Nottingham Trent University

School of Science and Technology

A New Form of Educational Logic Gate Simulator

by

Elliot Christopher Harding

in

2020

Project report in part fulfilment

of the requirements for the degree of

Bachelor of Science with Honours

in

Software Engineering

I hereby declare that I am the sole author of this report. I authorize the Nottingham Trent University to lend this report to other institutions or individuals for the purpose of scholarly research.

I also authorize the Nottingham Trent University to reproduce this report by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signature

Elliot Christopher Harding

Abstract

This work introduces the use of logic gate simulators in educating students about the functionality of logic gates and how they come together to form circuits. It introduces existing methods on how this is accomplished. The report then goes on to investigate an apparently new design of logic gate simulator intended to educate students on logic gates, to an A-Level standard, more effectively. The new design, with its implementation described within the report, involves a challenge mode designed in a ‘level’ like manner, in which teachers can create truth table to circuit and circuit to truth table conversion-based challenges for students. The challenge-based approach is intended to engage students in the ‘learning through reflection on doing’ process. This approach through means of user testing and observation has been deemed successful in that it provides students an alternative method to learn about logic gates, and evidence in this report shows that it may be more effective than previous approaches.

Acknowledgements

Enter acknowledgements here. It is usual to acknowledge those that have assisted you in your work and will normally include your main project supervisor. The order of acknowledgments (most important first) and their respective length indicates their relative importance to you.

Neil Sculthorpe ~ Tutor, review points, provided information relevant to report, overlooked project planning document ect…

Ferreira, Joao Filipe ~ Helped with formulating idea for new approach to logic gate simulator education

Table of Contents

[Abstract ii](#_Toc39410126)

[Acknowledgements iii](#_Toc39410127)

[Table of Contents iv](#_Toc39410128)

[List of Figures viii](#_Toc39410129)

[List of Tables ix](#_Toc39410130)

[CHAPTER 1 1](#_Toc39410131)

[Introduction 1](#_Toc39410132)

[1.1 Introduction 1](#_Toc39410133)

[1.1.1 Scope 1](#_Toc39410134)

[1.1.2 Relevance 1](#_Toc39410135)

[1.2 Background 1](#_Toc39410136)

[1.2.1 Logic Gate Simulators 2](#_Toc39410137)

[1.2.2 Use of Logic Gate Simulators for Education 3](#_Toc39410138)

[1.3 Issues with Using Logic Gate Simulators for Education 3](#_Toc39410139)

[1.4 Report Layout 4](#_Toc39410140)

[CHAPTER 2 5](#_Toc39410141)

[CONTEXT 5](#_Toc39410142)

[2.1 Introduction 5](#_Toc39410143)

[2.2 Existing Solutions to Logic Gate Education 5](#_Toc39410144)

[2.2.1 Books or Online Research 5](#_Toc39410145)

[2.2.2 Using Models 7](#_Toc39410146)

[2.2.3 Simulator Software 9](#_Toc39410147)

[CHAPTER 3 12](#_Toc39410148)

[New Ideas 12](#_Toc39410149)

[3.1 Introduction 12](#_Toc39410150)

[3.2 Requirements 13](#_Toc39410151)

[3.3 Project Planning 14](#_Toc39410152)

[3.3.1 Methodology Used 14](#_Toc39410153)

[3.3.2 Deliverables 15](#_Toc39410154)

[3.3.3 Milestones 18](#_Toc39410155)

[3.3.4 Workload Balancing 18](#_Toc39410156)

[3.3.5 Tools And Resources 19](#_Toc39410157)

[3.3.6 Contingency Planning 23](#_Toc39410158)

[CHAPTER 4 26](#_Toc39410159)

[IMPLEMENTATION 26](#_Toc39410160)

[4.1 Introduction 26](#_Toc39410161)

[4.2 Base Simulator Design 26](#_Toc39410162)

[4.2.1 Simulator Objects 26](#_Toc39410163)

[4.2.2 Code Core Design 30](#_Toc39410164)

[4.2.3 Features?? 30](#_Toc39410165)

[4.2.4 GUI 30](#_Toc39410166)

[4.3 Task Feature 31](#_Toc39410167)

[4.3.1 Development 31](#_Toc39410168)

[4.3.2 Result 31](#_Toc39410169)

[4.4 Unit Testing 31](#_Toc39410170)

[4.4.1 Benefits of Unit Testing 31](#_Toc39410171)

[CHAPTER 5 33](#_Toc39410172)

[RESULTS / DISCUSSION 33](#_Toc39410173)

[5.1 Introduction 33](#_Toc39410174)

[CHAPTER 6 34](#_Toc39410175)

[CONCLUSIONS / FUTURE WORK 34](#_Toc39410176)

[6.1 Conclusions 34](#_Toc39410177)

[6.2 Future work 34](#_Toc39410178)

[6.3 Legal, Social, Ethical and Professional Issues 34](#_Toc39410179)

[6.4 Synoptic Reflections 36](#_Toc39410180)

[ReferenceS 37](#_Toc39410181)

[Bibliography 38](#_Toc39410182)

[Appendix A 40](#_Toc39410183)

[ 40](#_Toc39410184)

**Note:** To insert TOC here follow the instruction below. You may also right click on the above text and use “Update Field” to update the TOC.

Insert > Reference > Index and Tables … > Table of Contents

List of Figures

[Figure 1: Microsoft XP 5](#_Toc125880397)

**Note:** To insert List of Figures here follow the instruction below. You may also right click on the above text and use “Update Field” to update the list of figures.

Insert > Reference > Index and Tables … > Table of Figures > Caption Label => Figure

List of Tables

[Table 1: Microsoft Office. 5](#_Toc125880862)

**Note:** To insert List of Tables here follow the instruction below. You may also right click on the above text and use “Update Field” to update the list of tables.

Insert > Reference > Index and Tables … > Table of Figures > Caption Label => Table



Introduction

Introduction

### Scope

This report will focus as discussed in the abstract on a new design of logic gate simulator for educating A-Level students on the functions of logic gates. The new piece of software intends to target the issue discussed below and as a result provide a better means of educating students.

In order to get a better understanding of the topic at hand this report will delve into the general subject area surrounding logic gates and their education. This involves the areas of Computer Science, education techniques and various methods of educating students on logic gates.

### Relevance

Like why this is important….

## Background

Logic circuits are a fundamental part of all computer science related education programs. An understanding of these gates, how they come together to form circuits, and how these circuits are used is necessary for the successful study and implementation of the technological systems from which they are composed.

### Logic Gate Simulators

Logic gate simulators attempt to simulate logic gates and their behaviour when combined into circuits. They achieve this at varying levels of physical detail, such as at the transistor, gate, electronic system, or behavioural levels.

The primary use case for these simulators, especially those that are more complex, is for circuit design verification. By allowing users to directly interact with their designed model, they can see it in action without having to physically build it themselves. This dramatically reduces development costs as circuits only need to be built once users are assured the logic behind the circuits is functional to their requirements.

1.2.1.1 Event Driven vs Cycle-Based Simulation

Currently, there are two main methods of logic gate simulator software design, traditional event driven simulators, and cycle-based simulation. These differ in the type of models they use to process changes.

1. Event driven simulators essentially imitate the function of a circuit and its components. They involve connected modules (gates) which handle events, the modules produce an output which if connected, triggers an event in the connected module, essentially propagating the event. This simulator design method handles only necessary events, meaning simulation of the entire circuit at one given time is not required.
2. Cycle based simulation involves simulation of a circuit on a cyclic based timing schedule. The circuit itself is compiled into a set of instructions which are executed each clock cycle. This form of circuit simulation is better for analysing the timing efficiency of a circuit since the compiled model closer represents the timing of a real-life circuit. As a result, this form of simulation is used more so for circuit design verification.

(Where they run ie. desktop)

(Maybe give some example of logic gate simulators here)

### Use of Logic Gate Simulators for Education

Using software as a teaching method.

Logic gate simulators can be used as an education tool to teach students the behaviour of gates and how they come together to form circuits. This report shall focus on logic gate education for A-Level students. From this perspective, features are usually from the gate/transistor level upwards.

Most of these tools make use of the experiential learning process, defined as ‘Learning through reflection on doing’ [1]. Various studies [2][3] demonstrate the effectiveness of the experiential learning process. These methods will be discussed further in detail within the context section of the report.

1.2.3.1 Recent Work

## Issues with Using Logic Gate Simulators for Education

Without a challenge or problem to overcome, getting students engaged in the learning process can prove difficult. Current logic gate education software tools fail to use challenge-based learning to achieve their goal. This is unfortunate, since challenge-based learning, based off-of experiential learning, is proven to be effective in engaging students in the learning process [4][5].

## Report Layout

A literature review will be performed within the context section of this report. Here, various other methods of logic gate education will be looked at. This review avoids accidentally repeating a technique that has been tried before ensuring that the method is entirely unique. The creation of this section will also aid in shaping the method and ensuring its validity.

The new ideas section of the document will outline the planning of the implementation of this new piece of software. This will outline the various stages of implementation and introduce some of the tools used in the process.

The design of the software will then be discussed within the implementation section. Here, the structure of the design will be explained so that readers can replicate the work. How the tools are used for the software will also be discussed here along with a display of the product itself.

Results and the overall success of the software on tackling the goal it set out to overcome will be discussed within the discussion section of the report. Finally, the report will be summarised within the conclusion section of the report.



CONTEXT

Introduction

The purpose of this section of the report is to review work within the general field of logic gate education. This avoids accidentally repeating a technique that has been tried before ensuring that the new method is entirely unique. The creation of this section will also aid in shaping the method and ensuring its validity. This is done though identifying gaps within existing approaches to logic gate education, showing the purpose of the new solution.

There are three main methods of logic gate education, many A-Level students are taught through a combination of these methods. Firstly, classroom theory usually from textbooks or their teacher. The other two methods involve putting their knowledge to use through practical engagement, physical experiments with electronic circuits and software simulations [xxx]. This literature review will investigate all three.

## Existing Solutions to Logic Gate Education

### Books or Online Research

Books are the main method of educating students on logic gates. Books, alongside teachers in classrooms, are used to teach students the theory of logic gates. Without the theory, engaging in experimental or practical based learning would be very difficult. This is because students require knowledge of the functions of logic gates before they can engage in using them.

Any A-Level computer science book should have theory information on logic gates and their functions. For example, Computer Science by Bob Reeves (<https://www.amazon.co.uk/AQA-level-Computer-Science-Reeves/dp/1471839516>). Students are usually given or requested to acquire these books when at school.



**Figure 1: Computer Science by Bob Reeves**

Students also make use of online resources for acquisition of theory knowledge of logic gates. As an example, khanacademy.org has all the information of basic functions of logic gates, enough for passing at A-Level. Both books and online resources also provide test questions, these allow for the student to see if they have correctly learnt theory.

While books and or online resources technically give students enough information for A-Level computer science logic gate knowledge, they do have downsides. On average people retain 25% of what they hear, 45% of what they hear and see, and 70% of what they hear, see and do. (Edwards 1985 ppp). These learning resources fail to make use of the ‘learning by doing’ education approach. As a result, learning using these methods is not as efficient as it could be. This study shows engaging students in some sort of practical exercise, in combination with initial theory would be greatly beneficial for them.

### Using Models

Module kits

There are many companies which sell logic gate modules for education, an example being Sphero Inc’s littleBits, figure 1 shows an example of one of their modules. (<https://classroom.littlebits.com/lessons/introduction-to-logic> picture from there to)



**Figure 2: A Sphero Inc’s littleBits Module**

The modules are individual gates which connect with other modules to form circuits. These companies attempt to offer a low-cost hands-on method for learning about logic gates. This method engages students in experiment-based learning a method proven, by many studies [yyy][zzz] to be effective within the classroom.

A paper on a proposal for a new system of 3D printable logic gates for students to learn about logic gates, demonstrates the validity of this method, as it shows research within the topic area is ongoing [fff].

Using module kits, while successful in part at educating A-Level students on logic gates, require theory in order to be used. Therefore, they act more as a supplementary aid in education, and cannot replace theory teachings.

Using a logic gate module kit for learning also does have its downsides as well. For instance, acquisition of the learning material must be done well before classes to ensure enough material exists for all students. While this material is low-cost it still does impact on education systems budgets. These modules are also physical, students may end up breaking, stealing or loosing them. Education using the modules is limited to within the classroom students taking them home would be impractical. Demonstrating the use of these in front of the class may prove difficult as well due to their small size. Students would need to huddle around the table on which they are being used. If the teacher is the only person to have the kit within the classroom then there is little benefit as compared to theory on a whiteboard. This would also not be making use of experimental-learning process.

A logic gate model board

An article on the creation of an E-Logic Trainer Kit explains the design as well as effectiveness of the prototype within an educational setting (<https://online-journals.org/index.php/i-joe/article/view/11410>). The abstract of the study outlines an overall positive response to evaluations, with 60% of the correspondence giving positive feedback.



**Figure 3: E-Logic Trainer Kit**

This model has similar teaching benefits to that of the module kits mentioned above, mainly that it engages students in experimental learning in a hands-on manner. However, it also shares the downsides of the model kits with the additional downside of its overall bulkiness.

### Simulator Software

There are various logic gate simulator software applications available. These fall into two main categories, those for education and those for circuit design verification. There is however some overlap. These simulators run on various platforms, as discussed below.

One popular open source logic gate simulator which runs on windows and mac OS computers, designed for education is Logisim (<http://www.cburch.com/logisim/>). This simulator is used by many schools and universities in classes ranging from GCSE Level to computer architecture courses. A conference paper investigated using Logisim as an educational tool [hhhh].

The results of a survey on the tool was in the paper. The average rating for all questions on the first part of the survey was 4.55 out of 5 (5=Completely agree, 4=Mostly agree, 3=Partly agree, 2=Mostly disagree, 1=Completely disagree), for questions relating to the effectiveness of the tool itself and in education. The second part of the survey was consisted of questions of a similar manner however asked in a more open-ended way. Nearly all the feedback within this section was positive, pertaining to “its effectiveness, system independency, and its ability to check and simulate the functionality of designed circuits using only a hand tool”.

From this survey it can be concluded that software simulators of logic gates are a very effective tool for practical learning of these gates. In many ways the software tools are a much better method of education than other practical engagement methods discussed above. Most simulator software is free, there are no limitations on the number of gates students can use. Circuits can be constructed and deconstructed much quicker and easier than physical education tool competitors. Demonstrations are much easier to perform within the classroom. Circuits can be setup, saved and returned to later. This method does require students to have computers, however in todays day and age, this is no longer much of an issue.

This article within the European Journal of Engineering Education also investigated the effectiveness of software simulators for educating students [uuu]. The abstract demonstrates similar results were concluded to that of the study conducted on Logisim.

Other platforms

Educational logic gate simulator software also reaches platforms other than those for just desktop/laptop computers. An article on the Web Based Interactive Digital Logic Circuit Simulator[qqq] is an example. The article itself has no survey results, however its existence is proof to a certain extent on the validity of such an approach. Mobiles can also be a target platform for these simulators, a conference paper done on the development of a logic gate simulator for mobiles[jfk] for education purposes demonstrates this. How effective this would be within an educational setting is up to debate given the limitations of phone hardware and students tendencies with distraction.

* Mention limitations of research, which is where you build off for your solution
* As a result of your literature review you should be able to elaborate on the limitations of existing methods of solution for your particular problem.



New Ideas

Introduction

This part of the report is to justify the new design based off an analysis of the weak points in the methods discussed within the previous chapter. This new design with the tools used for it will then be outlined, and the planning that went into the project will be demonstrated.

…

As mentioned before, engaging students in the learning while doing process has been proven as a very effective method for educating them. However, without proper incentivisation for learning, students may not be as engaged as possible.

While textbooks and online resources do offer questions, they do not engage students in practical hands-on style problem solving, and instead demand theory style answers. The current methods of education aside from textbooks and online resources, fail to engage students in challenge-based learning. This is where justification for the new design begins to shine through. Challenge-based learning, based off-of experiential learning, is proven to be more effective in engaging students in the learning process [4][5]. Essentially, without challenges to overcome, students won’t find as much use in practical engagement with software/hardware models of gates since humans are goal-oriented thinkers. Usually to overcome this, teachers, books or online resources provide questions such as truth table to circuit and circuit to truth table conversion tasks.

Having the challenge mode within the new piece of software acts as a software solution for the providing of questions, which means students won’t have to refer to external sources for the questions, making the learning process more streamlined. As an additional benefit the “level” layout of the feature should engage students even more thanks to its game like style. This goal-oriented nature of the design means students are engaged in challenge-based learning as opposed to just learning while doing.

As a result of the analysis within chapter two it is obvious that a software implementation of this solution works much better than that of a hardware one. Firstly, the cost for students is free, thanks to no manufacturing cost for hardware along with no risk of losing or breaking components required for the teaching. Having the challenge element built into the software removes the need for paper. And the interface provides an easy method for teachers to construct the challenges for the students.

The programming design model used by this new piece of software is the event-driven based model, as apposed to the cyclic based model (discussed in the introduction). This is because the cyclic based approach, as a more complex model, is unnecessary for the type of logic gate simulator designed. At the A-Level stage of education it is not necessary to consider the timing of the circuits being designed, as education is focused solely on the functionality of circuits.

Further justify the idea..

Why windows OS…

## Requirements

For the new idea to be achieved a piece of software has been created. This software had the following requirements set out during its creation.

* To model all logic gates taught in A-Level Computer Science classes, with the ability to link them together to build circuits. These gates should be placed, moved and removed from a “field” which the user should be able to pan over and zoom into.
* The ability to save and load previously made circuits and pages of circuits. This will be done through saving/loading page files containing information of gates, locations and links between them as well as custom circuit files. This also should give the added benefit of allowing students/teachers to share these pages or prebuilt circuits.
* Provide a user-friendly graphical user interface. Interactions with gates and circuits and operations must be obvious to the user. This keeps the learning curve to a minimum and allows for a smooth learning process.
* The software must be robust in terms of performance. This will be ensured through unit and user testing.
* The software must have a challenge mode, in which teachers/students can create truth table to circuit and circuit to truth table conversion-based challenges for students. These challenges must be able to be saved/loaded as well as transferred between computers. The software must keep track of completed challenges. This feature must be designed in a game level like manner to keep students engaged.

How requirements relate to background research.

## Project Planning

### Methodology Used

The project was developed using the Waterfall software development methodology. This method works by only moving onto the next step in the development process once the previous step has been completed. Waterfall happens to be a perfect fit for the size of this project. Choosing this development methodology made sense due to the various dependencies between deliverables discussed below, and the fact that the method works very well for this. This methodology also makes the entire development process easy to document, beneficial for the creation of this report.

### Deliverables

The development process was planned out before it began. The work was sectioned off into different deliverables to be completed by certain milestones. This was to balance the workload and ensure there was enough time to complete the project. Below are these deliverables, followed by a breakdown of them.

1. Gate simulator with basic gates that are taught in education
2. Friendly UI method for selecting circuits to analyse
3. Generate a test framework for the gate simulator
4. Challenge mode functionality
5. Evaluation of the software – user testing
6. Completed dissertation

**3.3.2.1 Gate Simulator with Basic Gates**

The gate simulator acts as the base for the other software deliverables. It contains the main hub to be used to navigate to the various features of the software, along with the basic gate functionality, saving and loading features. Tasks to complete this deliverable include:

* Research programming architecture
  + Qt architectures
  + General architectures
  + Code style of existing logic gate simulators
* Research good educational aspects of existing simulators
  + Ask students & teachers
  + Review existing simulators
* Gates
  + Design images
  + Develop classes & functionality
  + Design links
* Design home page including:
  + Gate selection panel
  + Gate information panel
  + Design gate page class (Class to hold all gates on a page)
  + Operation buttons panel (Move, drag, break link, undo/redo, select, delete gate)
    - Design images
    - Develop functionality
  + Truth table to circuit dialog
  + Karnaugh map dialog
* Saving/loading
  + Develop loading class
  + Add save functions to gates & pages

**3.3.2.2 Friendly UI method for Selecting Circuits to Analyse**

Depends on: Deliverables 1

This deliverable consists of the tasks:

* Develop a gate selection class
  + Research into what makes a friendly UI
  + Develop class
  + Add on truth table & optimization functionality options (Not developed)
  + Add multi select

**3.3.2.3 Test Framework for Base Simulator**

Depends on: Deliverable 1

Unit testing is a type of software testing in which individual components/units of a software are tested. This allows for validation of the functionality of the individual components.

* It increases confidence in changing/maintaining code. When well written unit tests are run on every system change, errors can be caught quickly preventing these changes from being added to version control.
* It makes code more reusable. This is as a result of the forced format which code is forced into in order to make unit testing possible.
* Generally, the cost of fixing an error is lessened in terms of time and effort for unit tests in comparison to normal debugging. This is because unit tests make the location of the error immediately apparent.
* It provides a basic documentation of the system. Reading unit tests gives developers a better understanding of the features being tested within the software system.

This was used in order to ensure correct functionality of the base gate simulator. Its tasks include:

* Choosing a testing framework system to use
  + Research into different testing frameworks
* Development

**3.3.2.4 Challenge Mode Functionality**

Depends on: Deliverables 1 & 2. Tasks include:

* Developing a:
  + Challenge builder dialog for teachers
  + Challenge list saving capabilities
  + Challenge list loading capabilities
  + Challenge dialog for students
  + Results dialog for students & teachers
* Implementing saving/loading functionality

**3.3.2.5 Evaluating the software**

Depends on: All deliverables (Excluding 6)

In order to evaluate the software, university students, who are already familiar with A-Level logic gate education, will be asked their opinions on the effectiveness of the software has on achieving it’s goal.

**3.3.2.6 Writing dissertation**

While not technically depending on any of the deliverables, an application of some sort would need to be developed in order to complete the report. The report also acts as an evaluation of the entire projects process, and therefore has been added as the final deliverable.

### Milestones

The deliverables were planned to be completed by certain milestones to ensure completion of the project within the allotted time. Table 1 contains these milestones and the deliverables which needed to be completed by their dates.

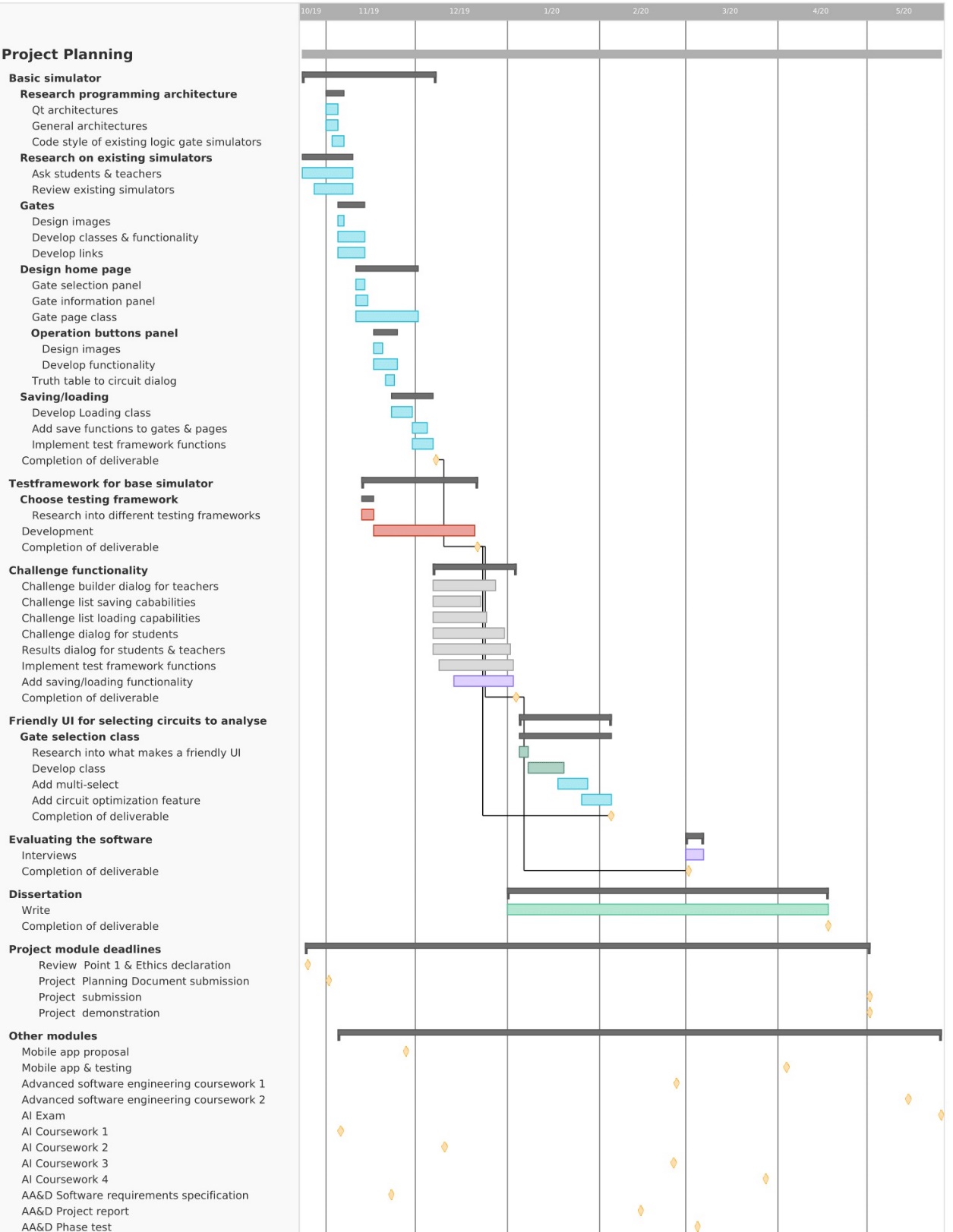
**Table 1: Milestones**

|  |  |  |
| --- | --- | --- |
| Milestone | Date | Deliverables completed |
| Review Point 2 | 06/11/19 | 1 & 2 |
| Tutorial 3 | 20/01/20 | 3 & 4 |
| Tutorial 4 | 01/03/20 | 5 |
| Project submission | 24/04/20 | 6 |

### Workload Balancing

In order to gain a visual representation of the workload of the project and other work over the coming months, a gantt chart was created. The chart helped with planning a balanced workload and setting the completion times of the deliverables within the milestones outlined above.

**Figure 4: Gantt chart**



Gates to be used?

### Tools and Resources

3.3.5.1 Why C++

Various programming languages have been used in the development of logic gate simulator software before. However, for this project C++ was chosen, this was due to several reasons. Firstly, the developer was already familiar with the language therefore no learning of language features was required. Its object-oriented style fit well with the creation of the gate objects and the pointers functionality of C++ made the modelling of links between these gates simplistic. Its multithreading capabilities allowed for the creation of timer gates, and simultaneous updating of both GUI and logic. Lastly, due to the low-level style of the language, resource consumption of simulations could be kept to an acceptable level.

3.3.5.2 Tools

Once the programming language had been decided, tools used to create the software were looked at. Below in table 2 are the tools used along with their purpose and the reason behind their choice.

**Table 2: Tools**

|  |  |
| --- | --- |
| Tool & purpose | Reason |
| Desktop. Development tool. | Used to host the software needed for developing the application. |
| Qt creator IDE. Environment for developing the software required for the project. | One of the best tools for creating C++ applications with a GUI interface. This is due to its drag and drop interface constructor. QT applications also benefit from being cross platform compatible with little effort, a potential for future development. |
| QTest framework. A testing framework for unit tests on developed software. | An evaluation of other testing frameworks was performed, while there were many acceptable options, the QTest framework was a good choice due to it competing against other frameworks while also being an extension of the IDE in use, making setting it up much easier. |
| Paint.net. An image editor for designing images of buttons, icons, and gates used in the simulator. | Paint.net is a reasonable choice due to its simplicity making it easy to use and learn, while still containing enough functionality to create the required assets. |
| GitHub. Version control software acts as a storage method for files related to the project. | Github is industry standard, allows for saved iterations of the software & documents, preventing loss of work. This also means if errors are encountered during development a rollback to previous iterations is possible. |
| Notes document. To record relevant information about the project which may come in use later. | Perfect for making general notes to use in the development process, quick and easy, already a part of the windows operating system, which the system was developed on. |

3.3.5.3 Resources

To conduct the project, the following sources of information are required:

**Table 3: Resources**

|  |  |
| --- | --- |
| Information source | Reason |
| Qt documentation | Used to become familiar with and overcome issues related to the IDE, base library, and unit testing framework. |
| Google | Used for research into.   1. Programming architecture of simulators 2. Programming languages 3. Design styles 4. Testing frameworks 5. Literature review documents   Also used as a method of accessing other information sources. |
| Competing software | By analysing competing software, a list of necessary components to be used in the core simulator was derived.  Since not all competing software is freely available, there are some constraints as to the extent they can be analysed. To overcome this as best as possible, documentation and YouTube reviews of the paid software was used. |
| Education websites & literature | Provide information on exam style questions used for logic gate education to get a better idea on how to formulate the challenge designer. This was also used in the literature review for formulating the solution. |
| Educators & Students | Provided information on what works well and what features are missing in existing educational software. Students also helped in testing of the software. Educators also acted as an aid for technically demanding areas of development. |

### Contingency Planning

During planning various risks were determined to pose a threat to development of the project. Of the known risks, potential solutions were planned out to avoid unexpected disaster as best as possible. These are outlined in the table below. As seen in the Gantt chart, no work was planned out for most of March. This was done intentionally to create a time buffer to compensate for lost time.

**Table 4: Contingency plans**

|  |  |  |  |
| --- | --- | --- | --- |
| Risk | Probability (out of 5) | Cause | Potential solution |
| Technical requirements found to be too complex for developer | 1 | Developer lack skills in the scopes of the technical requirements. | Cutting out aspects which prove too technically demanding. |
| Friendly UI not done in time. | 2 | A delay in development. | Cut back on time consuming aspects, opting for functionality over presentability |
| Too many features to develop or a delay in development. | 4 | Time limitation / unexpected circumstance. | Focus on features which make software unique to retain its value. May potentially retarget the software for GCSE students instead. |
| Computer used to develop software breaks. | 1 | Damage to computer. | Using alternative computers provided by university. |
| Insufficient feedback from students/educators during research phase | 2 | Students fail to answer questions.  Educators fail to identify needs in software. | Find more students/educators. |
| Significant amount of feedback from students/educators is redacted | 2 | They feel the information provided may be incorrect, or do not wish it to be used for the project. | Find more students/educators. |
| Developers become sick | 3 | Illness | Cutting back on lower priority elements of the project, so that the project still retains its uniqueness. |



IMPLEMENTATION

Introduction

This section of the report explains the implementation process of the software, how the tools were used, along with a demonstration of the implementation itself.

### Coding Standards

Utilizing the headers and source code split of C++ allowed for a good general overview of what a class contained, such as member variables, public and private functions. Code variable, function, and class naming conventions were used. Reasons for using conventions included:

* Reducing effort needed to read and understand source code.
* To allow for easier debugging of source code, such as knowing the scope of member variables.

These naming conventions are outlined in table x below.

**Table x: Coding standards**

|  |  |
| --- | --- |
| Code Element | Rules |
| Classes | * In camel case, with capitalized first letter. * Dialogs start with ‘DLG\_’. * Gates start with ‘Gate’. |
| Functions | In camel case, with first letter capitalized, except for qt slot functions. |
| Variables | * Class member variables start with ‘m\_’. * Pointers start with ‘p’ * Dialogs have ‘Dlg’ * Widgets have ‘Widget’ * Constant variables start with ‘c\_’ |

## Base Simulator Design

### Simulator Objects

All gate models, following an object-oriented design style, have an individual gate class, all of which inherit from the ‘gate’ base class. This class inherits from the ‘draggable game object’ and ‘game object’ classes. This hierarchical structure aided development by making it easier to add additional objects to the simulator, while removing the need for duplicate code.

4.2.1.1 Game objects

The GameObject class represents anything located on a GateField (discussed below). At this level of inheritance, these objects are relatively simplistic in that they have certain basic properties and functions. As seen in the appendix code-1 these functions include positioning and dimension variables, drawing of a set image, click hitbox checking, and setting of a disabled user access property.

The draggableGameObject, also in appendix code-1, class inherits from the game object class and is inherited by most of the objects located on a GateField. This implements the ability of making the object draggable by the user. Although this could have been a feature added directly into the GameObject class, the node class, discussed below, which inherits from GameObject did not need the draggable feature therefore it made sense to keep the classes separate, along with the potential for static objects in future updates.

4.2.1.2 Gates

Upon the completion of this report there were 18 different gate models for simulation. These included basic gates taught for A-Level computer science, (AND, OR, EOR, NOT) as well as the ‘notted’ versions of the first three (NAND, NOR, XNOR) and triple input versions of these as well. There are also signal gates including, a toggle, a signal emitter, a signal receiver, and a timer gate with a customizable frequency.

All different gate classes inherit from the gate base class (appendix code 2). Table x describes some of the major functions within the gate class.

**Table x: Major Functions of Gate Class**

|  |  |
| --- | --- |
| Function | Description |
| UpdateGraphics | Called when the GateField wishes to redraw the gate. The majority of the gates do not override this function, which calls GameObject::UpdateGraphics to draw the gates icon as well as draw the gates nodes. Certain gates with custom drawing override this function. |
| UpdateOutput | This function is called to update the gate’s nodes, all gate classes override this function. A gate class which overrides this function takes the values of the input nodes of the gates, performs the Boolean operation of that gate, and then updates the gates output node. (Example in appendix code 4). |
| SaveData | This function saves the details of the gate, such as type, position and node links to an std::ofstream. All basic gates do not override this function, however some more complex ones with additional information to save do. |
| DeleteClick | This function returns weather or not the gate should be deleted given certain input parameters. It is called when the user performs a mouse click on the GateField a gate is in when the click operation is that of delete. |
| Clone | Acts as a copy constructor for a gate and returns a pointer to a Gate class. All gate classes override this function. The majority of the overrides clone the gate, its nodes and their link information. |

The gate class also has various node handling functions. These functions include abilities such as returning a node selected by a user, returning a searched node, detaching all of the gates nodes from other nodes, returning a list of pointers to either all output or input type nodes, or returning weather or not output or input nodes are connected.

All the functionality within this class that is public is accessible from any instance of the gate class within the GateFeild class, discussed below. Gates also have the hierarchy-based functions SetParent and GetParent, which act as a setter and a getter for the gates parent GateField. As a result, functions are called both down the hierarchy, from GateField to Gate, as well as up, Gate to GateFeild. A diagram representing this interaction is shown within the GateField section of this document. Having all gates inherit from the base Gate class means that a GateFeild instance has access to all gate types functionality with a pointer to their Gate base class, given that the functionality of the Gate class is overridden from the inherited gate class.

All gates except the TextLabel gate have one or more nodes. Instances of node class (appendix code 3) are used for connections between gates. Each node has a list of pointers for connected nodes, and a type variable m\_nodeType which determines whether it receives or emits a signal to/from a linked node. The node class inherits from the GameObject class. Nodes are responsible for the drawing of links, this occurs on their UpdateGraphics function call, overridden from the GameObject class. The code for the drawing of nodes and their links is demonstrated in appendix code – 3.

4.2.1.3 Gate Collections

The GateCollection class, as the name suggests, acts as a container for a set of gates usually in the form of a circuit. They were created to tackle a section of the second requirement of the system; to be able to save/load custom circuits for students/teachers to share. The GateCollection class inherits from the gate class, however it also has a vector of pointers to gates. This vector is all of the gates within the gate collection. This setup allows for the collection to be manipulated as though it is a single gate, while still having the functionality and accessibility of the gates from which it is composed.

An instance of the GateColleciton class displays a series of buttons for the control of the gate as a whole or the gates within it. These buttons include;

* A delete button, which remove the collection but keeps the gates inside it.
* A delete all button, which removes the collection object and its contained gates.
* A drag all button, which toggles between dragging of the entire collection or manipulation of individual gates.
* A save button, which saves the collections to file for loading or transferring to another computer.

In the appendix section

### Code Core Design

4.2.1.1 DLG\_Home Class

When the application starts it creates an instance of DLG\_Load, which in turn creates an instance of the DLG\_Home class. The loader dialog was created since the DLG\_Home class hosts most of the functionality of the application and loading times for it may be slow depending on the host system. The loader dialog has a progress bar which monitors the progress of the execution of the constructor of the DLG\_Home class instance.

The DLG\_Home class acts as the main window of the application. As a result, it hosts a large proportion of the functionality of the application. From this window the user can create GateField pages to create circuits on, perform saving/loading operations, manage custom circuits, and visit other dialogs. Various instances of system wide variables and widgets are hosted within this class to be used. An example of one of these variables is the DLG\_Message instance, which acts as a pop-up messaging box. The DLG\_Home class contains a public function SendUserMessage() which displays the already instantiated DLG\_Message instance with the message passed through the functions parameters. This function is called from various locations in the system, as a result instances of the message dialog are not constantly recreated, and duplicate code is avoided. The DLG\_Message variable is instantiated within the constructor of DLG\_Home as with many of the other system wide variables. As a result, initial loading times may be long, but these variables will not have to be re-instantiated keeping runtime loading times fast.

The DLG\_Home class contains many nested widgets, these include:

* Gate selection widgets, which the user uses to select gates.
* A gate information widget, which displays information of a selected gate.
* A tabbed page widget, which hosts various GateField objects.

This modular design allows for clarity and makes future updates simpler in terms of the ability to re-use certain dialogs. An example of this is the re-use of the GateField widget class for the challenge mode dialog.

When a user selects a gate within the gate selection widget it is added to the currently open GateField instance. This is managed through DLG\_Home which acts as a hub for this operation. The functionality of many similar operations, such as the pressing of buttons is handled through DLG\_Home. Diagram x demonstrates this as well as the entire direction of communication through DLG\_Home.

4.2.1.2 GateField Class

The GateField class as mentioned previously, acts as a page for circuits to be displayed on. The GateField class inherits from QWidget, a QT class, and as a result has access to its functions. It is responsible for the handling of the gates which make up these circuits, this includes drawing the gates, updating their outputs and managing their positions. GateField instances can be saved to or loaded from disk.

The gates within the object are stored within a vector of pointers to the gates in memory. Having the gates as pointers means that a single container can be used for many different gate types which inherit from the Gate class. Certain operations of the GateField class involve looping through the contents of this vector calling a specific function on each gate.

An example is the paintEvent function which is overridden from the QWidget class it inherits. Each time an object instance is redrawn, this function is called. The function as well as performing additional operations loops through the container of gates calling each to be redrawn. In future updates, a variable determining weather or not an individual gate should be redrawn could be added to ensure the re-drawing is only done when necessary.

Undo and redo operations also make use of this vector of pointers to gates. After each major action a copy of all gates within the vector is added to another vector acting as a backup. Previous versions of this vector can then be loaded into the main vector. As a result of asynchronous events on the vector of gates, the GateField class makes use of a mutex object, m\_lockAllGates, in order to avoid crashes.

The QT library came in much use for this class. GateFields make use of a lasso Qt tool in order to make selections of circuits. These selections then turn into GateCollection objects. Thanks to the QPainter class, drawing and performing zoom operations was also made easy.

4.2.1.3 Saving and Loading

Saving and loading operations are called from various locations within the application. These are performed on both GateFields and GateCollections. When saving an std::ofstream is passed to the container of the gates, which is then passed to each individual gate which writes its information to the file. For loading, in order to simplify the code and avoid code duplication the class GateReader was created. It is a passed an std::ifstream of the file and the container to load the contents into. The class then reads the file, creating the gate objects and the links between them.

In order to save the information of gates various things were introduced. Firstly, the GateType enum was added to the Gate class. This assigned each gate with a specific number which would be saved to or read from a file. The enum also had the additional benefit of circumnavigating type casting inorder to determine the class of a pointer to a Gate. In order to save the information of links between gates the Node class was assigned an id variable. Links are saved as a set of two node ids.

4.2.1.4 Memory Management

As a result of the C++ language, memory management needed to be handled. All classes member variables are deallocated within destructors if they are instantiated anywhere else. Memory creation to a functional scope was also ensured to deallocate any allocated memory. Finally, tests were run on the system using task manager to ensure running the application caused no memory leaks.

Memory results::

### GUI

4.3.3.1 Images

Images were designed using paint.net. The applications produced various png image files which were added to the QT resource file for the logic simulator. In terms of images for buttons, their design followed that of similar applications to not confuse the user, this was a focus during user testing. All these images followed a similar grey colour, minimalist design style. This was also the case for the gate images. When gates are drawn their respective image is drawn onto the screen, and then using QT’s paint library nodes are drawn on top.

4.3.3.2 ScrollBar Class

The scrollbar class is used in various locations of the applications. For instance, to change the zoom level on a GateField or to scroll through the various gates on the gate list dialog. Use of QT’s library came in handy when designing this class. The class itself is drawn with aid of QT’s paint functions and objects, and the mouse scroll ability of it makes use of QT’s event functions.

4.3.3.3 Overall Design

Thanks to QT’s GUI designer a lot could be done with the overall look of the application. The colour palette of the application was light and dark greys with white and some bright green to draw user’s attention to important features. This palette was intended to match the ‘techy’ goal of the application. The appendix section x shows an example view of the applications main window.

## Task Feature

### Development

The task feature was intended to tackle the final requirement of the system; a challenge mode designed in a game level like manner to engage students. Due to this being quite a large feature, a separate dialog was created for it. In order to reduce code duplication, the challenge task dialog, dlg\_task, inherited from the DLG\_Home class. A large proportion of the features of the DLG\_Home class were not actually used within the dlg\_taks class however, resulting in redundant code in terms of inheritance. Potential future updates to fix this would be creating a third dialog class which hosts the functionality shared between DLG\_Home and dlg\_task, from which they would both inherit.

The intention of the dlg\_task class is to display the task required for the user to perform. As mentioned before, these tasks include filling in a truth table from a displayed circuit or creating a circuit from a displayed truth table. In order to verify the results of the tasks the displayed circuit is executed, and the resultant outputs of the circuit are then cross verified with the values of the truth table. Appendix x demonstrates the code for this.

The dialog classes DLG\_CircuitTaskDesigner and DLG\_TruthTableTaskDesigner are both used to create the tasks for students to perform. They both produce a task file which is saved to memory.

The dialog class DLG\_TaskManager is used to display tasks to a user. Once run, it reads files in the applications task folder location, storing their names in an array. The completion result of the tasks is stored in the file names. The dialog then displays a series of buttons representing each task, the buttons are coloured green or grey depending on weather the task they represent is completed.

The modular approach of different dialogs for different stages of the task creation, allows the user to easily understand the process of task creation. This also adds the benefit of making updates to the code much easier, as a section can be updated while easily ensuring other sections are not affected by the update.

### Result

Potentially discuss theme.

## Unit Testing

In order to perform the unit tests a second QT project was created. The make file of the test project linked the header, source and UI files of the main project. This meant that unit tests were always run on the latest system code. The project consisted of a single header and source code file, with the class Tests. This class contains various functions to test the features of the application including:

* Logic gate function tests. This consists of a test function for each of the gates within the application.
* Tests for links between logic gates, including some test circuits. This consists functions which build circuits and then run them checking they produce the correct results.
* Tests on the loading and saving of custom gates. This involves creating a circuit which is then saved to a file, and later re-loaded. The loaded circuit is then checked to be the same as the saved one.

ssadm

Userfriendlyness, HCI



RESULTS / DISCUSSION

Introduction

The technique developed in your project is supposed to show improvement on techniques previously available. Therefore it may be necessary to spend time investigating whether this is true. Perhaps you need to set up some sort of quantitative test and do a little statistical analysis to confirm the improvement. This chapter will provide evidence, from the tests that you carry out, of the outcomes of your project.

Explain the success and limitations of your work and show how this relates to the aims and objectives set out in the introduction.



CONCLUSIONS / FUTURE WORK

## Conclusions

Whatever it was that your results showed should be summarised here. Your project or may or may not have achieved all that you set out to at the start.

This is your opportunity to conclude whether the project was a ‘success’ and how it might have been tackled differently in hindsight.

## Future work

In either case there should be some reference to future work, either to forward and expand on the successful outcome or to test ways of overcoming the shortfall in your ideas that didn't work out quite as expected but there should be something that shows you can see further implications of what you have achieved.

## Legal, Social, Ethical and Professional Issues

This section should include a discussion of the four LESPIs and the way in which you project has/will/could impact on each.

* Describe the four LESPI’s

Legal issues could be improved by considering relevant legislation, e.g. GDPR, Accessibility Legislation. Some points would benefit from more discussion, e.g. your intent to release the software as open-source (why are you doing this, what implications could it have?). For professional issues, you could look at a Code of Conduct from a professional body, e.g. BCS, and see if anything is relevant to the project.

During the research phase of the project, students and teachers will be interviewed on the features of logic gate simulators. In order to ensure this does not raise any LSEPIs, each interviewee with be given a participation consent form which they will sign. The form will outline what the interviewee will be subject to during the interview, the fact they will remain anonymous, as well as provide contact details so they can redact provided information later if they feel the need to do so.

A similar form will also be used during the evaluation phase of the project, in which interviewees will be asked to evaluate the software produced.

Once the software is published, it will be released as open source and free to download. Due to this it will be licenced with the MIT license.

When being used in the classroom if the software fails to emulate gates correctly, students may become misinformed on the interaction or properties of logic gates. To ensure any issues which arise because of this are covered, a declaration stating that the software may not be entirely accurate will be added to the release documents.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Legal issue | Social issue | Ethical issue | Professional issue |
| Interviews |  | X | X | X |
| Software release Licencing | X | X |  | X |
| Use of the software |  | X | X | X |

## Synoptic Reflections

This section will comprise of a reflection on the project in relation to employment aspirations and the skills that you have developed towards this through engagement with the project.

ReferenceS

Vogt, C. 1999. Creating Long Documents using Microsoft Word. Published on the Web at the Nottingham Trent University.

**Note:** References are a list that includes the essential bibliographical details for each item to which you have referred in the body of your paper. It should ONLY include items to which you have made direct reference. A direct reference is where you have quoted/reproduced text or diagrams from another author or mentioned/referred to the work of another author in your report. That is quoted directly what they have said about something or mentioned their views or conclusions in your report. For details of citation and references see the information in the Project Guide.

A Bibliography is a list of published materials that you have read or consulted for general information in the preparation of your work, concerning the subject of your Project, but have not made any direct reference to in your report i.e. 'background reading'.

You should always provide a Reference List. **A Bibliography is optional but when provided it should include all items in your Reference List as well as any additional items consulted in preparation of your work.**

Bibliography

Vogt, C. 1999. Creating Long Documents using Microsoft Word. Published on the Web at the Nottingham Trent University.

Coote, H., Dobbs, B. & Jones, C. (1996). Defining databases. Wiley: Melbourne.

Applications and Science in Soft Computing, Lotfi, Ahmad; Garibaldi, Jonathon M. (Eds.) 2004, X, 346 p. Springer, ISBN: 3-540-40856-8

**Note:** A Bibliography is a list of published materials that you have read or consulted for general information in the preparation of your work, concerning the subject of your Project, but have not made any direct reference to in your report i.e. 'background reading'.

You should always provide a Reference List. **A Bibliography is optional but when provided it should include all items in your Reference List as well as any additional items consulted in preparation of your work.**

[qqq]<https://www.researchgate.net/publication/318361261_WEB_Based_Interactive_Digital_Logic_Circuit_Simulator>

[jfk]<https://www.researchgate.net/publication/320468266_Development_of_A_Mobile_Learning_for_Digital_Logic_Gates_in_Vocational_Schools>

[uuu]<https://www.tandfonline.com/doi/abs/10.1080/03043797.2016.1196344?scroll=top&needAccess=true&journalCode=ceee20>

[hhhh]<https://www.researchgate.net/publication/309907692_Using_Logisim_Educational_Software_in_Learning_Digital_Circuits_Design>

[fff]<https://www.researchgate.net/publication/305751603_Teaching_Digital_Logic_Circuit_Design_via_Experiment-Based_Learning_-_Print_your_own_Logic_Circuit>

[ppp] Edwards, M. (1985). The Mercedes Benz of interactive video. Hardcopy, 14. 74-80

[XXX] C. Hacker, R. Sitte “Interactive Teaching of Elementary Digital Logic Design With WinLogiLab”, IEEE Transactions on Education, Vol. 48, No. 2, 2004, pp. 196-203

[yyy] Kruger, D., Kruger, J. and Suzuki, R. (2015). *Assessing the Effectiveness of Experiential Learning in a Student-Run Free Clinic*. [online] researchgate.net. Available at: https://www.researchgate.net/publication/276847418\_Assessing\_the\_Effectiv-eness\_of\_Experiential\_Learning\_in\_a\_Student-Run\_Free\_Clinic.

[zzz] Ban, Y., Okamura, K. and Kaneko, K. (2017). *Effectiveness of Experiential Learning for Keeping Knowledge Retention in IoT Security Education - IEEE*

*Conference Publication*. [online] Ieeexplore.ieee.org. Available at: https://ieeexplore.ieee.org-/document/8113334.

Appendix A

PUT USER DOCUMENTATION

Code 1 – GameObject and DragableGameObject classes (gameobject.h)

# 

Code 2 – The gate class (gate.h)



Code 3 – The node class (gate.h)



Code 4 – And-gate example override of UpdateOutput function

Code 3 – Drawing of nodes and links between nodes (gate.cpp)



Task design verification.



Gate Images 1 – Example of a GateCollection object



The content of these will differ with the different types of project. Any design and analysis charts/diagrams will be included here in full. In projects where software has been developed there will be an appendix for this. Our departmental requirement is that a CD, DVD or USB memory stick of all source code is submitted to your project supervisor. The appendix contained in the report will refer to this CD, DVD, or USB memory stick, provide a directory style listing of the files submitted and instructions for rebuilding and running the software. This might be source code of programs written in high level languages (C, C++, etc) together with any pertinent files ('make' files, non-standard libraries, etc). Alternatively, or in addition, you can place some or all of the source code in the appendix. In any case the source code needed to reconstruct any software you have developed must be submitted in its entirety in the CD, DVD, or USB memory stick. (Any code that has been used from a third party should reference the original developer).

Hardware designs will require schematics/circuit diagrams, PCB layouts, simulation tests and pin outs.

Most projects will require some form of user documentation to explain how to use the software/hardware produced. A researcher following up the work may wish to utilise the work of the original author and an appendix laying out the format of input files and how to interpret the output is required.