Logic Circuit Builder

Samuel Luke Williams – C1526328

Prof. David W Walker & Dr. Philipp Reinecke

BSc Computer Science

One Semester Individual Project – CM3203

School of Computer Science and Informatics

Abstract

This project sought to develop a software environment in order to allow users to build computer logic circuits from simpler circuits. The environment needed to include a graphical user interface where users are able to drag, drop and connect logic gates to build a complex circuit. The environment also needed to offer the ability to save the circuit, load a previously created circuit and generate a truth table from the circuit. The development process from start to finish has been documented within this report, along with the research and testing on the final system.

The outcome of this project was successful with a web application being created which contains a graphical user interface and many features such as saving a circuit, loading a circuit, generating a truth table as well as many more.

Acknowledgements

I would like to take this opportunity to thank my family, namely my mother, for the encouragement and support I received throughout this project. I would also like to thank my supervisor, Prof. David W Walker, for his guidance and constructive feedback.

Table of Contents

List of Figures

List of Tables

1. Introduction

1.1. Overview

The aim of this project is to develop a software environment for building logic circuits from simpler circuits. By taking logic gates consisting of NAND, NOR, NOT, AND, OR, and XOR the graphical user interface should allow a user to select a gate to be dragged, dropped and connected within a workspace to build a more complex circuit. The user should also have the ability to store and access built circuits to build yet more complex circuits. Finally, the software environment should be able to generate truth tables for the constructed circuits.

The purpose of this report is to give a detailed description of the design and implementation, background of the project and outcome of the project along with any future work, finally concluding with a reflection on learning throughout the project development. The remainder of this section will give a general introduction to the project.

Although the intended audience for this software environment is niche, there is a tremendous amount of usability, reusability and effectiveness of the product for the targeted audience. By creating a usable and helpful logic circuit builder, it will allow for the user to view a visual representation of a logic circuit in which they can control and manipulate to produce a desired truth table. Since it is understandable that someone without an interest in computer science or logic circuits would ever have a need for this software, it is aimed at targeting students who are looking to study computer science or related fields at a university level, first year undergraduate students studying computer science or related fields at a university level, and people with an interest in logic circuits or computer science looking to improve or test their knowledge on logic circuits.

The scope of this project is at a larger scale than it may seem, while creating an algorithm that can process the logic circuits efficiently, store the circuits and load circuits from a file, it is also required to design and create a graphical user interface that is simple and intuitive for a new user. With an efficient algorithm and a simple intuitive design, it should allow for a user to generate a truth table from a complex circuit that would otherwise take a considerable amount of time without the aid of this software.

1.2. Project Motivation

When beginning this project my understanding of logic circuits, and logic gates in general, was very good. I understood how to produce a truth table from some fairly complex circuits and how they operated, due to my 3 years of studying Computer Science at a university level. As a result of this I felt very confident that developing an algorithm to produce the intended outcomes would not be a problem, I had worked with java several times on previous projects, although once project development began I decided that JavaScript and HTML would suit this project far better (see section 3), of which I also felt very comfortable working with and had used on many projects previously.

In addition to creating a complex algorithm for processing these logic circuits, much of the work required also focused very heavily on the graphical user interface design. My knowledge within the field of graphical user interfaces was casual, I had worked with some in the past

such as the most commonly used “GuiProgramming” toolkit for Python, called TkInter [16], and Balsamiq [4] for designing graphical user interfaces. I had not created a fully interactive interface such as this one that would allow components to be dragged, dropped and connected, but I was very excited to research this area and get a better understanding of interactive graphical user interfaces. By having an understanding of the algorithm required and a motivation for understanding how the graphical user interface could be designed to fit this project, coupled with my interest I already had in design and technology which I had studied during high school, I was confident I could develop a good solution.

2. Aims and Objectives

Initial Brief:

"The aim is to develop a software environment for building computer logic circuits from simpler circuits. At the lowest level the simplest circuits are NAND, NOR, NOT, AND, OR, and XOR gates. The environment must include a graphical user interface allowing circuits to be dragged, dropped, and connected to build more complex circuits. These circuits can then be stored and subsequently used to build yet more complex circuits. The environment should also *be able to generate truth tables for circuits that are constructed."*

With the main brief in mind and the intended audience for the project decided, I was ready to design and create this interface, but jumping from nothing to a software that meets the expectations would not be possible without setting aims and objectives first which would collectively produce the expected product.

In order to do this I split the project into 4 main aims giving each of these aims an acceptance criteria that would answer the main aim. Since the time frame for this project was around 12-13 weeks, I decided that implementing an agile/scrum methodology during the development of the project would be the most beneficial method to manage these aims, along with its success in previous projects I had worked on. With twelve weeks and 4 main aims, I decided to split each aim into a 2-week sprint period where I would achieve the aim within this two week period, this work plan would then allow for 2-3 weeks remaining to construct the final report, allow more time if any unforeseen circumstances arise, any external obligations such as exams, remove any bugs and add extra features to produce the ideal product as discussed below.

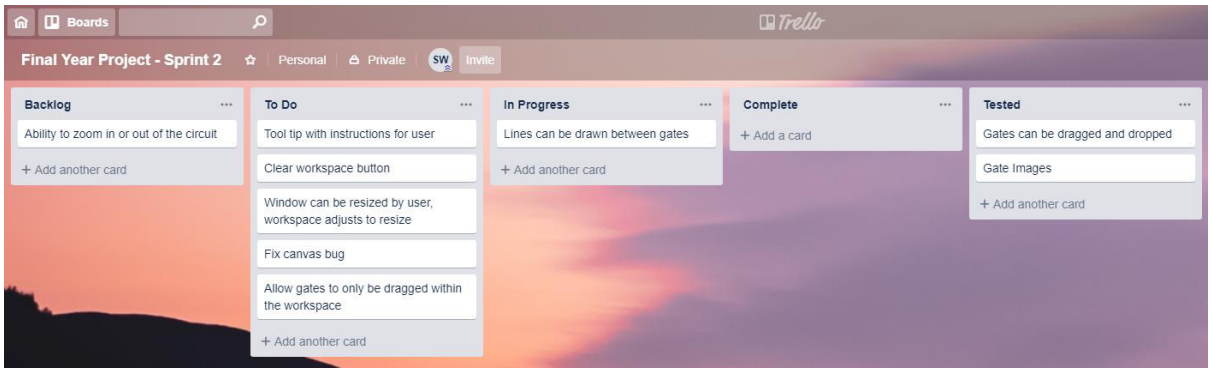


Figure 1 - Image of board from Trello

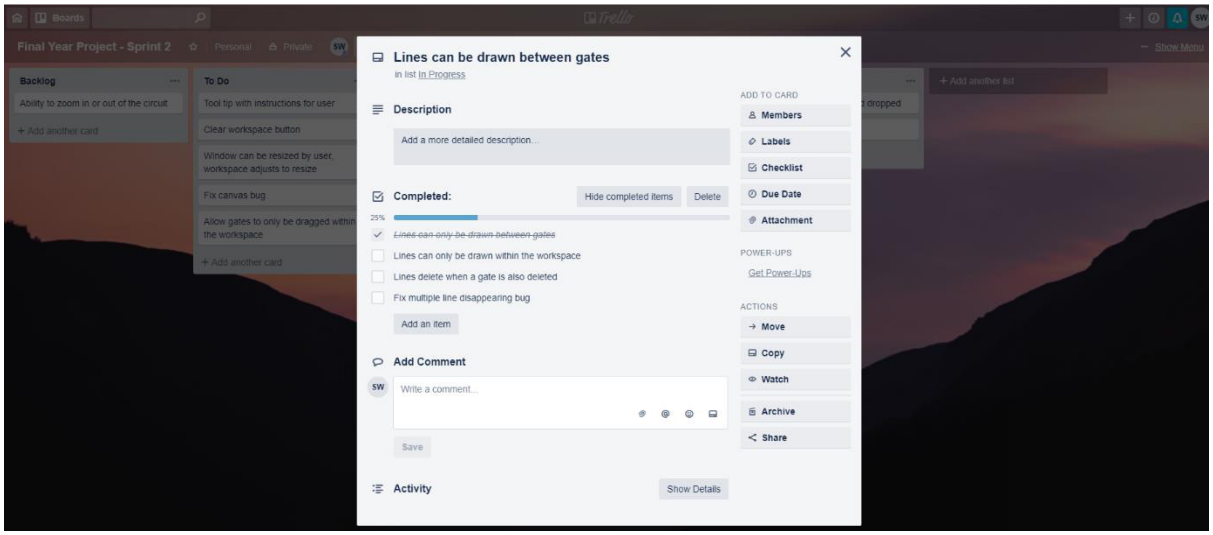


Figure 2 - Detailed expansion of card from board in Figure 1

With each sprint a new Trello Board [2] would be created which allows me to create separate lists labelled Backlog, To Do, In progress, Complete and Tested where each sub goal I created during the sprint would be moved along from to do, all the way until tested where there it was complete. Any goals that were not able to be complete within the sprint would be placed in the backlog and completed during the next sprint/when possible; more details on these labels can be found within the initial plan. This way of recording the goals for each sprint allowed me to split the result expected into three categories, the minimum viable product (defined as what we know we can produce in the time frame), the target product (defined as what I should be able to produce within the time frame) and the ideal product (defined as what we may be able to deliver within the time frame i.e. additional features). Below is an example of how each sprint aim would look:

Sprint 2: User interface allows for the logic gates to be dragged, dropped and connected inside a designated workspace within the interface. By allowing the logic gates to be easily dragged, dropped and connected we can offer an intuitive and simple interface for users that give them the freedom to control the components and produce a circuit of any form.

Date: 18th February 2019 – 3rd March 2019

Minimum Viable Product	Target Product	Ideal Product
- Window contains list of logic gates and a blank workspace where users will create the circuits.	- Users can drag, drop and connect the logic gates within the workspace. - Workspace can handle lots of components.	- Users can zoom in or out while also re-adjusting the size of the circuit and gates. - Help tab within the window which contains instructions on how to create a circuit and other resources.

Figure 3 – Image of sprint 2 taken from the initial plan

2.1. Main Aims

Aim: Software contains pre-defined logic gates that include at least NAND, NOR, NOT, AND, OR and XOR.

Acceptance Criteria:

- Logic gates should be easily accessible, and each gate should be labelled clearly.
- Symbols for each logic gate should be clear and large.
- List containing the logic gates should be scrollable/dropdown if all symbols do not fit in the window.
- Symbols should match the universal logic gate symbols.

Aim: User interface allows for the logic gates to be dragged, dropped and connected inside a designated workspace within the interface. By allowing the logic gates to be easily dragged, dropped and connected we can offer an intuitive and simple interface for users that give them the freedom to control the components and produce a circuit of any form.

Acceptance Criteria:

- Design should be simple and easy for a new user to create a circuit and generate a truth table.
- Users should be able to select a button that allows them to clear the workspace.
- Users should be able to resize the window.

Aim: Constructed circuits can be stored and easily accessed/loaded into the workspace.

Acceptance Criteria:

- Saved circuits should be saved into a certain file type, these file types can then be called by the software to load the circuit into the workspace. By saving as a file, users can obtain files from other users and load these into the workspace, allowing for different circuits to be shared and edited.
- When a file is loaded into the workspace, all logic gates and corresponding connections between these logic gates should also be generated.
- Loaded files should be editable.

Aim: Truth table can be generated from a constructed circuit.

Acceptance Criteria:

- Truth table should be displayed in a separate window in tabular form with the option of saving/exporting as a file for the user.
- Indication of which circuit corresponds to the truth table.

3. Background

With a large array of logic circuit builders existing, it was important that this one existed to solve a problem and stand out to the target audience in comparison to any of the products that already exist, and by researching already existing products and websites it became apparent very quickly what this problem would be.

3.1. Existing Products

Within the area of logic circuit building software there are many websites in particular that dominate, upon a simple google of "Logic Circuit Builder" you are greeted with the main circuit building websites that consist of CircuitVerse [10], ScienceDemos Logic gate simulator [9], Academo Logic Gate Simulator [1] and simulator.io [6]. Without searching deeper, or clearly stating it, it proves very difficult to locate a logic gate builder that is available for download and not hosted online instead. With the accessibility to the internet and the technologies available for creating such a website online, it is understandable that not many logic circuit builders exist as a downloadable software and although I had originally planned to develop this software within JavaFX, by instead using a language that could be hosted online far easier, I would be able to create a software that was far more accessible and far more efficient to access anywhere.

Upon further inspection of these online logic circuit builders, the same problems were arising with many of them, this was the ability to generate a truth table from the logic circuit that was built. Although many offered the ability to add a lightbulb/signal at the output to determine whether there was an output or not, I was only able to find one that allowed for a truth table to be generated from a user-built circuit. By creating the logic circuit builder I had designed and hosting online, I believe it extends the idea behind already existing products to provide a truth table generator, but which could eventually completely replace the already existing websites by incorporating any remaining features that the current project may not have. The reason behind why I do not believe the project I have designed and developed cannot yet replace already existing logic circuit builders, although many of it features are far better, is because of the current target audience. Another problem that I would attempt to resolve was the difficulty and the confusion caused by many of the graphical interfaces for these logic circuit builders, for someone with a very good understanding of logic circuits these pre-existing websites would be very useful as they provided many different additional components and input/output controls, but for the target audience I had identified previously, these interfaces would prove very daunting and complex, deterring many users from these websites who were not very familiar with these additional components provided. By creating a simple intuitive design I could create a software for new users that would quickly feel familiar and comfortable to use, this would allow me to keep the best features of these pre-existing products in a simpler intuitive way while also allowing the ability to add useful additional features such as a truth table generator.

3.2. Software

As discussed above, after researching existing products and identifying the problems that this project would attempt to solve, the software used had to be ideal for allowing ease of access to the product created, a simple intuitive graphical interface that would feel natural for a new inexperienced user, and an algorithm that computes the truth table efficiently. Below is the software used during development:

HTML, JavaScript & CSS: In order to make the logic circuit builder as accessible as possible for users, the ideal method was to create a webpage that could be hosted online. By using these two languages, I could easily create an interactive interface that contained all the features required, this webpage could then be accessed on almost any browser from anywhere without the requirement of any additional software or language to be installed onto the device. The HTML and CSS would allow for the design of the webpage to be created, while JavaScript would then be used to create the back-end of the website where it would deal with processing the truth table, storing the circuits and loading the circuits already saved as well as any other features that felt necessary. Through using these languages in a combination, it would allow for the site to operate within a single file (excluding the CSS) without the need for any additional support from external software. Despite this, the graphical interface would require additional support from jQuery [19] since the dragging and dropping of components and ensuring their containment within the workspace would not be possible with simply JavaScript and HTML. jQuery is a JavaScript library that allows many of the things used within this webpage such as event handling, HTML document traversal and manipulation to become much simpler and fluid. During the development of this project HTML5, JavaScript ECMAScript 5 and jQuery Version 3.4 were the versions used. A Font Awesome [7] library was also used to place icons on various buttons within the system.

Sublime Text 3: This language editor and compiler was used during the development of this project for its clean, fast and functional code editing. Its highlighted code and built in features allowed for the algorithm to be tracked with ease. Version 3.1.1 of Sublime Text 3 [18] was used.

Google Chrome: When creating the project, much of the work done was on a home device and lab computers. Both these devices run Google Chrome as the primary browser and since this is the most used browser worldwide, accounting for over half of the web traffic [3], the system was developed primarily in Chrome Version 73.0.3683.103. Although the system was also tested on additional browsers.

When beginning this project I had originally intended on creating the software environment within Java by using JavaFX, a GUI library for Java SE that is intended for developing rich client applications. But after researching existing products it quickly became apparent that the accessibility to the logic circuit builder would be greatly reduced by becoming a client application that is required to be downloaded and installed before it can be used. As well as this, pre-existing products similar to this were very complex for the audience they were

intended for and lacked features such as the generation of truth tables, which almost felt necessary in any logic circuit building application. In order to overcome this problem, the aim of this project is to create a software environment for building logic circuits that will allow users to drag, drop and connect logic gates within a graphical user interface, which can then be stored and accessed to build yet more complex circuits. In addition to this, the user interface should be a simple intuitive design that is welcoming for the target audience and offers additional features such as the generation of truth tables from a user-built circuit.

3.3. Design Research

One large problem facing this interface and many interfaces like this one, is the abundance of white space on the screen that is created by the large workspace. Although this space is required for creating the circuit, for a user quickly browsing through several webpages, a webpage with vibrant colours, graphics and imagery may be far more welcoming for a user than a white page with neutral colours such as grey or black. Despite this, the project that is being created as of now is intended for academic & educational use, as well as a specific target audience who are seeking this type of software, meaning that the need to draw in users from a business aspect is not as strong of a requirement as it would be if this system were to become commercial in the future perhaps. As a result, when researching the design of this interface, the focus was prioritised towards the layout of the interface, rather than the imagery, animation and colours.

Contained within the interface are three major components, the workspace, the buttons to perform features and the list of logic gates. With the workspace requiring the most area, around a 70/30 split of the window was required between the workspace and logic gates & buttons respectively. This would be most effective if the split was vertical rather than horizontal, with the workspace being placed to the right of this split.

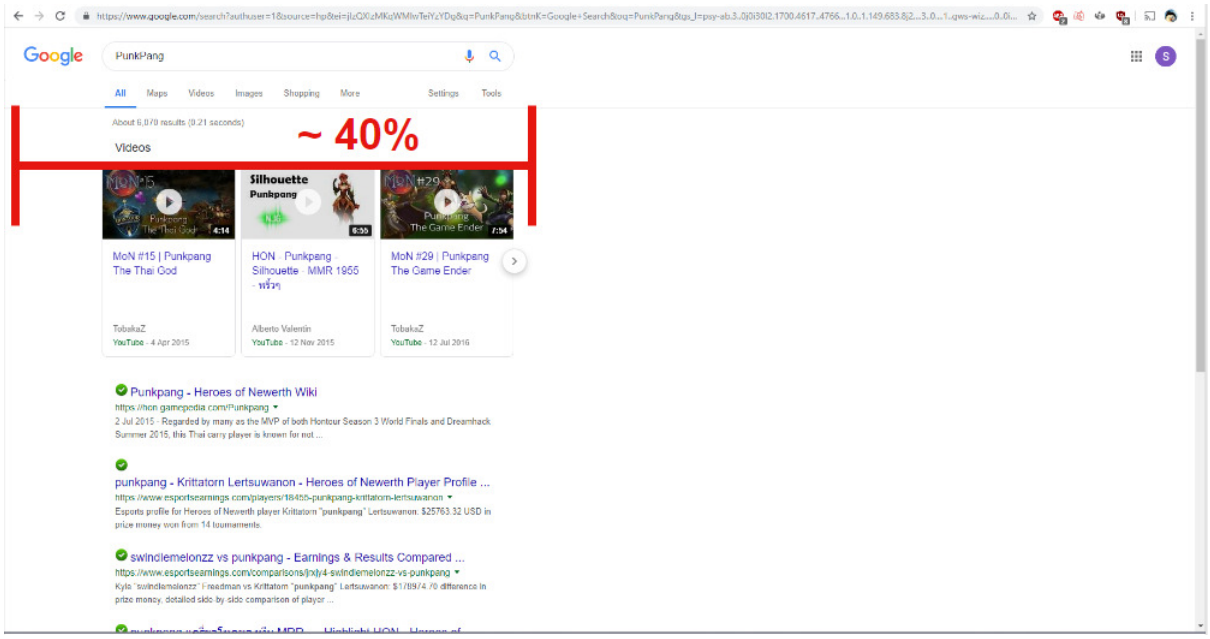


Figure 4 - Image displaying layout of google search result

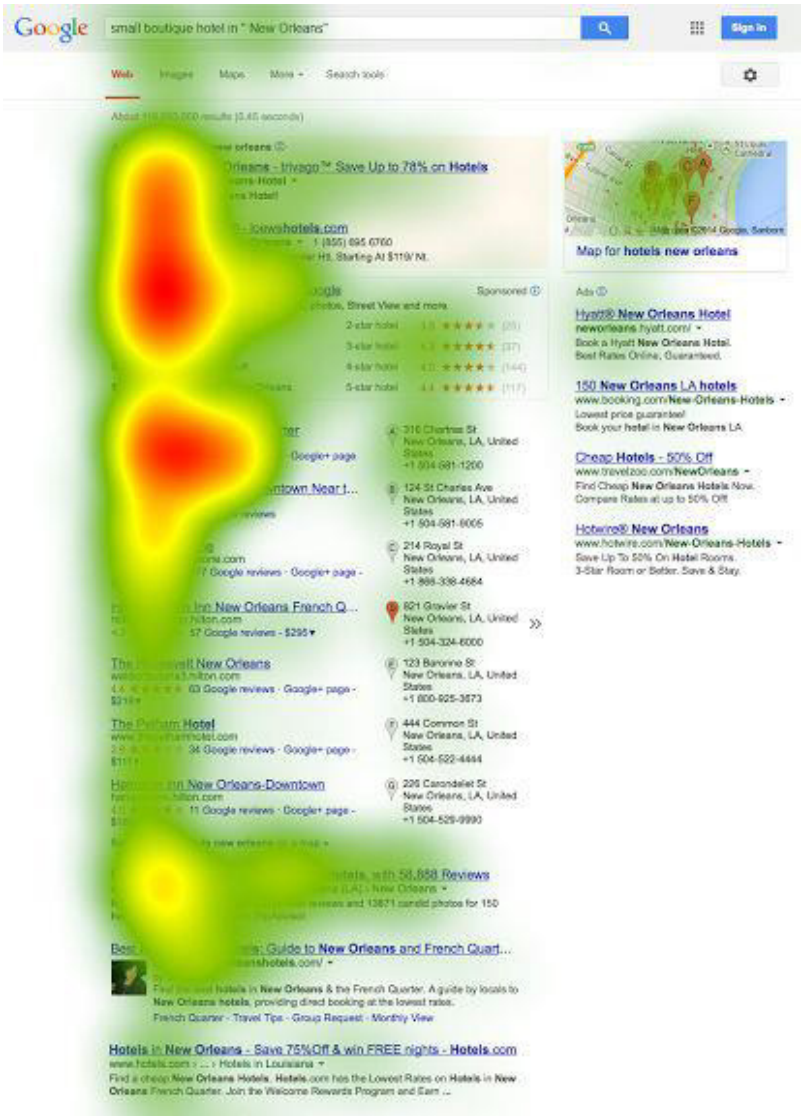


Figure 5 - Heatmap of user eye tracking when viewing google search result, obtained from Mediative Report [12]

The reason for placing the workspace to the right of the split is to allow the key components for building and processing the logic circuit to be placed on the left, since this is where most users tend to spend their time viewing a webpage. With recent research into the horizontal attention of web users, it has found that users spend 80% of their time viewing the left half of the page and the remaining 20% viewing the right half [8]. By using the information contained within this research we can prioritise the positioning of our key components to allow for a more efficient experience when building and performing features on the circuit, as well as ensuring the images of logic gates and buttons are what welcome a user upon opening the webpage. One of the best examples of the efficiency this layout produces is demonstrated in figure 4, this image highlights how a basic search result within Google will display all results within the left half of the screen to allow users to quickly locate the link they are attempting to find. This coupled with the research demonstrated in figure 5 show how web users eyesight tends to focus vertically on the left side of the screen, allowing for the layout of our project to be designed with the confidence that users will engage with the interface, regardless of the large workspace.

Although figure 5 contains a report discussing how the trend within user's eye tracking has changed from the original "Golden Triangle", it reiterates and supports my claims of user's horizontal attention leaning left.

3.4. Error Handling

Many of the problems that could arise within this project had been accounted for when developing the work plan, but once the project had been complete and the system implemented, errors could still occur. These errors would mainly occur due to user error when interacting with the system, as a result I ensured that during the design of the system these errors would be accounted for. Errors were first identified then given a likelihood rating along with a rating of the impact they would have on the functionality, these were rated using Low, Medium and High. Finally a solution to solve these problems would be provided where they could be implemented when developing the system.

Error	Likelihood	Impact	Prevention
Clicking an incorrect button.	Medium	Medium	By ensuring buttons within the interface are clearly separated and are clearly labelled with their intended function.
Dragging an incorrect gate or multiple.	Low	Low	Gate's must be separated evenly and labelled clearly along with their correct image.
Line not connecting to the gates within the workspace.	Low	Medium	Ensure that lines are clearly indicated whether they connect to gate or not. Could be done through only allowing a gate to be drawn when it is connected.
Accidentally deleting a gate or line within the workspace.	High	Medium	Ensure a user understands that they are about to delete a gate or that it cannot be easily done when constructing a circuit.
User accidentally clears the entire workspace.	Medium	High	Present the user with an alert if this option is selected, informing them that it cannot be done and anything within the workspace will be lost.

Table 1 - Table containing errors that could occur through user error along with their likelihood, impact and solution.

With many of the errors identified within the above table 1, although they are fairly likely to occur, many of them on average won't have much of an impact on the system. While some, such as clicking the incorrect button could have a large impact on the system where a user accidentally deletes their entire circuit, these errors have been caught further where a user will then be presented with an alert indicating what they are about to do. Since on average most users would accidentally select the "Clear Workspace" button but select cancel when they are

presented with the message, it won't have an impact and this is why the impact is only placed at a Medium, since this is the average.

4. Specification & Design

4.1. User Requirements

When creating the user requirements, it was important that the functional and non-functional requirements were separated, this would allow me to ensure that all the functional requirements were first met before advancing to include the non-functional requirements, and that they would only be implemented if the functional requirements had first been met. By taking the requirements created during the sprint planning in figure 3, many of the minimum viable product requirements were treated as the functional requirements, these ensured that the project met the original requirements of the brief, while the target product and ideal product requirements served as non-functional requirements that only offered benefits and allowed the system to be unique from any pre-existing products, although some of these were placed in functional requirements as they offered a lot to the functionality of the system.

Functional Requirements:

- The solution must contain a list of pre-defined logic gates that include at least NAND, NOR, NOT, AND, OR and XOR.
- The user must be able to drag and drop gates from the list into a designated workspace.
- The user must be able to connect these gates within the designated workspace in order to create a logic circuit.
- By clicking a button, the user must be able to store/download the circuit onto their device.
- The user must be able to load a pre-made circuit into the workspace by selecting it on their device.
- By clicking a button, the user should be able to generate a truth table from the circuit created or loaded into the workspace.
- System must generate a truth table within a reasonable amount of time, regardless of the size.
- User should be able to edit a circuit that is loaded into the workspace from a saved file.

Non-functional Requirements:

- Workspace should be able to handle lots of components/gates.
- Logic gate symbols should be clear, easily accessible and labelled clearly.

- Logic gate list should be scrollable/dropdown if all gates within the list do not fit within the window.
- When dragging the gate from the list into the workspace, a clone should appear in the mouse position, allowing the user to understand where the gate is being placed.
- On generation of the truth table, the inputs and outputs of the circuit should be clearly labelled and match the labels within the truth table.
- User should be able to delete a gate within the workspace, removing any connecting lines to the input and output of that specific gate.
- User should be able to select a button that clears the entire workspace.
- Truth table could be generated within a separate window, allowing user to compare the truth table with another or edit the circuit without losing the generated truth table.
- Circuits should be stored in a file format that is readable for users, such as csv.
- Users could have the option to save the truth table generated by their circuit.

4.2. Static Structure

After researching the designs and understanding how to create a simple intuitive interface, I began by creating various mock-ups of how the page would look. By offering a simple design it was important that users did not have to navigate around a site with many different pages, and that all the required information was presented within the single page. This meant that spacing was important, while the workspace had to be big and serve as the focus of the entire page, the list of gates and features such as saving the workspace and generating a truth table also had to be made easily accessible. When changing the design between iterations I would ensure that any changes made to improve the users experience were justified. Below are iterations of the design ideas along with justification:

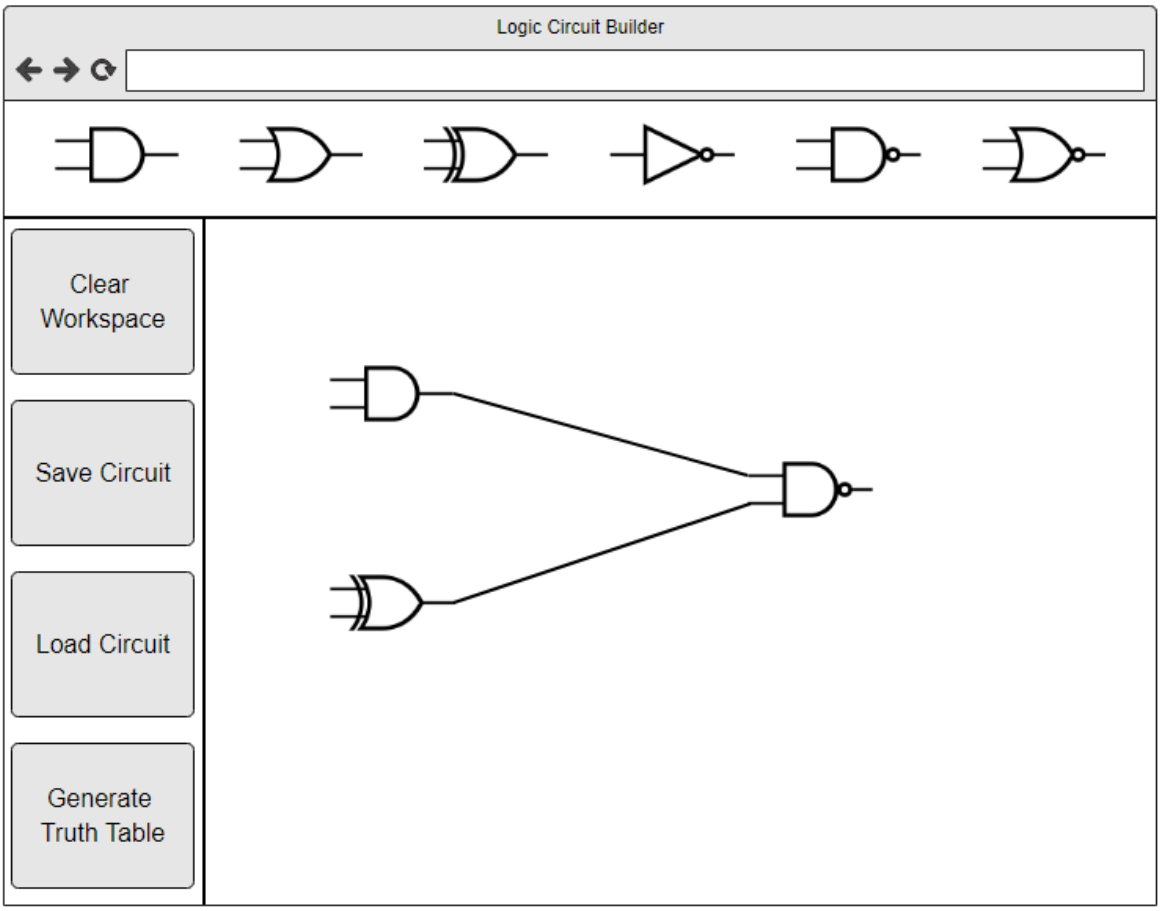


Figure 6 - First Iteration of website mock-up, made using moqups.com [13]

The above figure 6 shows the first iteration of the webpage, when designing this I had ensured that all the required components and features were included within the same page to avoid navigation around the website, offering efficiency to the user. At the top of the screen you can see the logic gates laid out horizontally and the buttons to perform features on the workspace laid out vertically below this. The rest of the webpage is filled with the workspace where the users can drag, drop and connect gates freely making it feel like almost the entire webpage is an area for them to design the ideal logic circuit. All these components are clearly separated by bold lines, helping indicate where the workspace begins and ends.

1st Iteration – Figure 6	
Pros	Cons
• Workspace is clearly indicated by bold lines containing it.	• Lots of workspace is wasted by large buttons. • Separate sections for gates along top and buttons down the side remove

• Workspace covers most of the webpage, offering a large area for users to use. • Buttons are labelled clearly with separation between to avoid user error when clicking buttons. • Simple interface that is not cluttered by unnecessary text or icons. • Bright off-putting colours are not used, colour theme remains consistent and doesn't clash.	some space from height and width of workspace. • Gates are not clearly labelled, could cause confusion for users who are not familiar with the gate symbols.

Table 2 - Table containing pros and cons from interface of Figure 6

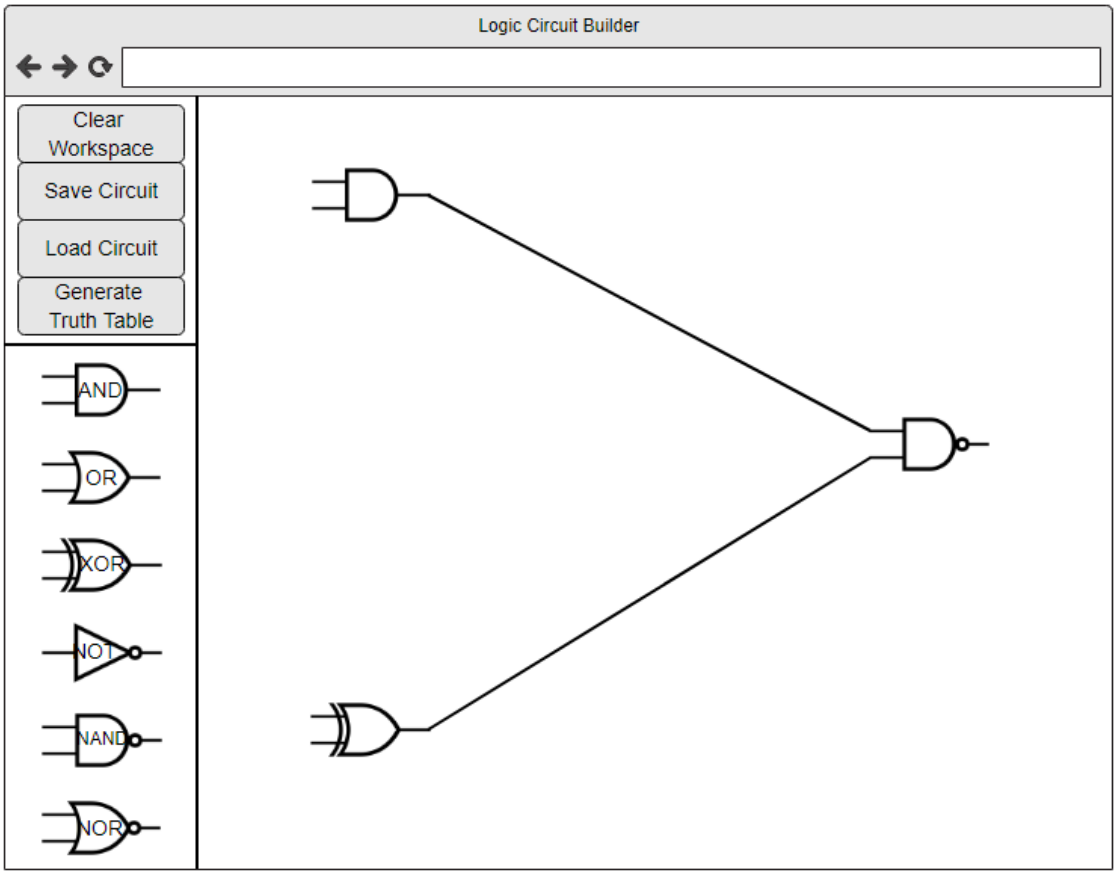


Figure 7 - Second Iteration of website mock-up, made using moqups.com

By the second iteration of the design, although many of the features had been included in the first page, there were still some major components that could be moved or added in order to create a far better interface. Firstly, the logic gates were moved from the top of the page, to the

left side where they were listed vertically. Separated by a line above the logic gates are the buttons, which have been reduced in height, and moved closer together in order to create more space for the logic gates below. By moving the logic gates to be listed vertically with the buttons, more height for the workspace has been created, raising the limit on the size of the circuit that would fit within the workspace.

2nd Iteration – Figure 7	
Improvements from previous iteration	Problems
• Larger workspace available for larger circuits to be built. • Interactive components such as gates and buttons, contained within one area for better accessibility. • Gates are labelled for users who may not be familiar with symbols alone. • Dragging gates from top of screen downwards would feel very unnatural. By changing the position of the gates to the left of the screen, a more natural movement of dragging the gate from left to right is produced, much like how we look when reading a book.	• Buttons are closer to allow gate list to fit. Accidental clicking of incorrect button could be caused as a result. • Gate label is not clear on all gates. Although the "OR" gate label is shown clearly, many of the longer words such as "NAND" may be difficult to read within the gate. By increasing size of gate to fit more clearly, gates will be pushed closer, causing some users to accidentally drag the wrong/multiple gates by mistake. • Label size on the buttons cause buttons to be larger than needed.

Table 3 - Table containing improvements from previous iteration and problems with interface in Figure 7

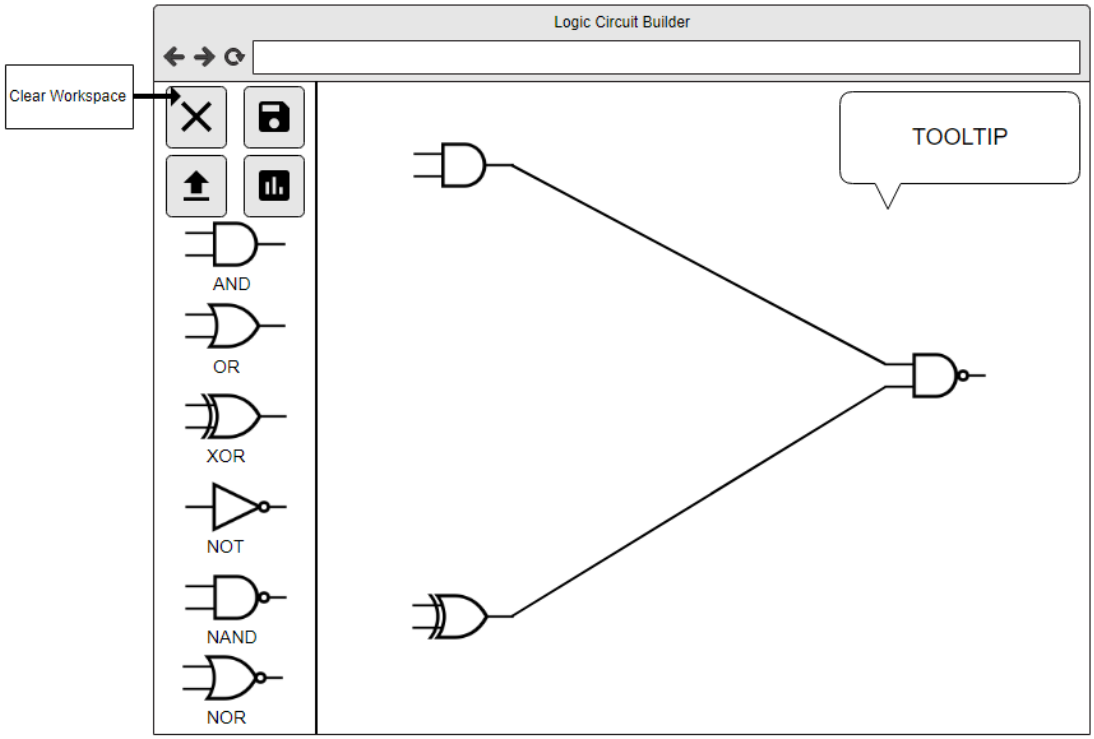


Figure 8 - Final Iteration of website mock-up, made using moqups.com

As you can see in the above figure 8, maximising area for the workspace while allowing the logic gates, their labels and the buttons to remain clear were the priority behind this design. First, I began by changing the buttons to be labelled by icons rather than text, this would allow the buttons to remain understandable while also greatly reducing their height and width from the first iteration. In order to also avoid any confusion or selecting the incorrect button, clear spacing is added between the buttons but also upon hovering the mouse over a button, a label will appear next to the button indicating the function of the button for the user (as demonstrated with the arrow on the "Clear Workspace" button at the top left of figure 8). With the change to the buttons reducing the area they occupy, this allows for the gates to be more evenly spaced along with their labels placed below to appear more clearly, in addition the line separating the buttons and the gates has been removed to reduce even more wasted area. While also improving previous iterations, additional features were added to the final iteration that had originally been planned but were not a possibility due to the lack of space available, but this was made possible by using similar features as the button, where a tooltip in the top right corner of the workspace would appear when the user first loads the page. This tooltip offers the basic instructions of how to operate the system for a new user, as well as additional features such as double clicking a gate to remove, which may not be as clear to some users. In order to avoid obstruction of circuits within the workspace, this tooltip will disappear upon hovering the mouse over it and its low opacity means a user can clearly place a gate behind it if they had not dismissed it prior to dragging a gate from the list.

4.3. Dynamic Structure

The dynamic structure & behaviour of this webpage is based around the interaction between the system and the user. To best represent this, a use case diagram has been used that demonstrates the interaction that would be expected between the user and the system by operating the user interface.

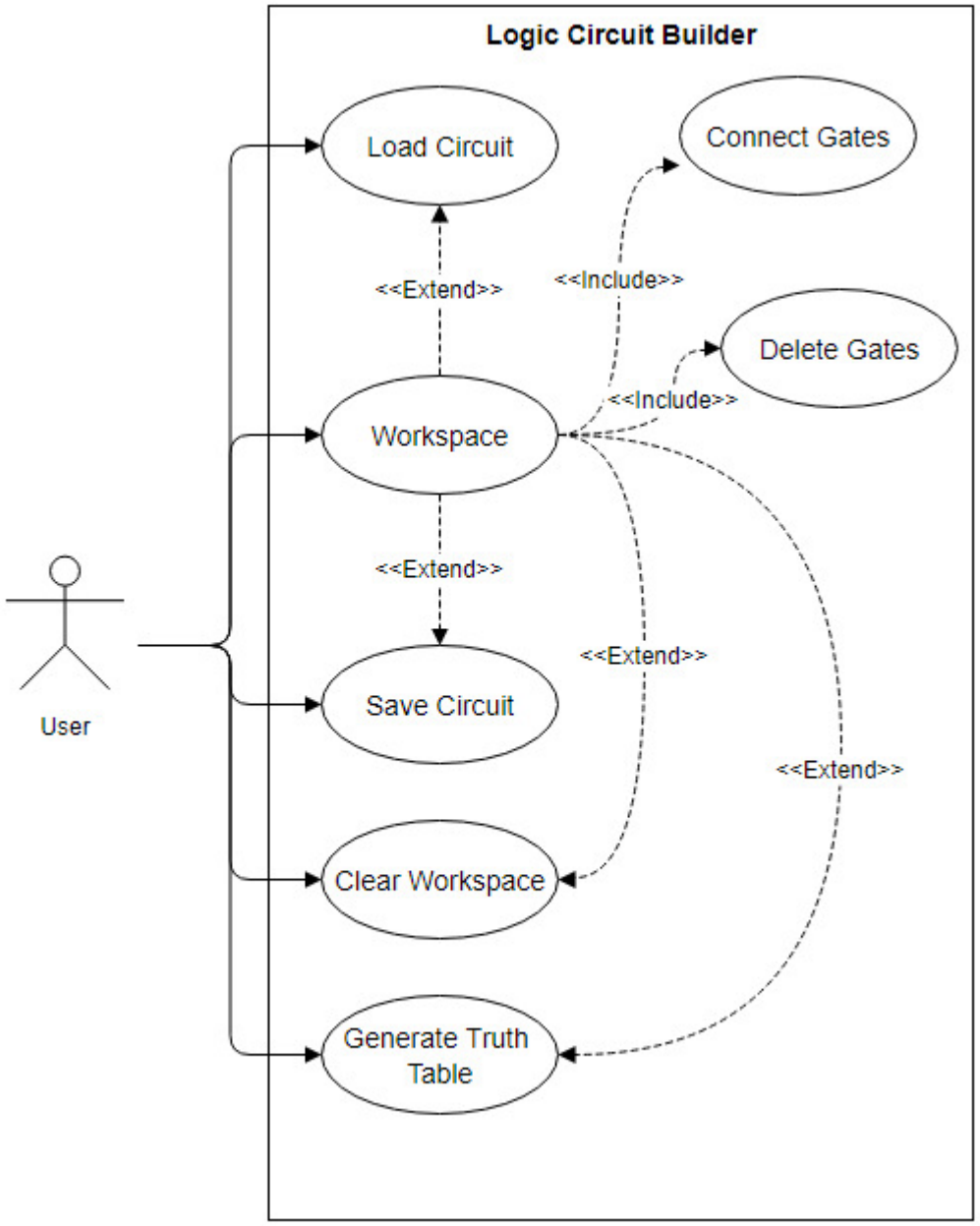


Figure 9 - Use case diagram demonstrating interaction between system and user

The use case diagram displayed in figure 9 demonstrates how the various features can be performed within the webpage, including which features extend and include each other. This diagram does not demonstrate how the webpage can be navigated since, as discussed in Section

4.2, the webpage is designed with the intention of all features being accessible from a single webpage.

4.4. System Design

In addition to designing an interface that solves the problem many similar systems had of being over complicated and daunting for new users, the system I was creating offered the large additional feature of generating a truth table. This meant that the algorithm behind the system would also have to be designed to ensure the information placed within the workspace by the user could be processed quickly and offer results within a reasonable time. In order to do this, any gates placed within the workspace, connected or even deleted would need to be tracked at all times, and by doing this would reduce the time needed when processing the truth table or saving the circuit, since the system would not have to retrieve the circuit and could simply process a representation of the circuit that was being updated as the user interacts with the interface. Below is a table demonstrating what data types different components in the system would be stored as upon being used within the workspace, and the reasons behind these decisions:

Component	Data Type	Reason
Logic Gates	String	By storing the logic gates with individual id's represented as strings, it allows for each individual gate to be differentiated from one another even if they contain the same operation. i.e. andGate0, andGate1 etc.
Logic Gate Positions	Integer	In addition to storing id's, the position of the gate within the workspace is important when saving and loading a circuit, as a result we store this as two separate integers representing X and Y respectively.
Lines Connecting Gates	String, Integer	When a user connects logic gates to one another, this connection must be represented by a solid line within the workspace. With each line representing a connection between two specific gates, the unique id of this line must also be stored as a string in order to retrieve it easily when generating the truth table or saving a circuit. Since the line is also being displayed for the user, the start and end position of the lines must be stored as integers representing "Xstart" and "Ystart" of the lines, and "Xend", "Yend" of the lines.
Generating Truth Table	Boolean	When generating the truth table, the input and outputs of each gate will be processed as true or false Booleans, since the built-in operators within JavaScript will allow for these values to be processed far easier and far more efficiently.

Table 4 - Table displaying the data types of components stored within the system along with the reasoning

By using the data types in the above table within the system, we can develop and implement an algorithm that stores these components within dictionaries and arrays (correct term is associative array but dictionary will be used for the purpose of this report to avoid confusion between the arrays), where dictionaries can be used when specific elements are required to be accessed while arrays will allow us to compute more efficiently if specific elements within them are not required to be accessed and instead only used when iterating through the entire array to store the circuit, or generating a truth table [15]. For example, when gates are placed within the workspace they will need to be stored within a dictionary with the gate id as the key and the gates position as the corresponding values. This will allow the algorithm to easily remove the specific gate and its position from the dictionary when a user deletes the gate from the workspace, instead of having to iterate through an array to locate that specific gate and its position. While on the other hand, during the generation of the truth table for a circuit, specific values will not need to be accessed and instead all the components within the workspace will need to be processed, and by using an array we can more efficiently iterate through all the stored components in comparison to using a dictionary.

4.5. Activity Diagram

Another method for capturing the dynamic behaviour of this system is through the use of an activity diagram [20] which will display how many of the features within the system will interact with the user's workflow.

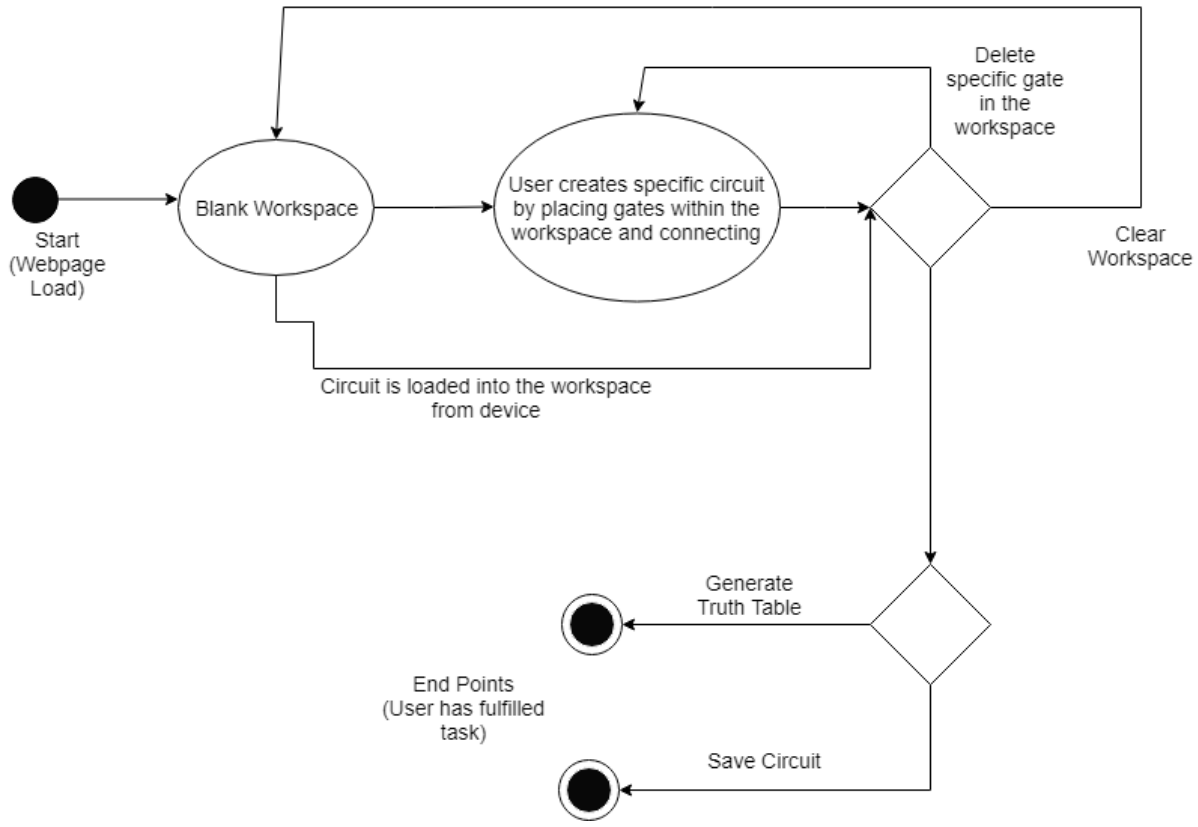


Figure 10 - Activity diagram of user interaction with the system

The activity diagram begins when a user loads the webpage since this is the where all users would begin regardless of their intended task, it also ends when a user completes one of the two specific tasks, generating a truth table or saving the created circuit. These are considered the end points within the activity diagram for a user since they cannot really progress further once these tasks have been completed, a user generating a truth table cannot use this truth table within the system for any further purpose while saving a circuit would mark an endpoint since any other tasks a user was looking to complete would have been done prior to reaching this end point. While these end points cannot be used within the same activity diagram again, the results from these end points can, once a user generates a truth table or saves a circuit they can begin the flow of the activity diagram again where the truth table can be used to compare against another truth table or circuit, or the file output from the save circuit end point can be used to load the same circuit back into the workspace.

5. Implementation

As discussed previously in the report, when beginning this project the software environment had originally intended to be within JavaFX, but due to the accessibility this offered compared to other languages such as HTML and JavaScript, the project was moved and developed with these instead. This was one of the first unforeseen circumstances that would arise during the development of this project, with around 2-3 weeks passed developing the project within JavaFX, the already limited time for the creation of this system was less. Luckily, problems like this had already been accounted for during the initial planning of the project with the final sprint being twice the length of any other sprint within the plan. This meant that time from this final sprint could be removed and placed within the first sprint instead, and since a lot of the time during the beginning of the first sprint had been spent researching JavaFX and understanding better, development of the interface had moved slower than expected, allowing the catch up within the new languages to progress much faster.

5.1. Interface

By beginning the implementation of the project with the interface, it would allow for a shell to be created that could then be filled by the algorithm. Constructing the interface would prove far easier than expected, with the mock-up created most of the interface was replicated by simply using HTML and CSS, along with additional libraries I had already mentioned. The gates were ordered into a list where each list item contained an image of a specific gate along with text labelling it, and placed into a division along with the buttons. This division would allow for these components to be separated from the workspace.

```
// Creates a clone of the gate being dragged for users to visualise, also allows gates in list to be draggable
$('#andDrag, #orDrag, #xorDrag, #notDrag, #nandDrag, #norDrag').draggable({
    containment:'#workspace',
    helper:"clone"
});
```

Figure 11 - Function used to allow gates to be draggable, as well as contained to the workspace division

To create the workspace a second division was formed, and differently from the original design a border was placed around the workspace using CSS, where by then using the containment argument as shown in the above figure 11 in jQuery, we could ensure that any gates dragged into this workspace would then be contained. By creating a border around the entire workspace rather than a single line to the left similar to the interface designs in section 4.2, it would allow the user to have a better understanding of where the workspace was contained to above, below and to the right, rather than just the left as the original design had shown.

Connecting the gates visually would prove far more difficult than expected, although section 5.2.3 will discuss how the system detected gates were connected, the user needed feedback to

show that they had been connected successfully. Creating lines within HTML is not difficult but allowing a user to draw their own lines is, and to do this a HTML canvas was placed within the workspace division. This canvas would be transparent and the identical size to the original workspace division border, where upon clicking within the workspace, the system would ensure that the point selected was within a placed gate, since lines were not allowed to be drawn unless they connected gates. If the detection of a gate was successful this point would be stored, where a second successful click with the same criteria would draw a line from the original gate to this one. This means that the gates placed and lines within the workspace are essentially at two different levels but appear to be contained within the same element for the user.

5.2. Algorithm

5.2.1. Storing Circuits

When developing the algorithm to store circuits it was always important that the circuit should be stored within a readable file format such as CSV. By creating a readable file format it would allow for users to view the source file of a circuit and understand the various attributes of that specific circuit, these could include things such as the numbers of gates used within the circuit, types of gates used within the circuit, gates that do not receive input from other gates and gates that do not output to any gate. A simple understandable file would also allow for users and myself to test that the circuit that loads into the workspace from this file is correct.

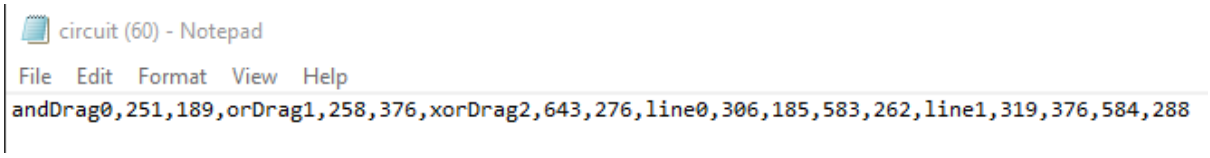


Figure 12 - Contents of a circuit file created by the system

As discussed in section 4.4, gate positions and line positions would be stored within an array or dictionary upon being placed or drawn within the workspace. This meant that once the saveCircuit function was called by selecting the button, these stored values could be iterated through and immediately written to a file. Since JavaScript cannot write or create files within the client computer for security reasons, a dynamic download file would have to be created named “circuit”, where a simple button click would be simulated causing the file to be downloaded and placed within the users download folder. This file contains CSV that include gates, their positions, lines and their positions. Gates are written to the file, then followed by the lines within the workspace, a typical entry for a gate within the file will look like this “GateID, X, Y,” where the GateID represents the specific gate id, this will be written with the operator first, followed by “Drag” and a unique number that increments each time a new gate is placed within the workspace i.e. “andDrag2”. The X and Y values represent the x, y coordinates of the centre position of that gate. Once all the gates in the workspace have been listed within the file, the lines contained will then also be written, these are formatted as

“lineID, startX, startY, endX, endY” where the lineID is the word line, followed by a count of the lines in the workspace i.e. “line1”. “startX” and “startY” represent the starting position coordinates x, y of the specific line, this starting position is the first position a user selects when drawing a line between two gates, these are then followed by “endX” and “endY” which represent the end of that unique line.

To summarise, the file downloaded by the user when the circuit is saved will contain an identical circuit to the one contained within the workspace, meaning a user loading this file into the workspace will be given an identical circuit containing the same inputs, outputs, gate positions and connections between gates.

5.2.2. Loading Stored Circuits

By giving circuits saved within files a particular structure, it allowed for loading of these circuits back into the workspace to be a far simpler process. Firstly upon clicking the “Load Circuit” button a user would be presented with a separate window, allowing them to select the specific file they wish to load into the workspace. Once selected, this file is retrieved and stored within an array, from here the system is able to iterate through the array beginning with the first element. Iterating through the array is not done one at a time for this specific array, since the structure is “GateID, X, Y,” or “lineID, startX, startY, endX, endY” depending on whether it is a gate or a line, we first locate the id which will be the element we start at when iterating through the array. By retrieving this id and verifying if it is a gate or a line the system can understand how the next few elements need to be dealt with (i.e. if it was a gate it would only need the next two elements to place the gate since these are X and Y where line would need four), from here the gate or line is placed within the workspace then the loop iterates forward a certain number of elements depending on the component that was just placed, where a gate would iterate forward three times and a line five; doing so we skip the coordinates that have been used and are able to begin again at the next id.

This process can be demonstrated by referring to figure 12, where the first element contains the id “andDrag0” meaning it is a gate, the system will then place this gate by retrieving the two elements after this element, “251” and “189”. Once this gate has been placed using the coordinates, our current position, which was on the first element, will be incremented by three meaning we are at the next id, “orDrag1” in this case. This gate will be processed again, and the position incremented by three, where it will place “xorDrag2” and increment by three again. Finally we will reach the element containing “line0”, since the system knows this is a line from the id, it will instead process the next four elements to get “306, 185, 583, 262” where upon drawing the line within the workspace it will increment by five to account for these coordinates and continue the process.

5.2.3. Connecting Gates Within The Workspace

Since the logic gates and lines within the workspace were essentially two separate layers (see section 5.1), an algorithm to detect the connection between the lines and the gates within the workspace would have to be created.

```
$('#workspace').click(function(e){
    var offset = $(this).offset();
    var mouseX = (e.pageX - offset.left);
    var mouseY = (e.pageY - offset.top);
```

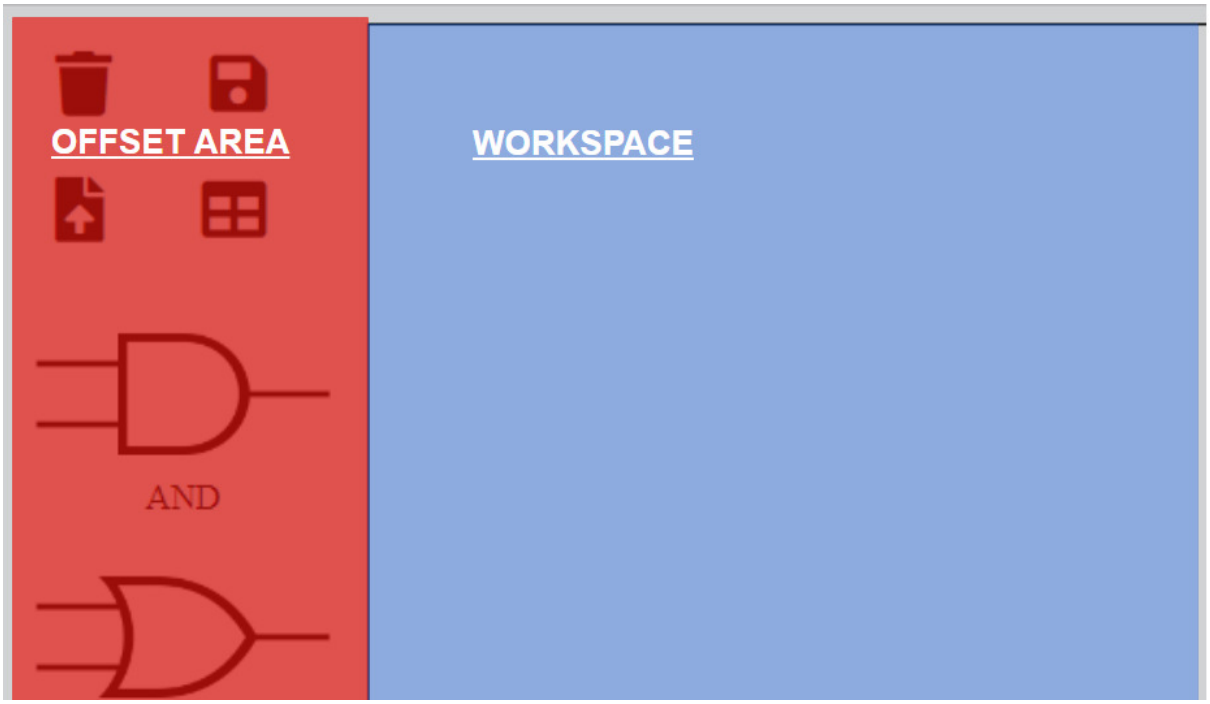


Figure 13 - Function used to calculate offset of mouse click (Top) and visual representation of offset area and workspace from the interface (Bottom)

To solve this problem a function was created that is only called by a click within the workspace. Since the function was contained to the workspace this meant that users would not be able to create lines that passed the boundaries of the workspace, in addition to this it only allowed users to create lines between gates, meaning any attempt to draw a line that wasn't connected to any gates would not be possible. In order to do this upon a click of the mouse within the workspace, the x, y coordinates of this mouse click are taken, and the offset of these clicks are calculated. As shown in figure 13, the offset is used in order to account for the space taken up by the gates and buttons to the left of the workspace, meaning the top left corner of the workspace division will become the 0,0 coordinates instead of the top left corner of the webpage. This will allow for gates to be stored in an accurate position within the workspace which will help in future with the loadCircuit function and the saveCircuit function, since this offset will not have to be accounted for in the future.

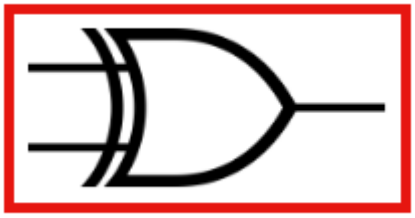


Figure 14 - Image of 'AND' logic gate with visual representation of bounding box

With the accurate coordinates stored, the function can now retrieve the context of the canvas within the workspace and then iterate through each gate within the gatesPlaced array, which represents each gate placed within the workspace by the user, and create a bounding box around the gate image. Since the workspace contains a canvas which is used to draw lines and the gate images are not stored within this canvas, by retrieving the image boundary and transferring this into a rectangle within the canvas we can replicate the area a gate would occupy within the canvas. Once the gate has been replicated we can use the function isPointInPath from the 2D canvas API [14], which returns a true or false value if the point is contained within a certain shape, to detect whether the mouse click we stored previously intersects with the replicated gate rectangle, if the click is contained within the rectangle we can store this click position ready to be drawn as a line once the second position is selected.

5.2.4. Generating a Truth Table

One of the biggest key features with this project was the production of a truth table from the user created circuit, this system would need to generate an accurate truth table regardless of the number of gates used, how they were connected or how many inputs the circuit received. Visualising the circuit within the interface was not a problem, and storing the gates & lines within this workspace was also not a problem, but connecting these different components was. When the user selected the button to generate a truth table, the system would have to understand where the circuit begins and where it ends, along with which gates the input flows through before reaching the output.

When the truth table generation function is called, it begins by creating a dictionary (again, known as an Associative array within JavaScript, although dictionary is used in this report for clarity between an associative array and a regular array) from the gates in the workspace, by taking all the gates within the workspace we can create a dictionary that is the same length as the gates in the workspace, ready to be populated. For each entry within the dictionary the gate id is assigned as the key were the corresponding value is [null, null, null], we use a three element array to represent values of each gate where the first two elements represent lines that connect to the input of the gate and the final element represents the lines connecting to the output of the gate.

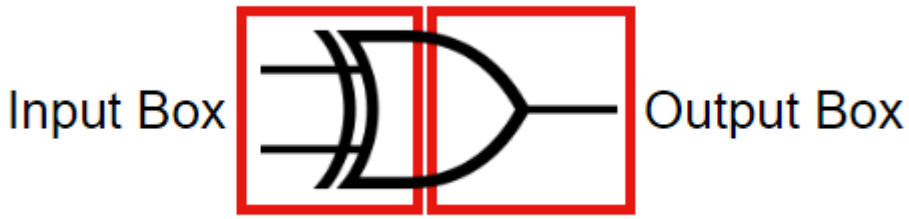
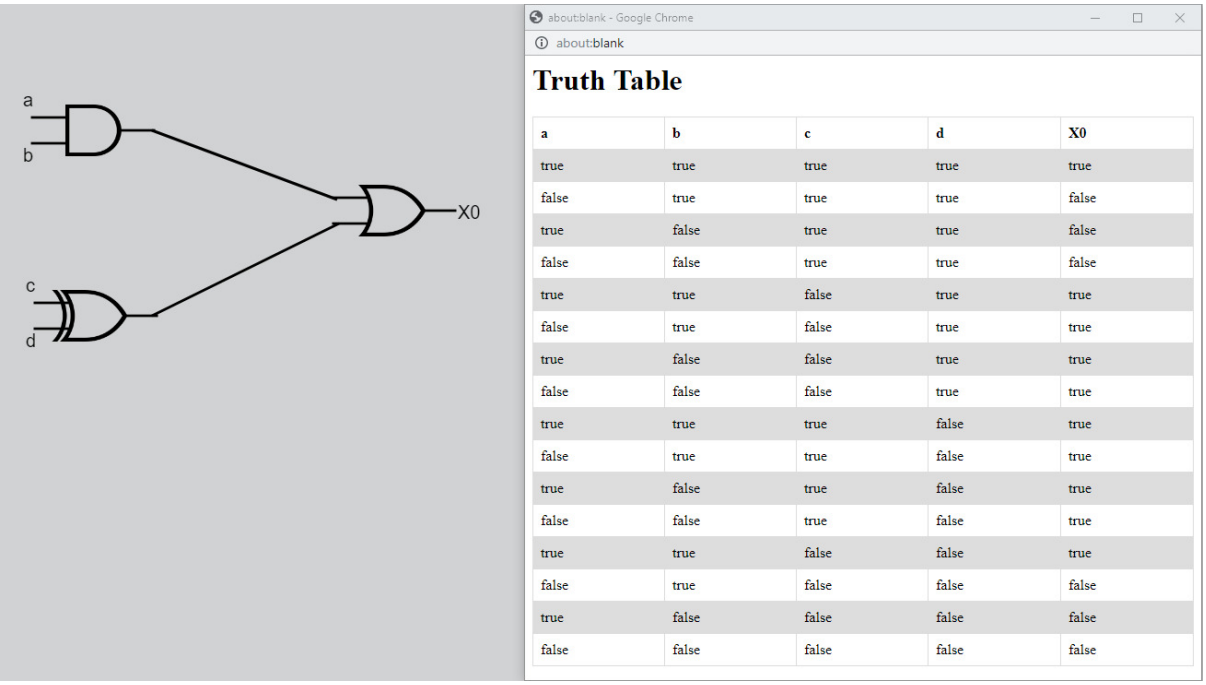


Figure 15 - Image of 'XOR' gate with visual representation of two bounding boxes labelled 'Input Box' (Left) and 'Output Box' (Right)

With the dictionary populated with "null" values, all the gates within the original gates dictionary are iterated through, where each gate is compared with all the lines in the lines dictionary. Again the isPointInPath function is used where the images of the gates are replicated within the canvas to detect which lines connect to which gates but this time with a small difference. Detecting whether lines and gates connected was already implemented within the system, so this should have proved no problem, but when processing the circuit for a truth table, the system needed to know which lines act as input or output for gates. To overcome this problem two bounding boxes were required for each gate instead of one, by creating two bounding boxes for each gate, one could serve as detection for whether a line was connected to the input by only covering the left half of a gate and another to detect lines connected to the output by covering the right half of the gate. Each entry within the dictionary was then populated with the lines that connected to each gate, where any empty inputs or outputs would remain as "null" for use in the rest of the function.

```
// Gates that have no input
} else if(layout[gateKey][0] === null && layout[gateKey][1] === null){
    var inputLabel1 = String.fromCharCode(97 + labelCount);
    draw.fillText(inputLabel1, imageBound.left-170, imageBound.top);
    tableWindow.document.write('<th>'+inputLabel1+'</th>');
    labelCount += 1;
    var inputLabel2 = String.fromCharCode(97 + labelCount);
    draw.fillText(inputLabel2, imageBound.left-170, imageBound.top+60);
    tableWindow.document.write('<th>'+inputLabel2+'</th>');
    labelCount += 1;
```

Figure 16 - Code to demonstrate how gates with no input are detected and a label assigned then placed within the workspace



a	b	c	d	X0
true	true	true	true	true
false	true	true	true	false
true	false	true	true	false
false	false	true	true	false
true	true	false	true	true
false	true	false	true	true
true	false	false	true	true
false	false	false	true	true
true	true	true	false	true
false	true	true	false	true
true	false	true	false	true
false	false	true	false	true
true	true	false	false	true
false	true	false	false	false
true	false	false	false	false
false	false	false	false	false

Figure 17 - Image of labelled circuit (Left) along with corresponding truth table (Right) generated from the generateTable function

Displaying a truth table for the user's circuit would not be enough, since a truth table contains multiple combinations of true and false values being input along with the output from these values, each of the inputs and the output of the circuit would need to be labelled clearly. When creating the initial plan and the requirements within this report, it was decided that the circuit within the workspace would be updated with labels when the system begins the process of producing a truth table, where these corresponding labels would be used within the truth table in a separate window (see figure 17). Firstly, all the inputs within the logic circuit would need to be found, and since we had created a dictionary containing all the gates along with their input and output, this would allow the process to run smoother. Since the dictionary was filled with "null" values before being updated with each gate input and output, any gates missing one of two inputs, or both, or had no output would have a "null" value still in place, meaning a label would need to be placed corresponding to the "null" value of that gate (see figure 16). Input labels iterated through the alphabet (i.e. a for the first input, b for the second, etc) and were placed, these labels were then written to the header of a table that had been generated in a separate window ready to be populated. The same was also done for the output of the circuit by locating the gate with a "null" value where its output element should be.

While processing each gate to detect its input & output then labelling if necessary, each gate was stored in a final array, with the structure [input, input2, gateID, output], input and input2 would contain the gate id for gates who's output was the input of this gate, if either input or input2, or both, did not link to a gate they would contain the input label that had been generated previously. gateID was the unique id of this gate and the output element would contain the gate id of a gate whose input was this gate's output, or output label if it did not link to another gate since this meant it was the output of this circuit. With this final array the truth table could be populated, for each combination of true and false that would be input (i.e. true, true, true then true, true, false etc.), these Boolean values would replace the labels we had originally stored

within this array, to find these input labels within the array was very simple, they were the only elements within the array that could be of length 1. Once this had been complete, we could iterate through the entire array using a while loop that only broke once the gate containing the circuit's output had been found and all the remaining gates had been processed, upon processing each gate they would be removed from the array. Since the gates available for use contain several different operators (such as nand, xor etc.), these would need to be identified when calculating the output of a gate based on its input, which was possible through the gateID. gateID's structure is very similar between all gates, where the type of logical operator is followed with the word "Drag" and then some unique number to identify, i.e. andDrag2. By removing the word "Drag" along with the unique number we are left with the logical operator which we can use to decide how the output will be calculated. Once the circuit's output had been found and no other gates remained within the array, the final output of this gate could be placed in the final column of the table we had generated and this row complete, this would be done for all combinations until finally the table is closed upon completion.

5.2.5. The "NOT" Gate

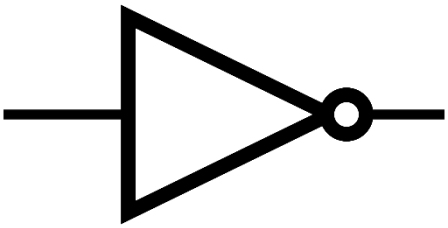


Figure 18 - Image of "NOT" gate which contains only one input and output

While most of the gates within the system allowed for at most two inputs, the "NOT" gate only handled one input, as a result of this many problems were caused by this gate being incompatible with a large portion of the algorithm. This was a problem I had failed to account for while designing the algorithm but required solving during development as it was essential to the functioning of the algorithm.

To overcome this problem I began by creating functions the same as they had intended to be during their design, once the functions were complete, handling of the "NOT" gate would be added. For most of the algorithm, handling the "NOT" gate wasn't a problem and required one or two lines of code, but for the function of generating a truth table, discussed during the previous Section 5.2.4, this required some changes to the algorithm. Luckily, since each entry for each gate within the final array would contain the value "null" before being populated, the algorithm was able to catch the "NOT" gates using their unique id that contained "not" and ensure that only the first input element would be updated for the "NOT" gate, this would also ensure that the second input, named input2 (See section 5.2.4), within the array would remain "null" throughout the function, allowing for the gate to be processed with the same structure as other gates in the system. This also meant that labelling the input of the gate within the workspace also required handling separately to other gates, and although this was handled

through adding additional code, since we only required the knowledge of whether the gate received an input from another gate or not, rather than the number of inputs, this could be solved far faster than expected.

5.2.6. Deleting Objects Within the Workspace

There are many problems that can arise for a user when given the freedom to design and create their own logic circuit. Most of these errors will occur within the workspace, these could be a result of the user placing a gate in the wrong position, placing the wrong type of gate, connecting incorrect gates or simply placing a gate within the workspace that is not required. While the option of clearing the entire workspace is available to the user, this would only create more problems for a user attempting to delete one small individual element from a large circuit.

Throughout the design of the layout and functioning of the system, I had ensured that increasing space for the workspace was the priority while also ensuring that any importance to the functioning or understanding of the system was not lost; demonstrated with icons being displayed on buttons rather than text to reduce their size, where upon hovering them a label appears. While it was possible to place a button that upon selecting the user could then select a gate or line to remove, or offer a bin that users could drag the gates into to erase, none of these felt natural with the theme of the interface and would require additional icons to be placed within the interface.

The final solution that remained was the ability to allow an event trigger upon double click of an element. Since a single click was used to drag gates, connect gates and interact with the buttons, it only felt natural to follow these actions by allowing the user to double click a gate to remove it from the workspace. This was an event trigger that was not being utilised by the system and with jQuery this process was simple. Double clicking a gate within the workspace would retrieve its id, this gate would then be removed from the placed gates dictionary using its id and finally the image clone deleted. But this meant the circuit would be left with unnecessary lines, causing problems within the rest of the system since many lines within the circuit would lead to or originate from nothing. A canvas within HTML is transparent, and it does not store the individual lines drawn within it, nor is it possible to select and erase these individual lines, causing many problems for this type of system.

```
// Delete original canvas and replace
var cnvs = $('#lineCanvas').position();
draw.clearRect(cnvs.left, cnvs.top, 900, 1610);
$(e.target).remove();
// Redraw lines
for(key in allLines){
    draw.beginPath();
    draw.moveTo(allLines[key][0], allLines[key][1]);
    draw.lineTo(allLines[key][2], allLines[key][3]);
    draw.lineWidth = 3;
    draw.stroke();
}
```

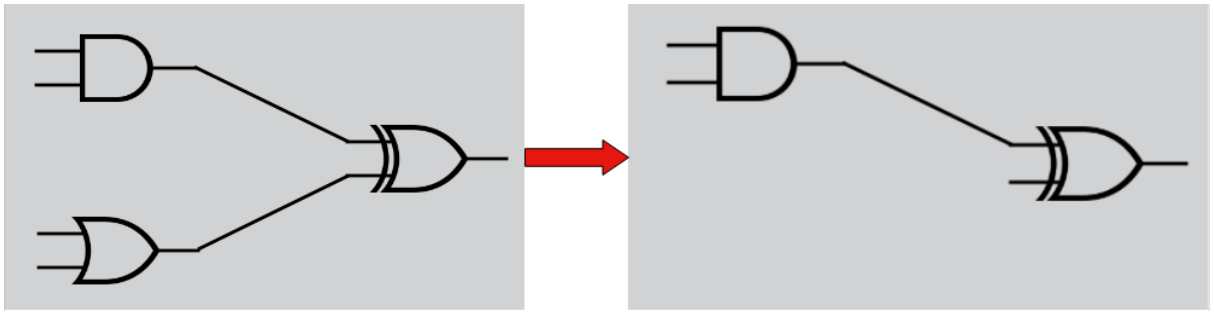


Figure 19 - Sample of code used to delete gates and lines from workspace (Top) and response of interface when "OR" gate is deleted from this circuit (Bottom)

Since lines drawn within the workspace cannot be done unless they are connecting two existing gates placed within the workspace (see Section 5.2.3), a user would not require the tool to delete individual lines, only the ability for lines connected to a certain gate to be deleted when that specific gate is deleted. To do this the original delete gate function was edited, originally it only removed the specific gate that was double clicked from the workspace and the gates placed dictionary, but by again creating a bounding rectangle around the gate and replicating within the canvas, as discussed in Section 5.2.3, we could then iterate through all the lines contained within the lines dictionary and check if their endpoints are contained within the bounding box, if so, those lines are deleted from the dictionary. Despite deleting from the dictionary, the user interface would still display a line, since, as discussed above, it is not possible to remove specific lines from a canvas. Figure 19 demonstrates how the code will delete the canvas that contained the original lines, where it will then create a new canvas in its place, essentially clearing all the lines from the canvas, and then populate the canvas with all the lines contained within the updated lines dictionary.

6. Results and Evaluation

The agile-scrum methodology that was used throughout the development of this project ensured that testing of a function or task was carried out before it could be considered complete and ready to implement into the final system. The tasks carried out during each sprint were tracked using Trello Boards (see Section 2), this application proved very useful for the development of this project with a new board being created each sprint and filled with new tasks. This offered a clear distinction between the features I was attempting to implement during each sprint while including a backlog section which allowed for uncomplete tasks to flow between sprints and ensure they were not forgotten in a previous sprint. Overall I believe this application proved crucial for supporting the methodology I had implemented, by offering the ability to easily track and test individual components before implementing into the system, a strong foundation for the entire system was in place and allowed for reliable testing of the entire system upon completion.

I believe the work plan I had implemented at the beginning of this project was also very effective, by splitting each sprint into a two week period it allowed for time to be managed well, it meant that more time could be allocated to larger tasks within that sprint period and offered a milestone to be accomplished by the end of that two week period. Throughout the development of the project, feedback on different aspects were very important and by organising a compulsory meeting with my supervisor at the end of each two week period, I could receive outside feedback on the goals I had achieved that week and ensure they met the expectations of the initial brief before moving forward. Additionally, planning optional meetings each week during the two-week sprint period that would only go ahead if I faced a problem were also very effective. They meant that if I encountered a problem with the project that I could not solve alone, this meeting would be organised and the problem solved, meaning less time was wasted stuck on a single task, but if a problem did not arise, these meetings would not take place and the flow of work would not be interrupted by a meeting that was not necessarily required.

This section aims to provide a summary of the testing that was performed on various components within the project, if and how the goals originally placed had been achieved, and whether the system implemented works as intended.

6.1. Functional & Non-Functional User Requirements Testing

Contained within the below tables are the functional and non-functional user requirements that had originally been set at the beginning of the project. The tables also contain a short description of how these requirements were implemented into the system along with a final value of whether they pass or fail; pass meaning the requirement works as intended, fails otherwise.

The purpose of these tables is to demonstrate which user requirements were implemented or not. Some functional and non-functional requirements within the tables required further testing, these will be covered later in this section of the report.

6.1.1. Functional Requirements

During development of the project, all functional requirements were implemented.

Requirement	Outcome	PASS/FAIL
The solution must contain a list of pre-defined logic gates that include at least NAND, NOR, NOT, AND, OR and XOR.	Interface contains all these logic gates within a list.	PASS
The user must be able to drag and drop gates from the list into a designated workspace.	All gates within the list can be dragged and dropped into the designated workspace, gates cannot be dragged or dropped outside of the designated workspace.	PASS
The user must be able to connect these gates within the designated workspace in order to create a logic circuit.	Lines can be drawn between gates to connect. Lines cannot be drawn if they are not connecting two individual gates.	PASS
By clicking a button, the user must be able to store/download the circuit onto their device.	Upon selecting the “Save Circuit” button, a dynamic download link will be created, and an artificial mouse click will trigger the circuit to be downloaded onto the user’s device.	PASS
The user must be able to load a pre-made circuit into the workspace by selecting it on their device.	Upon selecting the “Load Circuit” button, a user will be prompted within a separate window to select the file they wish to use. The gates and lines contained within this circuit will then be loaded into the workspace.	PASS
By clicking a button, the user should be able to generate a truth table from the circuit created or loaded into the workspace.	Any circuits created or loaded into the workspace can generate a truth table if they follow the rules of a traditional logic circuit.	PASS
System must generate a truth table within a reasonable amount of time, regardless of the size.	All truth tables are generated within a reasonable amount of time, regardless	PASS

	of the size. There is no noticeable delay between most circuit sizes.	
User should be able to edit a circuit that is loaded into the workspace from a saved file.	Circuits loaded into the workspace can be edited. User has all the abilities they would with a circuit they had just created within the workspace.	PASS

Table 5 - Table displaying which functional requirements were implemented or not

6.1.2. Non-Functional Requirements

Although most of the non-functional requirements were implemented into the system, two were not possible due to a lack of time and a low priority.

Requirement	Outcome	PASS/FAIL
Workspace should be able to handle lots of components/gates.	If circuit follows traditional logic circuit rules and is contained/fits into the workspace, system can handle regardless of the number of gates used.	PASS
Logic gate symbols should be clear, easily accessible and labelled clearly.	All logic gates have a fixed height of '70px' and a width of '150px' making them clearly visible and distinguishable. Clear label of the gate is also placed below.	PASS
Logic gate list should be scrollable/dropdown if all gates within the list do not fit within the window.	Due to a lack of time and a low priority for this requirement, lists are not scrollable if they do not fit within the window, although user can still scroll the web page to access.	FAIL
When dragging the gate from the list into the workspace, a clone should appear in the mouse position, allowing the user to understand where the gate is being placed.	A clone of the gate the user is dragging will appear in the location of the mouse while it is being dragged.	PASS
On generation of the truth table, the inputs and outputs of the circuit should be clearly labelled and match the labels within the truth table.	On generation of a truth table, the corresponding circuit within the workspace will be labelled. The inputs will be labelled in alphabetical order and the output labelled; these labels will	PASS

	then be used in the head of the truth table.	
User should be able to delete a gate within the workspace, removing any connecting lines to the input and output of that specific gate.	Upon deletion of a specific gate, any input and output lines connected to that gate will be deleted while any other gates or lines in the workspace will remain unaffected. (See section 5.2.6).	PASS
User should be able to select a button that clears the entire workspace.	"Clear Workspace" button will clear any lines or gates within the workspace.	PASS
Truth table could be generated within a separate window, allowing user to compare the truth table with another or edit the circuit without losing the generated table.	Truth table generated from a circuit is displayed within another smaller window that can be moved and resized by the user.	PASS
Circuits should be stored in a file format that is readable for users, such as CSV.	Circuits are stored within a CSV file that is human readable. (See section 5.2.1)	PASS
Users could have the option to save the truth table generated by their circuit.	Due to a lack of time, allowing a user to store their generated truth table was not possible. Although the window where the truth table was generated can be stored as a regular webpage file.	FAIL

Table 6 - Table displaying which non-functional requirements were implemented or not

6.2. Unit Testing

With the generation of a truth table being critical to the success of this project, allowing a user to generate a truth table within a reasonable amount of time was very important, and not only did it need to generate within a reasonable amount of time but the time it took to generate a table for a large circuit compared to a small circuit needed to be almost unnoticeable.

Unit testing allows us to test individual units/components of the software [17], making it the ideal type of test for this purpose. Unit testing offers the benefits of allowing confidence in our code since, through rigorous testing, we can be sure of the time we should expect to generate a truth table depending on its size. It also allows verification that the algorithm I originally designed is beneficial for this type of requirement.

To test the time it would take to generate a truth table for different size circuits, the "console.time" and "console.timeEnd" functions will be used from the Console API within Chrome [5], where the recorded time will be displayed within the console. To ensure a fair and consistent test, the following steps will be taken:

- 4 different sized circuits will be used for testing. Since size means the number of gates within the circuit, these 4 circuits will contain 1 gate, 3 gates, 7 gates and 15 gates.
- Each circuit size will be tested 10 times where the average will then be taken.
- All times will be recorded in milliseconds.
- 2 graphs will be plotted, 1 graph will be circuits containing only 1 type of gate while the other will be a variation of different gates.
- Each variation circuit will contain the same variation of gates, excluding the circuit with only 1 gate. Since the smallest circuit excluding the first is 3, the gate variation at most can be 3, AND, XOR and NOR will be used as this combination offers the widest variety.
- All gates used within each circuit will remain the same for all 10 tests.

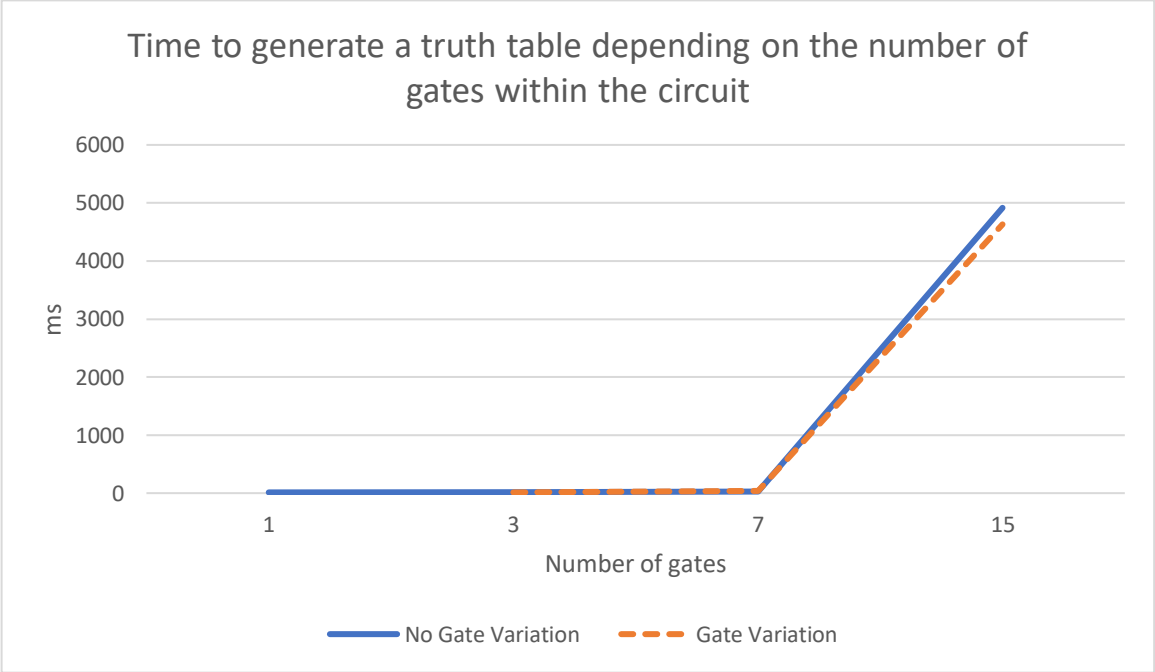


Figure 20 - Graph showing the relationship between the number of gates in a circuit and the time it takes to generate a truth table

In the above graph containing the results of the unit test for the time it would take to generate a truth table depending on the number of gates within the circuit, we can see that from 1 to 7 we see no real change between the times, although there are very small changes, they would be completely unnoticeable to the user and are unnoticeable within this graph. As we move from 7 gates to 15, we see a steady rise in the time it would take to produce a truth table, meaning the time to produce a table would grow exponentially as the number of gates are increased from around 7.

After conducting this test and plotting within a graph it has allowed me to confidently answer the question of whether the system is able to generate a truth table within a reasonable amount

of time. I believe the system and the algorithm implemented are able to produce a truth table within reasonable time, with the intended audience of this project I would expect a circuit containing around 3-7 gates to be most commonly used, and by comparing this with the graph, a truth table corresponding to a circuit of that size would be produced almost instantly. As stated at the beginning of this section, when creating the system I wanted to ensure that the time taken to produce a truth table between a large circuit and a small circuit would be unnoticeable, while 7 gates is a large circuit and the time taken is unnoticeable compared to a circuit of size 3, when reaching around 15 gates, the time becomes noticeable. Despite this, while the graph may seem like the time taken would be a considerable amount of time, since it is measured in milliseconds the time taken would only be around 5 seconds, and for a circuit of this size that is uncommon for a user to create, this is a very reasonable amount of time to wait.

6.3. Test Cases

Below are the test cases that were used during the end of each sprint to ensure the project had accomplished the main aim set during that sprint. Each test case will test the main aim of that sprint, along with any additional features that were added during that sprint. It will also test to ensure as much error prevention as possible has been implemented.

<table>
<tr><td colspan="2">Test Case ID: Sprint 1</td><td colspan="2">Test Purpose: Software contains pre-defined logic gates that include at least NAND, NOR, NOT, AND, OR and XOR.</td></tr>
<tr><td colspan="4">Environment: Google Chrome, Windows 10</td></tr>
<tr><td colspan="4">Preconditions:</td></tr>
<tr><td colspan="4">Test Case Steps</td></tr>
<tr><td>Step No</td><td>Procedure</td><td>Response</td><td>PASS/FAIL</td></tr>
<tr><td>1.</td><td>On webpage load</td><td>List is displayed within the interface containing NAND, NOR, NOT, AND, OR and XOR gates.</td><td>PASS</td></tr>
<tr><td>2.</td><td>On webpage load</td><td>Designated workspace is displayed, and boundary is clear.</td><td>PASS</td></tr>
<tr><td>3.</td><td>On webpage load</td><td>Help window/tooltip is displayed at top right corner of the workspace in which user can hover mouse over to dismiss.</td><td>PASS</td></tr>
<tr><td colspan="4">Comments:</td></tr>
</table>

Table 7 - Test case results for sprint 1

Test Case ID: Sprint 2	**Test Purpose:** User interface allows for the logic gates to be dragged, dropped and connected inside a designated workspace within the interface.		
Environment: Google Chrome, Windows 10			
Preconditions: Webpage is loaded, Workspace is clear			
Test Case Steps			
Step No	**Procedure**	**Response**	**PASS/FAIL**
1.	Any gate within the list can be dragged into the workspace.	Upon selecting a gate and dragging, a clone of the correct gate is always displayed at the mouse position. Gate cannot be dragged outside of the designated workspace.	PASS
2.	Gate is dropped into the workspace.	The gate that was dragged by the user is placed into the workspace at the same position it was dropped. Clone of gate is no longer displayed at mouse position.	PASS
3.	Another gate within the list is dragged by the user and dropped into the workspace.	Clone of gate is displayed at mouse position again and gate is placed at position it is dropped. Both gates are visible within the workspace and remain at the position they were dropped.	PASS
4.	Output of first gate is clicked, then input of second gate is clicked.	Line is drawn between the output of the first gate and input of the second, matching the click position of the user on both gates.	PASS
5.	Two random positions within the workspace are clicked, neither include gates.	No line is drawn between the two positions clicked; actions of user are ignored since they do not connect two gates.	PASS
6.	Gate in the workspace is double clicked.	Gate is deleted from the workspace while any lines connecting to the input or output of that gate are also deleted.	PASS

7.	"Clear Workspace" button is clicked.	All gates and lines within the workspace are deleted, clearing the workspace.	PASS
Comments:			

Table 8 - Test case results for sprint 2

Test Case ID: Sprint 3	**Test Purpose:** Constructed circuits can be stored and easily accessed/loaded into the workspace.		
Environment: Google Chrome, Windows 10			
Preconditions: User has constructed a circuit within the workspace.			
Test Case Steps			
Step No	**Procedure**	**Response**	**PASS/FAIL**
1.	"Save Circuit" button is clicked.	Circuit is downloaded within a file named "circuit" onto the user's device.	PASS
2.	"Load Circuit" button is clicked.	Window is displayed where user can select a specific file to be loaded into the workspace, once selected, the circuit is displayed within the workspace.	PASS
3.	Random file is selected that is not a saved circuit.	System will display an error message to the user indicating the problems with the file selected. Could be too large, wrong format or contain an incompatible circuit.	FAIL
4.	Circuit is loaded into the workspace.	Circuit that is loaded into the workspace can now be edited, saved again and used to generate a truth table.	PASS
Comments: Due to time constraints an effective error prevention for loading incompatible files into the workspace was not implemented. Users are given a window to select the correct file so attempting to load an incorrect file would be the result of user error. Circuits edited and then saved again will be saved as a new file to avoid overwriting a previous circuit.			

Table 9 - Test case results for sprint 3

Test Case ID: Sprint 4	**Test Purpose:** Truth table can be generated from a constructed circuit.		
Environment: Google Chrome, Windows 10			
Preconditions: User has constructed or loaded a circuit within the workspace.			
Test Case Steps			
Step No	**Procedure**	**Response**	**PASS/FAIL**
1.	"Generate Truth Table" button is clicked.	Inputs and outputs of logic circuit are labelled, a new window is also created which displays a truth table containing these labels.	PASS
2.	Button is selected to save/export the truth table.	Truth table is saved into a file which can be accessed to view truth table at any time.	FAIL
Comments: With time constraints and low priority, a function was not implemented for saving or exporting the truth table since there were many possibilities to store these already, such as saving the webpage or taking a screenshot.			

Table 10 - Test case results for sprint 4

6.4. Browser Testing

During Section 3.2 I had identified that the system would be created primarily in Google Chrome since this was the most used web browser worldwide and the devices I developed this system on primarily used Chrome. In order to ensure this system would be available to as many users as possible, the remaining browsers would also have to be tested and the system adjusted to ensure these browsers would also be compatible.

I began by identifying the browsers where I then created a checklist, this would ensure the browsers were fully tested for all features and not simply the appearance of the interface within that webpage. Below is the outcome of this test:

Figure 21 - Image of system interface within most commonly used browsers

Operation	Google Chrome	Internet Explorer	Microsoft Edge	Firefox	Opera
All components within the interface fit within the window.	✔	✔	✔	✔	✕
All gates are loaded correctly with corresponding labels.	✔	✔	✔	✔	✔
Correct icons are loaded for each button.	✔	✔	✔	✔	✔
Correct label appears when hovering the buttons.	✔	✕	✔	✔	✔
Gates can be dragged and dropped.	✔	✕	✔	✔	✔
Gates are contained within the workspace.	✔	✕	✔	✔	✔
Lines can be drawn between gates in the workspace.	✔	✕	✔	✔	✔
Individual gates can be deleted from workspace.	✔	✕	✔	✔	✔
Upon deleting a specific gate, connecting lines are also deleted.	✔	✕	✔	✔	✔
Entire workspace can be cleared using the “Clear Workspace” button.	✔	✕	✔	✔	✔
Circuit is downloaded when “Save Circuit” button is clicked.	✔	✕	✔	✔	✔
Circuit can be loaded into the workspace when “Load Circuit” button is clicked.	✔	✕	✔	✔	✔
Truth table is generated in separate window when “Generate Truth Table” button is clicked. Circuit in workspace is also labelled and corresponding truth table contains correct labels.	✔	✕	✔	✔	✔
Tooltip displays correct information and can be dismissed when hovered.	✔	✕	✔	✔	✔

Table 11 - Table displaying the compatibility between the system and commonly used browsers

After reviewing the browser compatibility test it is clear that the system developed is very compatible and remains responsive on almost all of the browsers. During testing I encountered no problems with Google Chrome, Microsoft Edge and Firefox, while Opera proved to respond perfectly fine to user input but encountered the problem of being unable to fit all the components within the window. This is due to the large bookmark bar that is default at the top of the page, causing all elements to be pushed down and making the bottom bar of the workspace boundary hidden. This bookmark bar can be disabled or reduced but when testing I decided it would be best if the browser settings remained default as this is how many users' browsers would look.

Finally, I was able to load the webpage within Internet Explorer with no problems, all gates and features were displayed within the webpage, but the webpage remained completely static and I was unable to interact with any of the components within the webpage. This could be caused by incompatible syntax used within a specific part of my source code, thereby causing a significant error within the system.

Since there were no clear ways to solve the errors that were making the system incompatible with Internet Explorer, I decided it would be best if the code were not changed to fix this since the scope of the problem was unknown. In addition, four of the five browsers were almost completely compatible with the code and were responsive, but by attempting to change the code in order to make the system compatible with Internet Explorer it would run the risk of reducing the systems compatibility with the remaining browsers. The priority to fix this browser was not worth the risk of losing compatibility with other browsers since Microsoft Edge was introduced into the Windows systems as a replacement for Internet Explorer, meaning that for most users who use Internet Explorer, it is likely that their system also has Microsoft Edge installed.

7. Future Work

This section of the report will outline any additional work I would have implemented into the project given more time, additionally it will offer a perspective on how the project could be expanded by me or someone else looking to continue the work.

7.1. Improving the Implementation

Below are the improvements I would have made to the implementation given more time.

7.1.1. Adjusting Gates

Gates within the system are already draggable, they can be dragged into a workspace and placed within any position of the workspace, but once a gate is dropped the user is unable to drag this gate again. There are many reasons why this ability was not implemented, the main reason being a lack of time and the amount of resources this feature would require. I had originally implemented the ability to drag gates during sprint 2, but had failed to include this ability when it was first implemented, and I was not aware that this would be a feature users would want until testing began at the end of development, where the remaining time had to be used to construct this report. Nevertheless I proceeded to find a solution to understand the scale of this feature and whether or not time could be allocated to include it.

Allowing users to drag a gate again once it has dropped requires a few lines of code within the gate draggable function discussed in section 5.1, I had found a solution to a similar problem on StackOverflow that quickly solved this problem [11]. But implementing this ability would require further error prevention and restructuring of other functions, since allowing a user to drag a gate once dropped would mean any lines connected would also need to be readjusted.

Although I had decided not to implement this feature since it would create further problems that required solving, anyone looking to implement could begin by making the gate draggable once dropped as discussed above, where on dropping the gate again the gate dictionary would be updated to contain this specific gates new position. When the gate is dragged again a bounding box will need to be created around the gates original position to detect any inputs or outputs, if it contained inputs or outputs these lines would need to be updated within the dictionary to correspond with the gates new position, once the dictionary has been updated the canvas would need to be cleared (see section 5.2.6) and a new canvas created then populated with the updated lines dictionary.

7.1.2. Multiple Inputs

While the gates within the system currently can receive up to two inputs, excluding the “NOT” gate, allowing users to add more inputs than two to a gate would provide users with the ability

to create a wider range of circuits. The algorithm currently in place would be simple to improve for this requirement, since many of the dictionary's and arrays used within the algorithm reserve input and output spaces with the value "null", adding the ability for a user to choose how many inputs a gate has would simply require the system to dynamically create an array containing a certain amount of elements depending on the amount of input selected.

Representing this within the interface to a user would prove more difficult. Since the gates have a fixed image containing two input connectors, excluding "NOT" gate, depending on the amount of inputs a user selects for a particular gate, these input connectors within that image would need to be updated to include the correct amount. This could be done by storing multiple images of each gate with a different input connector amount and retrieve the corresponding image on user input.

7.1.3. Multiple Outputs

At this point within the development of the project it is currently not possible to produce a logic circuit with multiple outputs. This is caused by how the algorithm labels the circuit and then uses these labels within the truth table, attempting a circuit with more than one output will cause an infinite loop within the system. Although error prevention has been put in place to avoid users creating a circuit with multiple outputs, moving forward with this project one of the first problems to address would be this one. By allowing users to create a circuit that contains multiple outputs, it opens a wide range of further circuits a user could create with this system.

7.1.4. Connecting Lines and Gates

When connecting lines and gates within this project, it was always important that it felt natural and easy, where a user did not have to struggle to select a small point on each gate to create a line. As a result I created "Lazy Lines" where a user could select any point on the gate to create a line, and since the algorithm only detected if this point was contained within the gate, the line would be drawn at that point, regardless of how it appeared to a user. While the system would understand where this point connected, since a user could place the line close to an output or input of a gate but did not specifically have to be on the gate, it could appear very confusing for a user and cause the circuit to look unstructured. This was not possible to improve within the allocated time for this project and would require a redesign of the line connecting algorithm.

A redesign of this feature would mostly change how the interface displays the connecting lines and gates, since the system would already understand how these gates connect. One solution would be to ensure the lines drawn between two gates moved to automatically connect to the gates input or outputs depending on the lines relative distance to them. This would remove space that can occur between lines and gates if the lines are not placed in the correct position by a user.

7.1.5. Loading Circuits

While the loading of circuits works as was intended when beginning this project, through feedback received it become apparent that the way this worked could be adjusted to allow the possibilities of circuits that could be created to increase. When a user loads a circuit into the workspace it is currently placed with the identical positions to the original circuit that had been saved by the user, while this is very useful, offering additional functionality to this would allow the system to benefit greatly.

To achieve this, when a user loads a circuit into the workspace, this circuit could be represented as a single object such as a rectangle, where this rectangle contains the same amount of inputs and outputs as the circuit that was loaded from the file had, and gives the same output as the circuit would depending on the input. From this a user can create a far larger and complex circuit than would originally be possible since the circuit they had loaded would be represented as a single object, reducing the area within the workspace it would occupy.

7.2. Expanding the Project

Below are the details on how the project could be expanded within the future.

7.2.1. Logic Circuit Workshop

From reading the brief at the beginning of this project, a desire to implement a cloud-based circuit storage system had persisted throughout the entire project.

This system would work by allowing users to first create an online account and login, from here they would have the ability when loading or saving a circuit to access an external system that would allow them to store or retrieve circuits from here. This would not only offer the benefit of users storing their circuits online, allowing them to be retrieved from anywhere at any time, but by creating the option of allowing a user to make their circuit public or not, a public circuit would mean other users in the system can access and retrieve this circuit to be loaded into their workspace, where they could generate a truth table or extend the circuit. Offering the ability to share circuits between users without the need to transfer files would allow this system to be used in a wider range of circumstances than the current one can, one example would be the ability to use this system as a teaching tool in Universities or School where lecturers could share circuits with their student or access circuits students had created.

Unfortunately, a feature implemented into the project of this scale would require more time, resources and additional members of the team perhaps; it would be a separate project in itself. Despite this the workspace that had been created for this project would be very compatible with this system, since only the saving and loading of circuits would need to communicate/operate with this system, meaning a massive redesign of the entire system would not be required, only these functions.

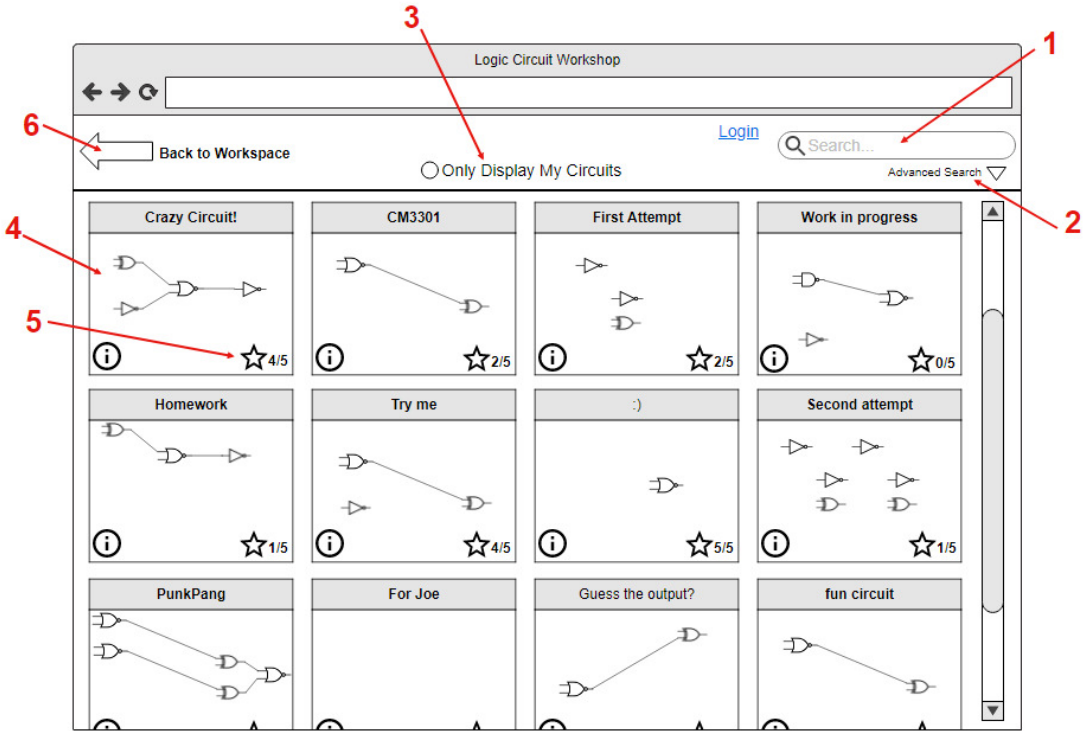


Figure 22 - Labelled interface mock-up for "Logic Circuit Workshop", created using moqups.com

Features:

1. Search bar for username of circuit creators and name of circuit.
2. Advanced search ability to filter certain gates only, star rating, specific users etc.
3. Ability to only display user's circuits.
4. Small preview of circuits.
5. Star rating and option for further information i.e. creator's username, date created.
6. Navigation back to workspace.

7.2.2. Additional Components

While the project answered the brief of creating a logic circuit builder that included NAND, NOR, NOT, AND, OR and XOR gates, adding additional gates and components into the list would also be possible. Due to time constraints only these six gates could be added, but with this algorithm additional gates could easily be implemented in future such as a XNOR gate. In

addition to adding more gates the project could be expanded to include different types of components such as:

Light Bulb which only lights up if its input is true.

Input controls such as a toggle switch or a push button to change the input of the circuit between true or false. This would also work very well with a lightbulb giving a real time response to the user.

4-bit digit which displays a hexadecimal digit that is constructed from four inputs.

8. Conclusion

After writing this report and developing the project over the past few months, I believe the main aim of developing a software environment for building computer logic circuits from simpler circuits was achieved. The research conducted at the beginning of this project that looked into many different aspects of the project such as design, layout, algorithm and similar products, allowed for key problems to be identified and a base for a good solution to be developed.

The work plan used of breaking down the main objective into smaller objectives, followed by spreading these out across separate sprints where smaller tasks could be created to achieve these sprint goals proved very effective. It also allowed for the project to achieve many of the "Ideal Product" goals I had set during the initial plan (see section 2), these could be done during the period between completing the main aim of the sprint and the end of that particular sprint.

I believe the system created provided a good solution to the problems identified and answering the brief. The implementation of the system proved very successful, while there is always room for improvement, the system achieved features that many of the pre-existing products could not, while ensuring many of the key features of these products were also implemented. All of the main goals set during the beginning of this project were fully achieved, and testing conducted within this report can fully support this. Moving forward with this project I believe this report has provided the reader or someone who would be looking to continue this work with a clear understanding of how the system operates along with my perspective on how this system could be improved and expanded. Throughout the report I believe I covered all of the project, from how the system would initially be developed, up until the implementation of the system and a thorough evaluation of this system. I ensured that this report gave a clear and concise explanation of the project along with how the system I had created would operate, I ensured that various sections within this report were always referenced to as it allowed the reader to not only navigate the report far easier but also ensure they had a good understanding of a particular component before diving deeper into it.

Overall I have found this project to be a very good learning experience, it has offered the opportunities to develop a project that I am invested in, research into domains that I may never have encountered and allowed me to utilise skills learnt throughout my studies.

9. Reflection on Learning

Working on this project has proved challenging but also very rewarding, I have grown and learnt more within the past few months than I ever thought possible. When beginning the project I was very unsure if it would be possible to create a good solution for the problem that had been identified and how I would manage this alone within the limited time frame we were given.

Although my supervisor, Professor David Walker, was very supportive during this process and offered any guidance I required, this project involved a tremendous amount of problem solving. I believe throughout this project my problem solving skills have greatly improved, not just in solving technical problems with the software but also within project management, since many new problems which I could have not been prepared for during my studies would arise. While I had worked on projects in the past, these had always been in small groups where the workload was split evenly, meaning many aspects of these projects I did not encounter as I was not responsible for. By working on this project alone I think it allowed me to improve particular aspects of my project management skills, since dealing with elements such as testing of the project, which I had not been responsible for in previous projects I worked on, would need to be completed by me.

Finally I am thankful for the technical skills this project has allowed me to improve. Beginning the project I was confident working in HTML and JavaScript, but throughout development of this project I was able to include external libraries that I had never worked with before such as jQuery. jQuery proved to be very useful for this type of project, while I had to learn how to operate this library while developing the project, I believe I utilised it very well within the system and I would be quick to use it again within future projects involving JavaScript.

References

[1] Academo.org, "Academo," Academo.org, 2016. [Online]. Available: https://academo.org/demos/logic-gate-simulator/. [Accessed May 2019].

[2] Atlassian, "Trello," 2019. [Online]. Available: https://trello.com/en-GB. [Accessed May 2019].

[3] Awio Web Services LLC, "Browser & Platform Market Share," April 2019. [Online]. Available: https://www.w3counter.com/globalstats.php?year=2019&month=4. [Accessed May 2019].

[4] balsamiq, "Balsamiq Wireframes," Balsamiq Studios, LCC, May 2019. [Online]. Available: https://balsamiq.com/wireframes/. [Accessed 2019].

[5] K. Basques, "Console API Reference," 1 May 2019. [Online]. Available: https://developers.google.com/web/tools/chrome-devtools/console/api#time. [Accessed May 2019].

[6] B. Born, "Simulator IO," 2015. [Online]. Available: https://simulator.io/. [Accessed May 2019].

[7] Delaware Corporation, "Font Awesome," Fonticons, Inc., 2019. [Online]. Available: https://fontawesome.com/cheatsheet. [Accessed May 2019].

[8] T. Fessenden, "Horizontal Attention Leans Left," 22 October 2017. [Online]. Available: https://www.nngroup.com/articles/horizontal-attention-leans-left/. [Accessed May 2019].

[9] D. Ford, "SceinceDemos.org.uk," April 2019. [Online]. Available: https://sciencedemos.org.uk/logic_gates.php. [Accessed May 2019].

[10] IIIT-Bangalore, "CircuitVerse," International Institute of Information Technology Bangalore, December 2018. [Online]. Available: https://circuitverse.org/. [Accessed May 2019].

[11] A. P. Johny, "jQuery Drag and Drop - Dropped element can not drop again," 20 March 2015. [Online]. Available: https://stackoverflow.com/questions/29159677/jquery-drag-and-drop-dropped-element-can-not-drop-again. [Accessed May 2019].

[12] Mediative, "Keeping an eye on Google – Eye tracking SERPs through the years," 2011. [Online]. Available: http://www.mediative.com/eye-tracking-google-through-the-years/. [Accessed May 2019].

[13] "moqups," 2019. [Online]. Available: https://moqups.com/ . [Accessed May 2019].

[14] Mozilla, "MDN web docs," March 2019. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D/isPointInPath. [Accessed May 2019].

[15] A. Osmani, "How To Write Fast, Memory-Efficient JavaScript," 5 November 2012. [Online]. Available: https://www.smashingmagazine.com/2012/11/writing-fast-memory-efficient-javascript/. [Accessed May 2019].

[16] Python.org, "TkInter," Python Software Foundation, March 2019. [Online]. Available: https://docs.python.org/2/library/tkinter.html. [Accessed May 2019].

[17] Software Testing Fundamentals, "Unit Testing," 2019. [Online]. Available: http://softwaretestingfundamentals.com/unit-testing/. [Accessed May 2019].

[18] Sublime HQ Pty Ltd, "Sublime Text," Sublime HQ Pty Ltd, 2019. [Online]. Available: https://www.sublimetext.com/. [Accessed May 2019].

[19] The jQuery Foundation, "jQuery," The jQuery Foundation, 2019. [Online]. Available: https://jquery.com/. [Accessed May 2019].

[20] tutorialspoint.com, "UML - Activity Diagrams," 2019. [Online]. Available: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm. [Accessed May 2019].