. Java is an object orientated language which would be important in this project for creating proof trees. Java is the language where I have had most experience and so therefore seemed a good choice. A major disadvantage of Java was that it did not lend itself nicely for web development which therefore could lead to problems when it was time to test the software and distribute the game to users.

Considering accessibility requirements and software requirements, a website based game was decided on early in the process. This led me to a choice where the project could either be created in a language that is native to being online, or create it in a different language and then allow a user to download and install the game. Because the project would be quite lightweight, with no need for external databases or dealing with large amounts of data, it was decided that a game that could be played in the browser would be the easiest way to move forward.

I settled on a HTML 5 canvas for the playing area and javascript for the majority of the logic. Javascript seemed a sensible option because of the way it runs. It is fast and has small waiting times because it is a client-side programming language. It doesn't need to contact a server so there is no

latency in this part, meaning that any code can be executed straightaway. Another reason that Javascript was chosen was because of its ease to learn and simplicity. There is lots of documentation and tutorials available online and this makes Javascript accessible to new web developers.

The fact that Javascript can be easily integrated with HTML was another big reason that it was picked for the project. A HTML 5 canvas was the easiest way to get a drag and drop interface working and made it easy to show a user which area is clickable. The logic is implemented by Javascript with the HTML canvas being the area that the user sees on the front end. Using these two languages together makes it easy to publish a web page, with no servers running continuously storing data. All that is needed is a website host.

Javascript is also mainly object orientated, where objects can contain other objects. \*\*\* CITE HERE \*\*\*\* <http://www.crockford.com/javascript/javascript.html> This is important with the data structure of the project, where a tree structure needs to be implemented to correctly draw natural deduction proofs. By having a structure where objects can be placed within objects, this tree structure can be achieved.

Another language that was considered for the project is PHP. Javascript and PHP both have their advantages and disadvantages and personal preference was the main reason Javascript was picked for this project. PHP also makes it easy for quick web development so would have been another sensible choice. PHP however is used more for server side programming, something that was felt that this project did not require. To keep this project as simple as possible for both the programmer and the user, server side programming was avoided. There is nothing in this project that would actively need the processing power of an external server, and everything that is created in this project should be able to run on the host's computer, as it is not very demanding.

### **3.6 Project Limitations and Changes to Existing Systems**

There comes a point in every development where there needs to be a cut off point. Realising that this project will not be complete is acceptable, but it must reach the minimum standard in the software requirements outlined above.

Due to all of the development will be client side, it needs to be accepted that it will be difficult to store information for different sessions. This project does not plan to use a database to store information so users will not be able

save levels for future use. Levels could still be created by users and then the data structures could be emailed so they could then be incorporated into the project. An idea could be for a user to press a button to submit an proof suggestion.

Another limitation of the project is that there will only be a finite set of things that the project can do. Whilst the basic elements of conjunction, implication and negation should be implemented, it may be more challenging to implement other operators that work in a different way. By getting at least the basics of Natural Deduction in there and working correctly, it will still be a functioning and valuable learning resource for an introduction to Natural Deduction.

Of the limited other systems that are currently in existence, it is important to recognise what they are doing well, and what should be improved upon. Pandora is "a tool for supporting the learning of first order natural deduction". \*\*\*\*CITE HERE\*\*\*\* It does a number of things successfully that this project would like to take on board. It has a series of exercises which users can work through which is what this project is aiming to create in the form of levels in a game. Pandora successfully teaches the user by increasing the difficulty of levels and giving hints and pointers as the user works their way through the game. Hints and tips is an important feature in the learning process so implementing this correctly and taking inspiration from Pandora will be useful in sculpting this project.

The main thing that will be done differently in this project will be the style of the proofs and how they are laid out. In Pandora, proofs are displayed using the Fitch notation. This is a series of lines where each one displays the next line of the proof. Whilst this notation is perfectly valid, in British Universities Natural Deduction and other logic systems are traditionally taught using the tree structures where all premises are above the line and the conclusion is below the line. Conclusions can also be premises for other logical statements and this is how the tree like structure is built.

Panda is an alternative current natural deduction tool that currently implements a tree structure. This is what this project does successfully well. \*\*\*CITE HERE\*\*\* This software displays the proofs graphically in a way I would like to replicate. Panda has a number of buttons for introducing a variety of different rules. This is also a nice way to do things and the user can easily understand what is happening.

One thing that this project will aim to do that panda does not do is make the complete interface drag and drop. Using Panda, there is still a fair bit of typing that needs to be done, which could be perceived as a bit clunky as the user needs to switch between the keyboard and mouse to undertake the tasks. If this software is created so it is all click and drag, it not only

makes the process of playing smoother, it could be easily created for other devices like touchscreens in future development iterations.

Both of these tools do a great job of using Natural Deduction in an unconventional way by creating software that people can learn from. What this project should build on is implementing more features to make a user feel like they are playing a game. If some of the key gaming attributes could be introduced to software which can teach you natural deduction logic, then users should be more engaged and will not even realise that they are learning when they are playing.

## Chapter 4

# Design

### 4.1 Data Design

Outlined here will be the different data structures that will be used to create the project and to make sure that all the data structures that are required are accounted for and that everything that the project will need to work successfully is outlined prior to implementation.

The project will consist of a number of drag and drop elements, each one can be individually selected by the user and moved. Every time a new element that can be dragged and dropped is created, a new object should be created that will have a number of variables that are individually associated with that object.

The x and y position of the object on the screen will obviously need to be stored and these values will be updated every time that the object is moved. The x and y coordinates stored will correspond to the centre of the object and this will be the point that the object will be moved from.

A piece of data that needs to be included in every object made associated with the x and y coordinates is the width and height of the box. The width and height of the box will vary based on how much of the proof needs to be drawn on it. The width and height of the box will determine the external structure where the border is drawn, with the idea that all of the proof should always fit within the structure.

The structure of the object should be stored. This will be how the proof is interpreted and the drawing function will recognise the proof structure and use an algorithm to draw the proof correctly. The structure of this will be an array of four elements. Each element will be one of three types, a string, a number, or another array. If text for a proof needs to be shown



then that particular element will be a string. If there is a dot where a user can drag on another part of the proof then a number will be displayed. The drawing algorithm will recognise the number and render it as a dot which can be dragged on to. Thirdly, the array element could be another array. This would represent further proof structure that needs to be drawn. The recursive process would then start again where this nested array would also have four elements.

The colour of the border will need to be stored. This will indicate whether this object is hovering over another element. If the border colour is changed when the user lets go of the mouse, then an action will happen. A different border colour could be used for different actions.

There will be dots in the proof where users can drag statements onto. Each dot will need to have a unique number so they can be identified. These dot numbers will need to be stored in an array unique to the object. This will be because when a proof structure is deleted from the canvas, all the associated dots need to also be removed from the canvas so that users cannot drag onto a point on the canvas that no longer exists.

The object elements which will be represented as drag and droppable squares on the canvas. These elements will all need to be stored in an array so they can be iterated through and all be drawn on the canvas. The draw function will continuously update what is drawn on the screen. By storing all the elements in an array, any new elements can just be appended onto the end of the array, because in Javascript there does not need to be a fixed sized array. When elements are deleted, they can be removed from this array and then they will not be rendered on the canvas.

Every dot will also need to be its own object. Each time a proof is created, the correct amount of dot objects will also need to be created. This is because dots need to be dragged on by other elements so the individual position of every dot is needed so the software knows whether the user is hovering over the dot. This means that the x and y coordinates need to be stored as variables of the object.

As well as this, each dot object will need to have a unique identifier. This will be the number that is represented in the structure of the proof and will act as the link between where the dot is rendered on the screen using the proof structure and the dot object which will hold all of the positional information.

The colour of the dot will also need to be stored. This will be the same colour as the colour of the object border that is being moved, so if the dot and object being moved are on top of each other then the colour of both will change. This will allow the user to clearly tell which dot the proof will go onto if the mouse button is released.

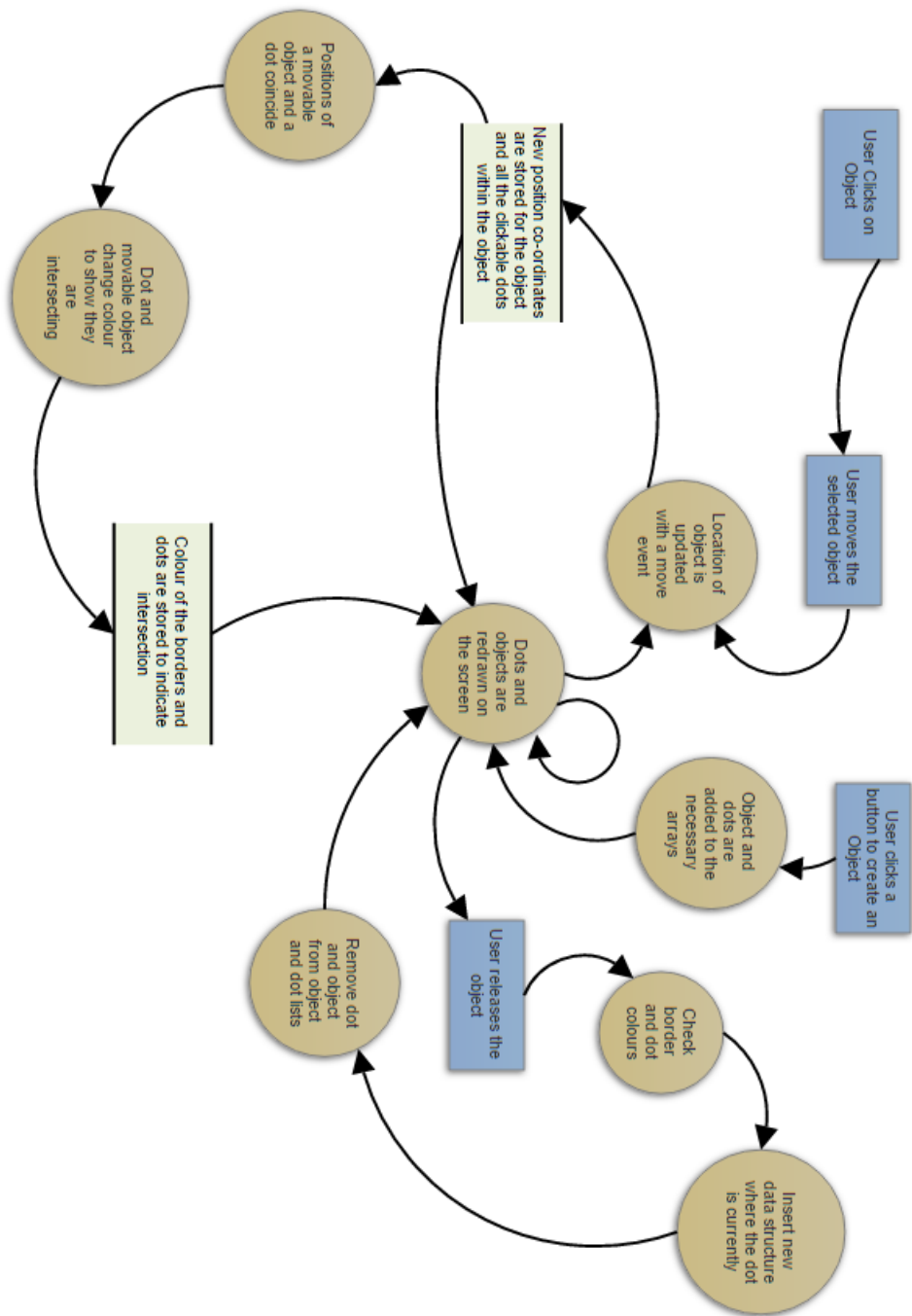
A final field will be needed which is a variable. This will be for if the dot gets replaced by a statement or a proof structure then whatever the structure is will be put into this variable. Next time the dot is due to be drawn, the algorithm will be able to recognise the presence of data in this variable and it will replace the dot with the structure. The normal drawing procedure will then continue and correctly draw the correct shaped proof.

Like the draggable objects, the dots will also need to be stored in an array. Whenever new proof structures are created, the correct of amount of dots will also need to be created and these can be added to the dots array. Dots can then be iterated through to find out if any positions of objects coincide with the position of the dot.

For each level of the game, data needs to be stored so when the user believes they have completed a level, they can check against the stored answer to verify. Stored answers to levels should be in the same format as the proof structure that will be created in the objects. When the proposed solution needs to be compared to the model solution, the two solutions can just be compared to see if they are the same. Solutions for each levels can either be stored as separate variables or as elements of an array. As the list of levels will be static, it does not matter too much what data structure the levels are in, as long as the level structure is the same nested list structure as what the user will create.

## 4.2 Data Flow Diagram

Figure 1 displays how data is stored and used throughout the dragging and dropping process. It includes the actions taken by the user and the processes that happen after the user has executed an action. Boxes in blue indicate User actions, brown circles indicate processes happening and white rectangles indicate data that is stored.



## 4.3 Architecture Design

This section describes the structure and layout of the program. This will describe in further detail how things will be done in the software and how all of the requirements will be met. It will outline how different sections of the program will interact and work together to create successful software without going into all the details of individual algorithms.

There will be a very simple file structure to the program. The site will be run from a HTML file. Embedded in this will be 2 Javascript files, one which will be a linear tutorial for users to go through and a second file that will allow the user play the game. A CSS file will be in charge of the design of the program which will make it look visually appealing and align elements in the correct places. These four files will contain everything that is needed to run the project. The two Javascript files will contain all of the logic of the program and have all the algorithms needed to draw the objects and to work out all of the dragging and dropping.

The tutorial file will be static code where the user presses a button to reveal more of the tutorial, explaining the basic techniques to the user so they can gain skills and knowledge. This will have the same layout and style as the main game which the user can interact with, so that the user will be familiar with what they must do when they have to make their own solutions. Each part will be hidden or visible as the tutorial progresses. As the user finishes this tutorial, they will automatically be transferred to the page where the user can play and interact with the game and fully play it.

The main Javascript file will contain many functions which all have specific tasks. All of these functions will be intertwined to make the key logic structure of the software. Keeping the functions small and specific will make the code more maintainable and easier to read. It will also help when scaling the software and increasing its capabilities.

The three basic functions needed for dragging and dropping will be functions what to do when the mouse is clicked down, and when then mouse is released. Whilst these are the main functions, other tasks will be in separate functions and called from these functions.

When the mouse is pressed down, the function needs to determine whether the location of the mouse is over any of the drag and droppable objects on the screen. If it is over an object then this should be the current object selected.

When an object is selected then the function for moving the mouse will be active. This function will need to continuously update the location of the dragged object so it is correctly represented when the mouse is clicked down.

This function must also recognise if the active object that is being moved has the same position of any of the other objects on the screen. If it does, then this function must react accordingly.

Finally, when the mouse is released any appropriate actions needed will actually be confirmed. This could be the joining together of two objects, the deletion of objects or possibly other interactions. It will also release the current object so the software recognises that no object is now selected to be moved.

Some of the main functions that will need to be called from these functions are as follows:

- A deletion function which will remove both the object that can be dragged and the dots associated with it
- A function that will check whether an object is being dragged onto another one
- Functions that will concatenate formulas and proofs based on what area of the proof was dragged on to

While these functions will be called by an event - a mouse click or movement - there also needs to be functions that will render the proof continuously so that the user gets a good experience from dragging and dropping items and it does not lag.

A drawing function must be called periodically. This can be done that from an initial function that will set up the environment, create the canvas and set the correct level so the user knows what they are doing. The initial function will be called on startup so the canvas is always present when the application is first loaded.

The drawing function will have to cycle through all the objects and draw them individually. If the proof is a statement which will be represented as a string, it will need to be rendered differently to if it is a proof structure, where multiple layers will need to be drawn and the box size changing respectively. For doing each of these things there will be a different function. Within the drawing function there will be a number of library functions used to draw text, align the text correctly and change the colours of text and borders.

Users will need to create their own proofs from scratch and they will do this by having a number of buttons available to them which will then call a function in the Javascript to create the desired object. The buttons will be created in the HTML file and will call a function in the Javascript which is connected to that button. There will be javascript functions for differ-

ent proof shapes, variables and connectors such as implication and negation.

The last set of functions that will be important for the game to work successfully is functions that determine the level information and that will verify whether answers are correct. A function will hide and show certain information depending on the level that a user is on. Removing certain elements that is not needed for a level makes it easier for a user to know what to do. Also, certain hints and tips can be shown on various levels depending on what the user has learnt to do. This function will necessarily make sure the user is not overwhelmed with content and only the parts they need to use are included in each level.

A final function will be a verification check to see if what the user has created is a valid solution, and whether this solution is correct. This will be a comparison algorithm between an ideal, valid solution for a particular level and the user's attempt at the level. This algorithm should take into account different orders that the proof can be presented in, which will make it more complex than just a simple comparison algorithm.

## 4.4 Interface Design

### 4.4.1 Design Principles

Designing the visual representation that the user will be presented with is crucial to any design process. Making something that is aesthetically pleasing to look at and also easy to navigate can be just as important to the user as the functionality of the software. Software which has the capabilities to do a lot and is computationally good but has a poor user interface can really be detrimental to user experience so thinking about how users interact with the software needs to be considered.

My user will be teenagers and young adults. Realising the users that will mainly play the game will affect the design. This age group are technologically savvy and are used to frequently using touchscreens where swiping and dragging is commonplace. While this project will not be using touchscreens, understanding that the audience does not need to be patronised about how to use a computer is important. By keeping a simple and straightforward design, the users can focus their time on learning Natural Deduction and less time navigating many menus. Users will achieve the levels quicker with a decluttered screen with only the necessary features displayed.

Consistency is also crucial when designing a user interface. Each level should have the same layout, the canvas in the same place, the buttons in the same

place. By creating this consistency, the user will familiarise themselves with the structure and layout, and they will know where to go to, for example, create a new variable object, or what to do when they need delete everything on the screen. The tutorial is especially important to have the same format as the main game, because this will be the first time a user will see the game and the interface. This will be when most of the learning of the interface will take place.

The interface should be designed so if the user has not learnt correct way to use the interface, they should be notified what they have done correctly or incorrectly. This instant user feedback will not only help the user how to use the interface correctly, but also stop future similar mistakes. Users should not be in a position that they are puzzled in what is going on and therefore should be constantly told if an action they have performed is invalid.

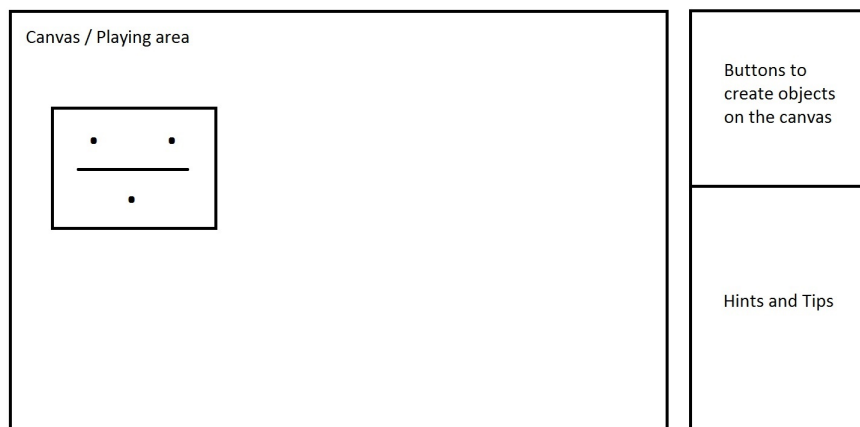
Even if an action is valid, sometimes the user is mistaken in their action. Users ideally would not want to start again if they make one mistake. Some way of rectifying a mistake will be beneficial for overall user experience. Creating a user interface which takes into account mistakes will hopefully be used much more that requires a user to go through a long process of creating their proof again.

Crucially, the user must feel they are in control of the system at all times. They will feel calm and comfortable when what they expect to happen does happen and this is helped by the consistent clear design. If the user doesn't encounter any surprises and is not confused then the design is fit for purpose.

#### **4.4.2 User Designs**

Taking into account all of these attributes, it is important to envisage and plan what you want the ideal user interface design to look like before implementing it. Four designs for a potential user interface were suggested for different elements of the game. Advantages and disadvantages of each design are outlined below:

### Basic Level Design

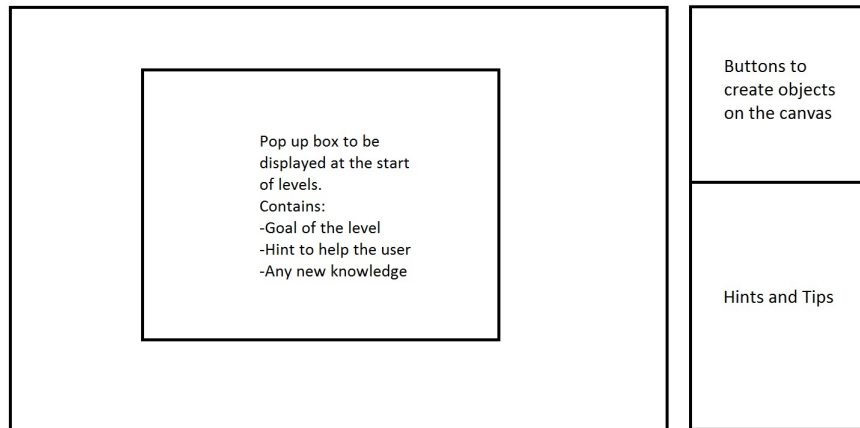


This first design is the basic layout of the game. It keeps a clean, simple design where every box has a very different purpose. The canvas is detached from the rest of the game to indicate the area you can drag and drop from. This will be outlined in the tutorial page. The box that is placed on the canvas is a proof structure which appears once a proof structure button is pressed in the top right box where the buttons are contained. Hints that the user can use to remind themselves what they have learnt are placed in the bottom right box and there may be an option to toggle hints on and off, so the user only gets the help if they need it.

A bin and brackets will be placed in the corners of the canvas so users can drag there elements to either the bin or brackets, depending on what the user wants to do. By keeping the core design to these simple elements, it will make the user feel in control because there are no complicated menus to navigate, making the user experience smooth.

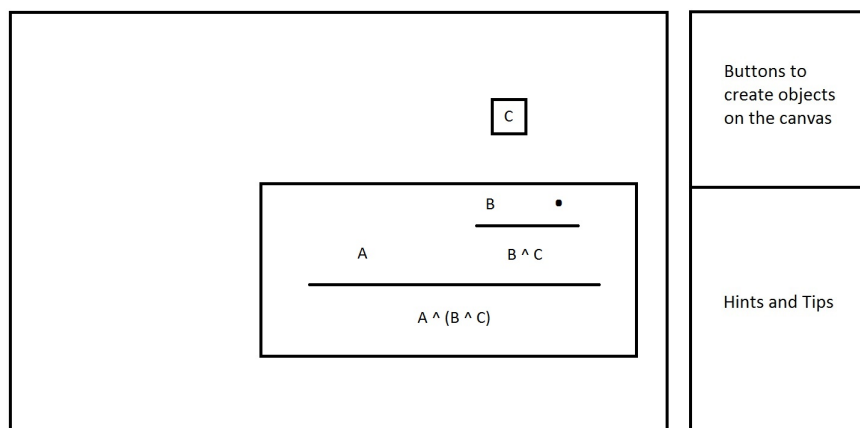


### Pop up boxes for level introduction



The second design screen understandably takes the very similar format of the first screen. All of the boxes are in the same place which keep continuity. This screen is to demonstrate what is presented to the user when they start a new level. All of the boxes including the canvas and the buttons will remain visible but not clickable whilst the pop up is active on the screen. The pop up will show the proof that the user needs to make. Pop ups will also be used to introduce new content to the user that they can then use in future levels. The user will have to click a button on the pop up to dismiss it and then they can play the level, like shown in the first design.

### Multiple lines of proof and other objects on the canvas



The third screen displays how a user will build a proof of multiple layers. Once the user drags proofs together then the software will automatically detect the new form the proof needs to be drawn in and will be rendered differently. The only positions that users will be able to drag onto the proof will be where the dots exist. In the design, the user would drag the 'C' element onto the remaining dot to complete the proof.

## Chapter 5

# Implementation

### 5.1 Beginning the project

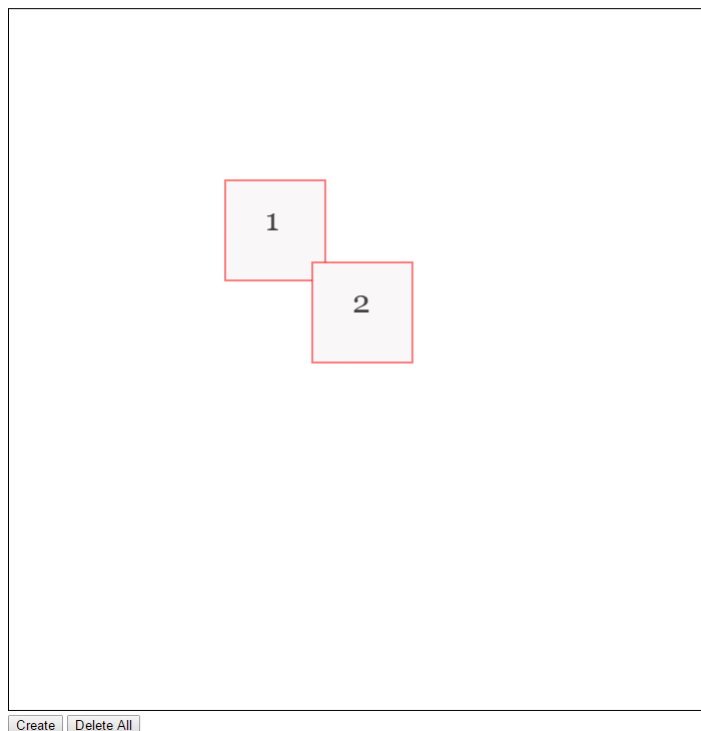
Being new to Javascript, the first few weeks of coding was learning the language and becoming familiar with how it all worked. This included knitting javascript together with HTML and CSS.

For any software project, keeping a good record of progress is important. In this case, GitHub was chosen as the version control. GitHub was simple to set up and use. It is used to keep notes of all of the progress of the project as it goes along and so advancements in the software can be documented easily, all in one place. It is also incredibly useful if changes are made to the software meaning that it no longer works. It is straightforward to revert to a previous committed version and carry on again from then.

Learning how to use the HTML 5 canvas as well as the methods associated with this that were available took some getting used to. The first mini project that I created was a small game that was played on a HTML canvas. The user would use a button to create numbers. They could drag and drop these numbers on top of each other. If they dragged an item over the top half of another item then the two numbers would subtract from each other and if over the bottom, then the two numbers would subtract.

### Addition and Subtraction Drag and Drop Game

Drag numbers to the bottom half to add them together (when it goes green). Drag a number of the top half of another one and it will subtract the one you are moving from the static one (when it goes red). Click create to begin. Numbers are created in order, starting with 1.



This game may sound trivial, but it helped me to understand how drag and drop worked, as well as understanding how items can react differently on a screen depending where the mouse is released. Most of this code is carried forward and used in the main Natural Deduction game.

## 5.2 Drag and Drop

Because I had no idea how to implement a drag and drop system, I followed an online tutorial and expanded my ideas from these foundations. <http://html5.litten.com/how-to-drag-and-drop-on-an-html5-canvas/> This was the tutorial that was used for both the initial addition game and also for the main software project.

The three functions created in drag and drop are all event based. These are when the mouse is clicked down, when the mouse is moved and when the mouse is released. All three functions have important roles and do different jobs that make the drag and drop successful.

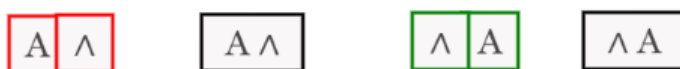
When the mouse is pressed down, an event handler runs, which runs a

function. The function iterates through all of the objects on screen, seeing if the location of the mouse click is in the same position as any of the objects. If this is the case then a variable is set which says that an element needs to be moved. The moving mouse function 'myMove' is always executed, but the logic within it is only run when an object on the screen is selected to drag and drop because the variable is set. The logic within the moving function went through different phases of development as the project progressed. Explained in detail below is both the legacy system and the new, improved way that it now works.

### 5.2.1 Original Colour System

When implementing the move function in the software, thought had to be given to how the system would react when items were dragged on top of each other. Original thinking was to base what happened to objects based on the location of other objects. Because of the tree structure, it naturally splits into four different parts. Top left, bottom left, top right and bottom right are all possible locations where a part of a proof can be located. The original system worked out the location of all the objects on the screen. If another object was dragged over one of the objects, then depending which corner it was dragged over, the algorithm will set the borders of the two objects of a certain colour. Different colours would be set depending on what type of object is being highlighted.

Variables and conditionals can be dragged onto by other objects of the same type. In the figures below, dragging to the conjunction onto the right hand side of the 'A' lights up both of the objects' borders red. Dragging the conjunction to the left hand side of the 'A' lights both of the borders green.



The same logic applies to proof structures as shown in the figure below, but all four of the corners could be dragged over to give different coloured bordered. Releasing the mouse when the borders are highlighted removes the object that is being dragged and places whatever is on the moving object to the static element in the correct corner.



This makes it clear which part of the proof the user is dragging to. This is the advantage of this method. Colours makes it clearer to the user and it means once a user is familiar with it, they will know what to do in future. This is also a disadvantage because it does not come across as intuitive because the colours are unknown to the user before they start the game. There are also no obvious points on the proof where the user can drag to, and without any kind of prompt it can be difficult about the possible valid moves available.

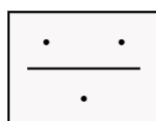
Another issue occurs when multiple layers of proof are dragged together. This system of associating colours with the area user drags to does not work intuitively when building proofs. Natural deduction proofs are often built from the bottom up. With this method, proofs cannot be built from bottom up because the dragging and dropping is all associated with the first object that is dragged onto. Any new proof dragged onto the top left or top right of the proof will remove the existing structure that is there. This leads to a peculiar way of forming proofs of multiple layers. Because of these reasons and reacting to feedback echoing this, the way users dragged onto objects was changed.

### 5.2.2 Dots System

A redesign of the system made the software fulfil more of the initial requirements, so it was a positive step, even if it was more technically more difficult. The idea behind this was to make it clearer to the user what moves were valid. The goal of the new system was also to provide a more flexible way of creating proofs, where users could drag onto any part of proof that was missing. This was achieved by adding dots to the proof structure.

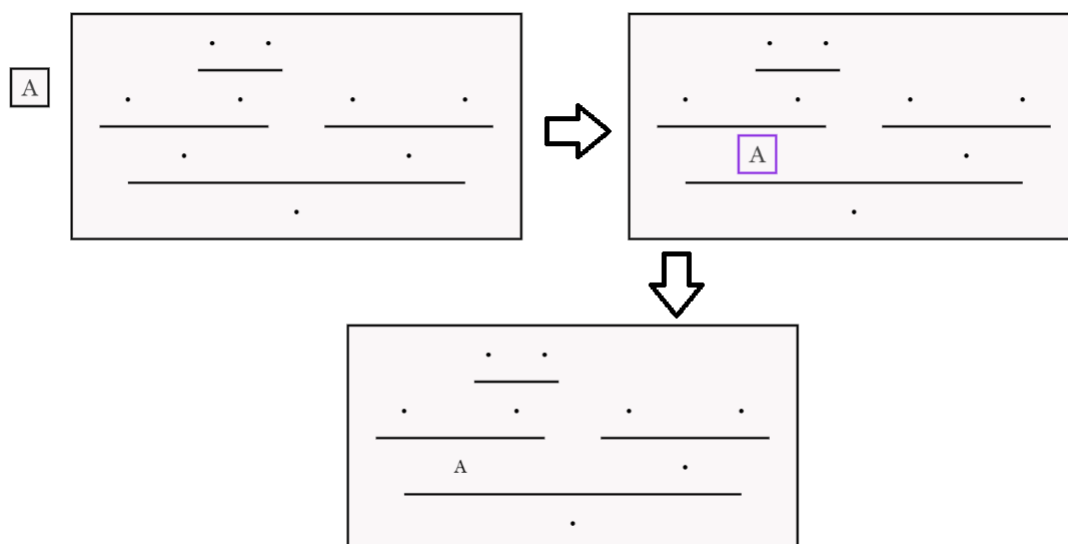
Dots automatically show to the user where they can drag onto on the proof. It makes it much clearer visually and it is more intuitive. Because now every dot in the proofs could be dragged onto, locations of each dot needed to be stored. This meant that every dot had to be a separate object with its own unique ID number and location information. These ID numbers would link to every proof object, so each proof object that can be dragged and dropped knows which dots are associated with it.

Also in the dot object is the colour of the border. If one of the objects is dragged over the location of the dot, then both the dot and the object being dragged over will turn purple. This is to let the user know which elements are being highlighted and what is going to happen if the user releases the mouse. This is a great improvement on the previous system and will reduce errors from the user because it is less likely that the user will make mistakes.



Another improvement from the first iteration of the code is that this system is more natural in the way proofs are created. Proof structures can be dragged onto the dots at the top of the current structure which leads to building a proof from bottom up. This is the conventional way to create Natural Deduction proofs in the tree structure and will offer a more realistic experience to the user.

In this way of dragging and dropping, the border of the object does not light up when the user is dragged onto. Only the dot of the static object lights up. Dragging to any corner of the object no longer has any effect, it is all based on where the dots are located. Because of this, multiple layers of proof can be built up with lots of dots contained in them. The user can then individually drop variables or statements onto the dots. This was something that was not possible in the previous way of dragging and dropping.



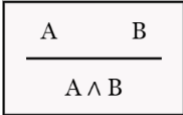
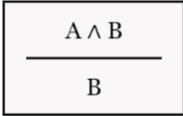
When another structure is placed on top of a dot, the dot object is removed and replaced with the structure that the moving object has, whether that is another proof structure or a statement. This structure of dragging and dropping proofs solved both of the main issues encountered in the original colour system. This is based on continuous internal testing as well as constant dialog with interested stakeholders such as the project supervisor Willem. Constant testing and improvement is a key part of any software project and will be covered in further detail later.

### 5.3 Data Structures

Due to the visual representation of the proofs as trees, storing the data in a similar way made sense as it made it scalable. Basic objects are stored as a list. Each draggable object structure was the same, whether it was a proof structure or a variable. The core structure had four elements. The first element represented the top left corner, the second element represented the top right corner and the third and fourth elements represented the bottom left and right corners respectively.

If an object was just a statement, like a variable, connector or a mixture of these, the string representing the statement would be placed in the first element and the remaining three elements would contain “%”. A percentage string was one that let the program know that this element will never need to be used for this part of the proof. For the statement  $A \wedge B$ , the data structure would be represented as  $[“A \wedge B”, “\%”, “\%”, “\%”]$ .

These percentage strings were also used for determining the layout of a tree structure. Two elements on the top and one on the bottom is used as well as one element on the top and one on the bottom in this game and they are represented with the following data structures:

2 Top / 1 Bottom	$[“A”, “B”, “A \wedge B”, “\%”]$	
1 Top / 1 Bottom	$[“A \wedge B”, “\%”, “B”, “\%”]$	

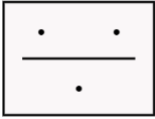
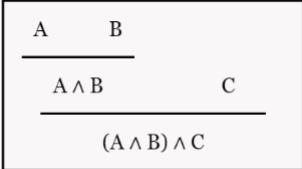
When dots are represented in the proof, the ID number of the dot number is the element in the proof structure. The drawing algorithm will recognise a number and render it in a dot. The ID numbers of the dot will link to a dot



and the dot object with the position coordinates associated with it.

The way larger proof trees were stored was in the form of nested lists. If an element of the list is stored as another list, then a proof structure of the same format can be contained there. This system works recursively to create proofs in tree shapes, which can then be easily be drawn by the software.

Proofs with dots and multiple layered proofs are represented as follows:

Dots in a proof	[1, 2, 3, "%"]	
Multi-line Proof	[[ "A", "B", "A ∧ B", "%"], "C", "(A ∧ B) ∧ C"]	

A simple, unified proof system allows both ease when drawing and when assessing the proof to check whether it is valid. Because the structure is consistent, comparison is possible.

## 5.4 Unification Algorithm

An important requirement for this software project was to recognise whether a proof is valid and correct. In Natural Deduction logic you can't automatically use distributive, associative or communicative properties automatically. Laboreo (2005) Every proof needs to be shown explicitly. This means that Natural deduction proofs can only be reordered where the substance of every branch is the same, but the location in the proof can be different.

As a simple example consider  $A \wedge B$ . Having both A and B on the top line states that A is true and B is true. Whether the A is on the left or the B is on the left is irrelevant but this shows that proofs can be represented differently. A whole proof tree can look very different whilst representing the same proof logically and this project provides an algorithm that achieves this.



The key part to this algorithm is that it uses recursion. The algorithm makes sure that a branch of the tree matches a branch of one, ideal solution that is provided. If this branch is correct then other branches are checked, again by recursion. If all branches in the user's answer are accounted for and match up with the ideal answer given then the solution is valid. If any one of the branches doesn't match up then the proof is obviously incorrect. It will return that the user's proof is incorrect and allow the user to try again.

By having this generic algorithm in place, it makes the system very flexible. Adding levels is incredibly easy because only one valid proof needs to be submitted and then all of the valid solutions will be permitted. This makes the software both flexible and easily scalable which are another two key requirements for this software.

## 5.5 Drawing and rendering algorithms

The drawing algorithm is another important part of the software to make sure that the proof is presented to the user in the way that they expect. Presenting information to the user correctly and reliably is a key feature that this project much achieve. The drawing algorithm uses `setInterval` which is a javascript function that runs a function at certain intervals. This runs the drawing algorithm that redraws what is on the canvas, giving a user an instant reaction to whatever moves they are performing.

The drawing algorithm was written in a way that it interprets the data structure to correctly render the proof in a tree shape. The drawing algorithm takes into account whether each structure is "2 Top/1 Bottom" or "1 Top/1 Bottom" and draws it appropriately. It also takes into account any dots in the system, which are represented as numbers in the proof system. The algorithm realises these numbers and renders them as a dot to the user.

Multi level proofs need to be drawn in a different location because they are on top of the first level of structure. Because of the potential different proof widths that need to be drawn, the first 4 levels of proof are individually accounted for. If more than four levels of proof need to be drawn, a recursive function is used. Another box linking to the proof is drawn in the same

format as the original proof. This is done for ease of readability. The user will still be able to read all parts of the proof easily.

In retrospect, a better drawing function could and should be created. One that offers the whole proof in one object that is still readable would be ideal. An algorithm that uses recursion more would reduce repetitive code. Tree like structures lend themselves well to recursion, so this could have been utilised more.

One feature that does work well is the resizing of boxes. When the height of a proof is increased, the size of the box will automatically increase as well. As the height increases, the width will also increase proportionally as well. This is to make sure that items on the top row on the proof will still be rendered clearly and that the user has a good user experience.

## 5.6 Levels and how the Software was turned into a Game

Although making sure the mathematic logic was complete and works successfully, it is just as important for this project to come across as a game to the user. There are many tools already available that will aid a user in learning Natural Deduction, but as previously discussed, many of these do not engage the user and lack other key elements that make these tools into games. Part of the implementation of this project was putting in key gamification attributes that would make this tool a game.

One of the main features that was important to include was the concept of levels. The implementation of levels in this game restricts the user from using certain elements and this helps the user by guiding them in what buttons they can press. In the Figure below, which represents Level 1 of the software, only the buttons needed to create  $A \wedge B$  are eligible to press. Buttons unavailable are greyed out and when the user hovers the mouse over the button, a 'not allowed' symbol is displayed.



Free play offers the user all of the buttons available but has no end goal for the user. A user can still drag and drop everything but the proof cannot be compared to anything. Clicking to check the proof runs the unification algorithm previously mentioned. This will either end the level and start the next one if the user is correct. If the user is wrong, it will let them try again.

The penalty for letting the user trying again is a reduction in a game score.

A user gets score of 10 for every level successfully completed with no mistakes. Every mistake they make- that is when they click 'Check Proof' and the proof is incorrect- is a deduction of two points. A user successfully completing a level correctly will always receive 1 point, no matter how many attempts are taken. Scores for all the levels are accumulated and a final score for a user is given.

The user can then play the game again and although it will be the same levels, the user can keep trying until they get a perfect score of 10 for every level. This makes the game more replayable and will hopefully subconsciously solidify the knowledge the user is learning about Natural Deduction. Scoring adds a competitive element between friends as well. Beating a peer may mean that a user has to play through the game again to get a better score, all of which will increase learning when the user plays through the game again.

By using levels and a scoring system, instant feedback can be relayed to the user. The user will know if their proof is correct as soon as they are ready. This gives the user feedback on demand when they want it. As well as this, further help can be provided via a hints button that has been implemented. This is to aid the user in the learning process. Clicking the hints button reduces the score by 2 points, so it encourages the user not to use the hints, but they are available if needed.

## Chapter 6

# System Testing

## Chapter 7

# Results

## Chapter 8

# Conclusion

# Bibliography

- Arthur, R. T. (2011), *Natural Deduction: An introduction to logic with real arguments, a little history and some humour*, Broadview Press.
- Chen, N.-S., Wei, C.-W., Wu, K.-T. & Uden, L. (2009), 'Effects of high level prompts and peer assessment on online learners' reflection levels', *Computers & Education* **52**(2), 283–291.
- Dempsey, J. V., Driscoll, M. P. & Swindell, L. K. (1993), 'Text-based feedback', *Interactive instruction and feedback* pp. 21–54.
- Garris, R., Ahlers, R. & Driskell, J. E. (2002), 'Games, motivation, and learning: A research and practice model', *Simulation & gaming* **33**(4), 441–467.
- Gasquet, O., Schwarzentruher, F. & Strecker, M. (2011), Panda: a proof assistant in natural deduction for all. a gentzen style proof assistant for undergraduate students, in 'Tools for Teaching Logic', Springer, pp. 85–92.
- Gentzen, G. (1964), 'Investigations into logical deduction', *American Philosophical Quarterly* **1**(4), pp. 288–306.  
**URL:** <http://www.jstor.org/stable/20009142>
- Halbach, V. (n.d.), 'Introduction to logic: Natural deduction', <http://logicmanual.philosophy.ox.ac.uk/vorlesung/logic6p.pdf>.
- Indrzejczak, A. (n.d.), 'Natural deduction', <http://www.iep.utm.edu/nat-ded/>.
- Jaśkowski, S. (1934), *On the rules of suppositions in formal logic*, Nakładem Seminarjum Filozoficznego Wydziału Matematyczno-Przyrodniczego Uniwersytetu Warszawskiego.
- Koster, R. (2013), *Theory of fun for game design*, " O'Reilly Media, Inc."
- Laboreo, D. C. (2005), 'Introduction to natural deduction', [https://homepage.univie.ac.at/christian.damboeck/ps06/clemente\\_nat\\_ded.pdf](https://homepage.univie.ac.at/christian.damboeck/ps06/clemente_nat_ded.pdf).



- McDonald, M. & Hull, C. (2012), 'Different ways of learning'.
- Montemayor, E., Aplatén, M., Mendoza, G. & Perey, G. (2009), 'Learning styles of high and low academic achieving freshman teacher education students: an application of the dunn and dunn's learning style model', *University of Cordilleras* **1**(4), 1–14.
- Muntean, C. I. (2011), Raising engagement in e-learning through gamification, in 'Proc. 6th International Conference on Virtual Learning ICVL', pp. 323–329.
- MyMaths Website* (n.d.), <http://www.mymaths.co.uk/>.
- nptelhrd (2015), 'Mod-01 lec-42 natural deduction in predicate logic', <https://www.youtube.com/watch?v=mJl9t1Pz5TU>.
- Ofcom (2015), 'The uk is now a smartphone society'.  
**URL:** <http://media.ofcom.org.uk/news/2015/cmr-uk-2015/>
- Oprescu, F., Jones, C. & Katsikitis, M. (2014), 'I play at work—ten principles for transforming work processes through gamification', *Frontiers in psychology* **5**.
- Pelletier, F. J. (1999), 'A brief history of natural deduction', *History and Philosophy of Logic* **20**(1), 1–31.
- PhilHelper (2013), 'A crash course in formal logic pt 8a: Natural deduction in propositional logic', [https://www.youtube.com/watch?v=ZebpxExsY\\_0](https://www.youtube.com/watch?v=ZebpxExsY_0).
- Pritchard, A. (2013), *Ways of learning: Learning theories and learning styles in the classroom*, Routledge.
- Sims, Z. & Bubinski, C. (2013), 'Codecademy website', <https://www.codecademy.com/>.
- Suhonen, K., Vääätäjä, H., Virtanen, T. & Raisamo, R. (2008), Seriously fun: Exploring how to combine promoting health awareness and engaging gameplay, in 'Proceedings of the 12th International Conference on Entertainment and Media in the Ubiquitous Era', MindTrek '08, ACM, New York, NY, USA, pp. 18–22.  
**URL:** <http://doi.acm.org/10.1145/1457199.1457204>
- Sutcliffe, G. (n.d.), 'Automated theorem proving', <http://www.cs.miami.edu/~tptp/OverviewOfATP.html>.
- UKIE (2014), 'The games industry in numbers', <http://ukie.org.uk/research>.

Wu, P.-H., Hwang, G.-J., Milrad, M., Ke, H.-R. & Huang, Y.-M. (2012), 'An innovative concept map approach for improving students' learning performance with an instant feedback mechanism', *British Journal of Educational Technology* **43**(2), 217–232.

Zichermann, G. & Cunningham, C. (2011), *Gamification by design: Implementing game mechanics in web and mobile apps*, " O'Reilly Media, Inc."