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

* Neil Sculthorpe. Acted as personal tutor during the year of project development. Provided support with the project in terms of aiding in project planning, offering resources, and engaging in project development review points.
* Ferreira, Joao Filipe. Helped with formulating the ideas for the project.

Table of Contents

[Abstract ii](#_Toc40307486)

[Acknowledgements iii](#_Toc40307487)

[Table of Contents iv](#_Toc40307488)

[List of Figures viii](#_Toc40307489)

[List of Tables ix](#_Toc40307490)

[CHAPTER 1 1](#_Toc40307491)

[Introduction 1](#_Toc40307492)

[1.1 Introduction 1](#_Toc40307493)

[1.1.1 Scope 1](#_Toc40307494)

[1.1.2 Relevance 1](#_Toc40307495)

[1.2 Background 2](#_Toc40307496)

[1.2.1 Logic Gate Simulators 2](#_Toc40307497)

[1.2.2 Use of Logic Gate Simulators for Education 3](#_Toc40307498)

[1.3 Issues with Using Logic Gate Simulators for Education 4](#_Toc40307499)

[1.4 Report Layout 4](#_Toc40307500)

[CHAPTER 2 5](#_Toc40307501)

[CONTEXT 5](#_Toc40307502)

[2.1 Introduction 5](#_Toc40307503)

[2.2 Challenge Based Learning 5](#_Toc40307504)

[2.3 Existing Solutions to Logic Gate Education 6](#_Toc40307505)

[2.3.1 Books or Online Research 6](#_Toc40307506)

[2.3.2 Using Models 7](#_Toc40307507)

[2.3.3 Simulator Software 10](#_Toc40307508)

[CHAPTER 3 12](#_Toc40307509)

[New Ideas 12](#_Toc40307510)

[3.1 Introduction 12](#_Toc40307511)

[3.2 Justification of Application 12](#_Toc40307512)

[3.3 Requirements 14](#_Toc40307513)

[3.4 Project Planning 14](#_Toc40307514)

[3.4.1 Methodology Used 14](#_Toc40307515)

[3.4.2 Deliverables 15](#_Toc40307516)

[3.4.3 Milestones 18](#_Toc40307517)

[3.4.4 Workload Balancing 19](#_Toc40307518)

[3.4.5 Tools and Resources 20](#_Toc40307519)

[3.4.6 Contingency Planning 24](#_Toc40307520)

[CHAPTER 4 27](#_Toc40307521)

[IMPLEMENTATION 27](#_Toc40307522)

[4.1 Introduction 27](#_Toc40307523)

[4.2 Design Method 27](#_Toc40307524)

[4.3 Coding Standards 27](#_Toc40307525)

[4.4 Base Simulator Design 28](#_Toc40307526)

[4.4.1 Simulator Objects 28](#_Toc40307527)

[4.4.2 GUI 38](#_Toc40307528)

[4.5 Task Feature 39](#_Toc40307529)

[4.5.1 Development 39](#_Toc40307530)

[4.6 Unit Testing 41](#_Toc40307531)

[CHAPTER 5 43](#_Toc40307532)

[RESULTS 43](#_Toc40307533)

[5.1 Introduction 43](#_Toc40307534)

[5.2 User Testing Questionnaire and Results 43](#_Toc40307536)

[5.3 Tackling of Objectives 46](#_Toc40307537)

[CHAPTER 6 47](#_Toc40307538)

[CONCLUSIONS / FUTURE WORK 47](#_Toc40307539)

[6.1 Conclusions 47](#_Toc40307540)

[6.2 Future work 48](#_Toc40307541)

[6.3 Legal, Social, Ethical and Professional Issues 48](#_Toc40307542)

[6.4 Synoptic Reflections 50](#_Toc40307543)

[ReferenceS 51](#_Toc40307544)

[Appendix Code 53](#_Toc40307545)

[Appendix Gate images 59](#_Toc40307546)

[Appendix Diagrams 60](#_Toc40307547)

[Appendix User documentation 61](#_Toc40307548)

List of Figures

[Figure 1: Computer Science by Bob Reeves 7](#_Toc40312767)

[Figure 2: A Sphero Inc’s littleBits Module 8](#_Toc40312768)

[Figure 3: E-Logic Trainer Kit 9](#_Toc40312769)

[Figure 4: Gantt chart 20](#_Toc40312770)

[Figure 5: GameObject Class 29](#_Toc40312771)

[Figure 6: UpdateGraphics Function of Node class 31](#_Toc40312772)

[Figure 7: Example of a GateCollection Instance 34](#_Toc40312773)

[Figure 8: Some of DLG\_Home’s Interactions and Relationships 36](#_Toc40312774)

[Figure 9: DLG\_TaskManager 41](#_Toc40312775)

[Figure 10: DLG\_CircuitTaskDesigner 42](#_Toc40312776)

[Figure 11: Diagram Representing the Creation of Tasks 42](#_Toc40312777)

List of Tables

[Table 1: Milestones 18](#_Toc40312240)

[Table 2: Tools 21](#_Toc40312241)

[Table 3: Resources 23](#_Toc40312242)

[Table 4: Contingency plans 24](#_Toc40312243)

[Table 5: Coding Standards 28](#_Toc40312244)

[Table 6: Major Functions of Gate Class 32](#_Toc40312245)

[Table 7: Results of User Testing Questionnaire 45](#_Toc40312246)

[Table 8: LSEPIs of Software 50](#_Toc40312247)



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, how they come together to form circuits, truth tables and conversion between these circuits and truth tables. The new piece of software intends to target the issue discussed below and as a result provide a better means of educating students.

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

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. While existing education methods have worked up until this point finding new and more effective methods of education will prove useful.

## Background

### 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.

Most logic gate simulators are free and open source, those that do cost are usually for use in industry. The main platform for these simulators is desktop software, however simulators for other platforms exist including web and mobile.

### Use of Logic Gate Simulators for Education

Logic gate simulators can be used as an education tool to teach students the behaviour of gates. Various simulators are designed for this exact purpose, for instance the desktop application Logisim. Teachers use these tools alongside theory from textbooks to achieve their goal.

Most of these tools make use of the experiential learning process, which Felicia Patrick (2011) defines as ‘Learning through reflection on doing’. Various studies (Kruger, D. et al., 2015) (Ban, Y., Okamura, K., and Kaneko, K., 2017) 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

Research within the realm of using logic gates for education is still ongoing. Example of existing research is the conference paper by Lukovic, V. et al (2016). The paper demonstrates the design of the logic gate simulator Logisim and presents the results of a questionnaire given to students on the effectiveness of their use of the software in an educational setting. This will be elaborated and more references to work will be discussed within the context section of this report.

## 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 by various studies (Roselli, R. and Brophy, S., 2006) (Mutasim, E. et al, 2018) to be effective in engaging students in the learning process. These studies will be further analysed in the context section of the report.

## 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. This literature review will investigate all three and existing research behind them, along with research on the effects of challenge-based learning in an educational setting.

## Challenge Based Learning

In studies conducted by Roselli, R. and Brophy, S. (2006) students were taught either with taxonomy-based or challenge-based approaches to instruction over a three-year period. The results of the study showed students learning using the challenge-based approach scored better on more difficult questions (35% verses, 4%). Interestingly surveys also indicated that students preferred the challenge-based approach, showing that it may be better at encouraging students than standard approaches.

This idea is supported by the results of the study in an article written by Mutasim, E. et al (2018). The results of the studies on students subject to challenge-based learning showed that 86.3% of students reported a positive perception of all items assessing the benefits of the challenge-based learning approach.

## Existing Solutions to Logic Gate Education

### Books or Online Research

Books are one of the main methods of teaching students about logic gates. These, with teachers in classrooms focus on the theory of logic gates to educate. 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, in figure 1. 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. These learning resources fail to make use of the ‘learning by doing’ education approach, discussed above. 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

2.3.2.1 Module kits

There are many companies which sell logic gate modules for education, an example being littleBits, Sphero Inc (2011-2020). Figure 2 shows an example of one of their modules.



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 (Kruger, D. et al., 2015) (Ban, Y., Okamura, K., and Kaneko, K., 2017) to be effective within the classroom.

A paper by Dürre, J et al (2016) 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.

Using module kits, while successful in part at educating A-Level students on logic gates, require theory 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 losing 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.

2.3.2.2 A logic gate model board

An article by Zulkarnain, M. et al (2019) on the creation of an E-Logic Trainer Kit explains the design as well as effectiveness of the prototype within an educational setting. 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

2.3.3.1 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.

2.3.3.2 Example 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, Burch, C (2014). This simulator is used by many schools and universities in classes ranging from GCSE Level to computer architecture courses. A conference paper by Lukovic, V. et al (2016) investigated using Logisim as an educational tool.

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 today’s day and age, this is no longer much of an issue.

Abeer Alsadoon, P.W.C. Prasad & Azam Beg (2017) investigated the effectiveness of software simulators for educating students. The abstract demonstrates similar results were concluded to that of the study conducted on Logisim.

2.3.3.3 Other platforms

Educational logic gate simulator software also reaches platforms other than those for just desktop/laptop computers. An article by Jovanovic, N. et al (2009) on the Web Based Interactive Digital Logic Circuit Simulator, 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 by Tafrikhatin, A. et al (2017) 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.



New Ideas

Introduction

This part of the report is to justify the new design based off the research and 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.

## Justification of Application

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, has been shown as an effective method in terms of education, discussed in the context section. Essentially, without challenges to overcome, students will not 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 will not have to refer to external sources such as text books or questionnaire sheets for the questions, therefore making the learning material all software based resulting in a more streamlined learning process. 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. Use of a software interface as apposed to a hardware one provides an easy method for teachers to construct the challenges for students.

The programming design model used by this new piece of software is the event-driven based model, as opposed to the cyclic based model (discussed chapter 2). 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.

While certain logic gate simulators delve into the transistor, or even lower level, of technicality in terms of gates they provide, the target audience for the new application is A-Level students. As a result, only the basic gates taught at A-Level were developed for the application.

## 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.

## 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. The waterfall method also fitted well because only one developer worked on the project.

The exact type of the waterfall design methodology used was SSADM Weaver, P., et al (1998).

### 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 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 perform user testing and be asked their opinions on the effectiveness of the software has on achieving its 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 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.

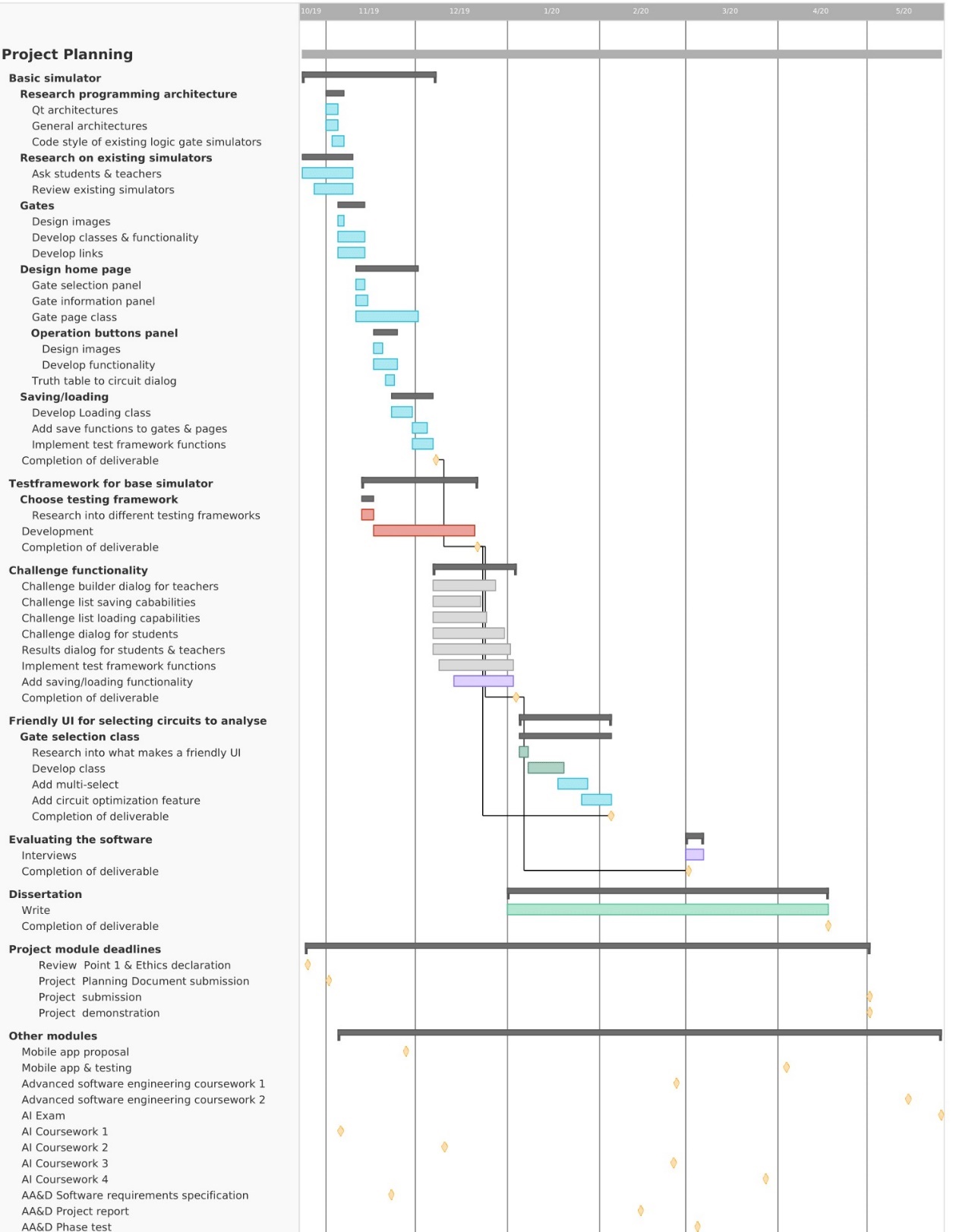


Figure 4: Gantt chart

### Tools and Resources

3.3.5.1 Why C++

Developing this project for the windows OS seems the most appropriate as an initial target platform release. This is due to its popularity, and ability to handle such an application with ease. As a result of this choice, a programming language for this platform needed to be picked.

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. |
| Google forms. Questionnaire software. | Of the various questionnaire software looked at google forms was the simplest to setup and contained all features necessary for evaluation of user responses. This was utilized in the collection of ideas for the new software, as well as during evaluation of the software. |
| 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 table 4. 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.

## Design Method

The design method followed was the Structured systems analysis and design method. Weaver, P., et al (1998).

## Coding Standards

Due to the software requiring many classes, a hierarchical structure of folders was created to separate the files in an organized way. The folders included.

* A GameObjects folder, holding all gate-like objects or the classes from which they inherit.
* A Dialogs folder, holding source and header files of dialogs and widgets.
* A Tests folder, which had unit tests within.
* A Resources folder, holding all images and icons used by the software.

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 5 below.

Table 5: 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

Following an object-oriented design style, all gate models have individual classes which all inherit from the Gate base class. This class inherits from the DraggableGameObject and GameObject 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. The figure below demonstrates this class, as seen these functions include positioning and dimension variables, drawing of a set image, click hitbox checking, and setting of a disabled user access property.



Figure 5: GameObject Class

The DraggableGameObject, also shown in figure 5, 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 gates except the TextLabel gate have one or more nodes. Instances of the node class (appendix code - 3) are used for signal input/output 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 below.



Figure 6: UpdateGraphics Function of Node class

The base Gate class of which the code is shown in the appendix (Code - 2), has a vector of pointers to nodes. Inheriting gate classes of the base Gate class fill this vector with their nodes setting their positions upon construction. This vector allows for many node operations, called from the gate, to be performed from a base gate class instance of abstraction as opposed to the actual gate class level of abstraction. This similar with many other gate functions, and is important, since the parent GateField or GateCollection classes in which gates are stored, store their gates from a gate base class level of abstraction as a vector of pointer to gates, this is discussed in more detail below. Table x describes some of the major functions within the gate class.

Table 6: Major Functions of Gate Class

|  |  |
| --- | --- |
| Function | Description |
| UpdateGraphics | Called when the parent 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. An example of this is in appendix code – 5. |
| SaveData | This function saves the details of the gate, such as type, node links and position 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 gates parent GateField, when the GateField click mode is that of delete and the GateField wishes to check if the click was on the gate. |
| Clone | Acts as a copy constructor for a gate and returns a pointer to a gate class containing the information of the copied gate. All gate classes override this function. Most of these 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 gate’s 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.

Gates also have the hierarchy-based functions SetParent and GetParent, which act as a setter and a getter for the gates parent GateField. Functions are called both down the hierarchy, from GateField to Gate, as well as up, Gate to GateField. A diagram representing this interaction is shown within the GateField description section of this document.

Having all gate classes inherit from the base Gate class means that a GateField or GateCollection instance has access to the different gate classes functionality with a pointer to their Gate base class, given that the functionality of the Gate class is overridden from the inherited gate class.

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 consists of all 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 it as a whole or the gates within it during runtime. These buttons include;

* A delete button, which removes the collection but keeps the gates inside it by releasing them to the GateField in which they are currently situated.
* 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 a file for loading or transferring to another computer.



Figure 7: Example of a GateCollection Instance

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. Appendix code - 4 shows gates being added through a DLG\_Home instance from the Widget\_AllGates class (A gate selection widget class). The functionality of many similar operations, such as the pressing of buttons is handled through DLG\_Home. Figure 8 demonstrates this as well as some interactions and the relationships between the DLG\_GateInfo, DLG\_Home, GateField, Gate classes and Gate selection classes.



Figure 8: Some of DLG\_Home’s Interactions and Relationships

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. When displaying the gates, the GateField class can perform various operations such as moving gates, panning and zooming. 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 of one of these operations 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 whether 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, to avoid crashes.

The Gatefield class makes use of a timer thread. This is for updating any timer gates it may contain. Development of this proved difficult, with a custom destructor for the timer needing to be created to avoid crashes.

The QT library came in much use for this class. GateFields make use of a lasso Qt tool 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, 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.

To save the information of gates various properties and functions 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 to determine the class of a pointer to a Gate, resulting in efficiency. 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. Once loaded the application uses around 40mb – 90mb. Adding a reasonable number of gates, moving them around causes this to go up to 200mb. Once these gates are removed and their GateFields are too (since they hold the undo operation memory). The memory levels return to that of 100mb, showing there is slight memory leakage.

### GUI

4.3.3.1 Images

Images were designed using paint.net. The application 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 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. 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. 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 code - 7 demonstrates the code for this.

The dialog class DLG\_TaskManager shown in figure 9, 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, for easier coding. 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.



Figure 9: DLG\_TaskManager

The dialog classes DLG\_CircuitTaskDesigner, in figure 10, and DLG\_TruthTableTaskDesigner (software images appendix - 3) are both used to create the tasks for students to perform. They both produce a task file which is saved to memory.



Figure 10: DLG\_CircuitTaskDesigner

Before either of these are run, an instance of the dialog DLG\_CircuitTaskDesignerSetup is created. This dialog gives the user the option to enter in the number of inputs and outputs for the task they are creating. These numbers are given to the corresponding task creation dialog for the user to set up their task. As an example, DLG\_TruthTableTaskDesigner takes these input and output numbers when instantiated and creates and places the number of input and output gates required on the GateField that the user uses to design the circuit for the task. The flow diagram in figure 11 demonstrates how this dialog is used for both task setup types to reduce code duplication.



Figure 11: Diagram Representing the Creation of Tasks

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.

## Unit Testing

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.

By the end of development, all unit tests were passing. This ensured the features above were working correctly. After any changes to the system sections that were tested by unit tests were tested, ensuring that these changes did not break the overall functionality.



RESULTS

Introduction

### Once development of the project was completed, an analysis of it needed to be conducted to measure its success. This section of the report is to demonstrate how the new software with its challenge mode compares against other teaching methods in terms of effectiveness. It will go over the anonymous results of user testing performed on the application collected through a questionnaire to do this.

## User Testing Questionnaire and Results

The people selected for the user testing and questionnaire were 12 computer science and software engineering university students. These students were the perfect choice for evaluating the specific application since they had undergone education of logic gates previously, and therefore had previous experience in learning about them and how they work to an A-Level standard.

During the user testing the students were asked to build some basic circuits, perform saving and loading operations, create two different tasks for the challenge mode, and then act out these tasks. The questionnaire then focused on the student’s overall experience of this, as well as how effective they thought the new software would be at educating A-Level students on the functions of logic gates.

The first part of the questionnaire consisted of statements about the software, in which testers would use a 0 – 10 rating scale to show how much they agreed or disagreed with each statement (10 for definitely agree, 0 for definitely disagree). Testers were also provided with an optional reason section for each question to explain the reason of their choice. Listed are these questions.

1. The use of the simulator software would have aided me as an A-Level student in learning about the functions of logic gates.
2. I found the challenge mode of the simulator a beneficial tool for engaging me in education of logic gates, truth tables and conversion between them.
3. I see benefit over use of the challenge mode part of the software compared to books or other software in engaging students in logic gate education.
4. The user guide for the software was clear and helpful.
5. Overall (excluding circuit design), the user interface of software was straightforward to use.
6. Creating circuits using the software was a quick and straightforward process.
7. Saving and loading features were clear to me.
8. I can see benefit in terms of education with the software’s ability to create and save gate collections (custom gates).

The results of this part of the questionnaire were summed and averaged, this is shown in the table below.

Table 7: Results of User Testing Questionnaire

|  |  |
| --- | --- |
| Question | Result out of 10 |
| 1 | 8.3 |
| 2 | 8.7 |
| 3 | 8.7 |
| 4 | 8.5 |
| 5 | 7.7 |
| 6 | 6.5 |
| 7 | 9.3 |
| 8 | 7.8 |

Overall results from the first section of the questionnaire showed the students considered the tool effective for the education of logic gates. The 8.1 score for question 1, 8.7 score for question 2, and 8.7 score for question 3 demonstrates the success of the software in achieving its goal. This shows that the tool is in fact worthwhile for educating students, compared up against previously used tools by the testers. Scores relating to overall user friendliness also scored high, demonstrating that the tool is pleasant to use. The lowest rating question was question 6, relating to ease of construction of circuits. Most of the reasons for lower ratings were because of the links feature for gates not being straightforward enough. This showed the need for software, and potentially user guide improvements in this field. Other reasons are covered by the missing features and improvements discussed below.

The second part of the questionnaire consisted of an open-ended question relating to what testers felt could be improved with the user guide and software. Most of the answers to this question involved improvements on the linking feature between gates, such as how they are linked or need for a coloured link system. Others included:

* The ability to directly copy paste selections as opposed to having to save them to memory before duplicating them.
* Adding text descriptions of buttons when the mouse hovers over them to reduce dependence on the user guide.
* Making dragging of gates easier since sometimes they are accidentally dropped when dragged.
* When selecting a gate to add, making it hover under the cursor to the user to position it instead of the gate spawning in the middle of the GateField which may overlap another gate.



CONCLUSIONS / FUTURE WORK

## Conclusions

From the results of the questionnaire, it is obvious that the system has various miss falls and areas for improvement. However, the high scores on the challenge functionality questions within the questionnaire demonstrates the success of this new piece of software in acting as an effective alternative method for educating students on the functionalities of logic gates, how they come together to form circuits, truth tables, and converting between these circuits and truth tables.

A lot of development time was spent on the simulator and additional features than the challenge mode, which was the highlight of the creation of the project. Allocating more time towards development of the challenge functionality during planning could have been more beneficial in demonstrating and analysing the effectiveness of the teaching method. For an example, the creation of the gate information dialog while serving a purpose, was not unique in any way and did not aid in engaging students to learn through challenges.

Overall, the development process of the project went well. The project could have benefitted from the addition of more tests, and possibly test-driven development. Having more confidence in the functionality of functions and classes because of these additional tests could have aided in the locating of code defects and as a result sped up the development process more.

Performing additional user testing earlier on in the project timeline would have allowed for time to make amendments and improvements suggested from testers.

## Future work

As mentioned previously, the simulator at the time of this report is in a basic state. Below are some improvements that could be made to it in future work.

* Ability to rotate gates.
* Add in more gates, such as transistors or FPGA’s.
* Ability to copy sections or circuit.
* Coloured links depending on if active or not.
* Better linking methods, or larger nodes to make the dragging and dropping easier.
* Text description of a button when mouse hovers over it.
* Port the application over for other platforms than just windows.
* Adding additional features such as positional data to the gate information widget.

The challenge mode itself could be extended as well, to improve upon the successful new teaching method.

* The challenge mode could be extended so that tasks can be uploaded online to a shared database of tasks.
* A batch task creator mode could be added, to speed up the process of creating multiple tasks.
* Additional tasks types could be added, to expand the area of education that the tool provides. For example, Boolean algebra to circuit translations and vice versa tasks.

## Legal, Social, Ethical and Professional Issues

At the start of the user testing session each student was given a participation consent form which they were required to sign. The form outlined what the tester would be subject to, the fact that they will remain anonymous and provide contact details so they can redact provided information later if they felt the need to do so.

After the completion of this report the new software will be released as open source and free to download. Due to this, it will be licenced with the MIT license. This licence puts very little restriction on reuse of the code and software, meaning people will be able to extend the software if they wish.

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 has been added to the release documents.

Table 8: LSEPIs of Software

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 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 |

Finally, to keep the work in the direction of public interest, the creation of the code and report followed along the principles of the BCS code of conduct.

## Synoptic Reflections

Working on this project was a large challenge to overcome. C++ programming is something that interests me very much and is the field of work I wish to enter once I finish university. The skills that I have developed from working on this project will greatly benefit me when it comes to the workplace. As a result of this project I have learnt about developing a large-scale piece of software and the planning, design, analysis, development, and testing skills required for such a project.

ReferenceS

Abeer Alsadoon, P.W.C. Prasad & Azam Beg, 2017. Using software simulators to enhance the learning of digital logic design for the information technology students. *European Journal of Engineering Education* [Online]. 42 (5) (08-Jun-2016), 533-546. DOI: [10.1080/03043797.2016.1196344](https://doi.org/10.1080/03043797.2016.1196344) [Accessed 4 February 2020].

Ban, Y., Okamura, K., and Kaneko, K., 2017. Effectiveness of Experiential Learning for Keeping Knowledge Retention in IoT Security Education. In: *6th IIAI International Congress on Advanced Applied Informatics (IIAI-AAI)*, 2017, Hamamatsu, Japan, pp. 699-704, DOI: 10.1109/IIAI-AAI.2017.206.

Burch C, 2014. *Logisim* [Online]. Available at: http://www.cburch.com/logisim/ [Accessed 4 February 2020].

Dürre, J. et al., 2016. Teaching Digital Logic Circuit Design via Experiment-Based Learning - Print your own Logic Circuit. *World Multi-Conference on Systemics, Cybernetics and Informatics,* WMSCI 2016. Orlando, Florida, USA, Vol: 1, pp. 242.

Felicia Patrick, 2011. *Handbook of Research on Improving Learning and Motivation*. Ireland: Waterford Institute of Technology. p. 1003. ISBN 1609604962.

Jovanovic, N. et al., 2009. WEB Based Interactive Digital Logic Circuit Simulator. *JOURNAL of the Technical University of Gabrovo* [Online]. 37, 58-62. Available via:https://www.researchgate.net/publication/318361261\_WEB\_Based\_Interactive\_Digital\_Logic\_Circuit\_Simulator [Accessed 4th February 2020].

Kruger, D. et al., 2015. Assessing the Effectiveness of Experiential Learning in a Student-Run Free Clinic. *Pedagogy in Health Promotion: The Scholarship of Teaching and Learning* [Online]*.* 1 (2) (May 2015), 91-94. DOI: 10.1177/2373379915575530

Lukovic, V. Et al., 2016. Using Logisim Educational Software in Learning Digital Circuits Design. *3rd International Conference on Electrical, Electronic and Computing Engineering IcETRAN 2016*. Zlatibor, Serbia, June 13-16, 2016.

Mutasim, E. et al, 2018. The Effectiveness of Problem-based Learning in Acquisition of Knowledge, Soft Skills During Basic and Preclinical Sciences: Medical Students’ Points of View. *Acta Inform Med* [Online]. 26 (2) 119-124. DOI: 10.5455/aim.2018.26.119-124. [Accessed 1 November].

Roselli, R. and Brophy, S., 2006. Effectiveness of Challenge-Based Instruction in Biomechanics. *Journal of Engineering Education* [Online]. 95 (4). DOI: 10.1002/j.2168-9830.2006.tb00906.x [Accessed 1 November].

Sphero Inc, 2011-2020. *Introduction to Logic* [Online]. Available at: https://classroom.littlebits.com/ [Accessed 4 February 2020]. Tafrikhatin, A. et al, 2017. Development of A Mobile Learning for Digital Logic Gates in Vocational Schools. *International Conference on Technology and Vocational Teachers (ICTVT 2017)* [Online]*.* Universitas negeri Yogyakarta, 2017. 10.2991/ictvt-17.2017.28. Available at: https://www.researchgate.net/publication/320468266\_Development\_of\_A\_Mobile\_Learning\_for\_Digital\_Logic\_Gates\_in\_Vocational\_Schools [Accessed 4th February 2020].

Weaver, P., et al, 1998*. Practical SSADM Version 4+ A Complete Tutorial Guide.* 2nd Edition. s.l.: FT Pitman.

Zulkarnain, M. et al, 2019. E-Logic Trainer Kit : Development of an Electronic Educational Simulator and Quiz Kit for Logic Gate Combinational Circuit by using Arduino as Application. I*nternational Journal of Online and Biomedical Engineering* [Online]. 15 (14) 2019. Available via: https://online-journals.org/index.php/i-joe/article/view/11410 [Accessed 4th February 2020].

Appendix Code

Code Directory Listing

Logic\_Gate\_Simulator\allgates.h

Logic\_Gate\_Simulator\circuitoptimizer.cpp

Logic\_Gate\_Simulator\circuitoptimizer.h

Logic\_Gate\_Simulator\clickmode.h

Logic\_Gate\_Simulator\filelocations.h

Logic\_Gate\_Simulator\gatefield.cpp

Logic\_Gate\_Simulator\gatefield.h

Logic\_Gate\_Simulator\gatereader.cpp

Logic\_Gate\_Simulator\gatereader.h

Logic\_Gate\_Simulator\Logic-Gate-Simulator.pro

Logic\_Gate\_Simulator\Logic-Gate-Simulator.pro.user

Logic\_Gate\_Simulator\main.cpp

Logic\_Gate\_Simulator\mainwindow.cpp

Logic\_Gate\_Simulator\mainwindow.h

Logic\_Gate\_Simulator\resouces.qrc

Logic\_Gate\_Simulator\truthtables.cpp

Logic\_Gate\_Simulator\truthtables.h

Logic\_Gate\_Simulator\Dialogs\dlg\_circuittaskdesigner.cpp

Logic\_Gate\_Simulator\Dialogs\dlg\_circuittaskdesigner.h

Logic\_Gate\_Simulator\Dialogs\dlg\_circuittaskdesigner.ui

Logic\_Gate\_Simulator\Dialogs\dlg\_circuittaskdesignersetup.cpp

Logic\_Gate\_Simulator\Dialogs\dlg\_circuittaskdesignersetup.h

Logic\_Gate\_Simulator\Dialogs\dlg\_circuittaskdesignersetup.ui

Logic\_Gate\_Simulator\Dialogs\dlg\_gateinfo.cpp

Logic\_Gate\_Simulator\Dialogs\dlg\_gateinfo.h

Logic\_Gate\_Simulator\Dialogs\dlg\_gateinfo.ui

Logic\_Gate\_Simulator\Dialogs\dlg\_home.cpp

Logic\_Gate\_Simulator\Dialogs\dlg\_home.h

Logic\_Gate\_Simulator\Dialogs\dlg\_home.ui

Logic\_Gate\_Simulator\Dialogs\dlg\_load.cpp

Logic\_Gate\_Simulator\Dialogs\dlg\_load.h

Logic\_Gate\_Simulator\Dialogs\dlg\_load.ui

Logic\_Gate\_Simulator\Dialogs\dlg\_message.cpp

Logic\_Gate\_Simulator\Dialogs\dlg\_message.h

Logic\_Gate\_Simulator\Dialogs\dlg\_message.ui

Logic\_Gate\_Simulator\Dialogs\dlg\_savegatecollection.cpp

Logic\_Gate\_Simulator\Dialogs\dlg\_savegatecollection.h

Logic\_Gate\_Simulator\Dialogs\dlg\_savegatecollection.ui

Logic\_Gate\_Simulator\Dialogs\dlg\_task.cpp

Logic\_Gate\_Simulator\Dialogs\dlg\_task.h

Logic\_Gate\_Simulator\Dialogs\dlg\_taskmanager.cpp

Logic\_Gate\_Simulator\Dialogs\dlg\_taskmanager.h

Logic\_Gate\_Simulator\Dialogs\dlg\_taskmanager.ui

Logic\_Gate\_Simulator\Dialogs\dlg\_textedit.cpp

Logic\_Gate\_Simulator\Dialogs\dlg\_textedit.h

Logic\_Gate\_Simulator\Dialogs\dlg\_textedit.ui

Logic\_Gate\_Simulator\Dialogs\dlg\_truthtabletaskdesigner.cpp

Logic\_Gate\_Simulator\Dialogs\dlg\_truthtabletaskdesigner.h

Logic\_Gate\_Simulator\Dialogs\movingwidget.cpp

Logic\_Gate\_Simulator\Dialogs\movingwidget.h

Logic\_Gate\_Simulator\Dialogs\simpleslider.cpp

Logic\_Gate\_Simulator\Dialogs\simpleslider.h

Logic\_Gate\_Simulator\Dialogs\widget\_advanced.cpp

Logic\_Gate\_Simulator\Dialogs\widget\_advanced.h

Logic\_Gate\_Simulator\Dialogs\widget\_advanced.ui

Logic\_Gate\_Simulator\Dialogs\widget\_allgates.cpp

Logic\_Gate\_Simulator\Dialogs\widget\_allgates.h

Logic\_Gate\_Simulator\Dialogs\widget\_allgates.ui

Logic\_Gate\_Simulator\Dialogs\widget\_customgates.cpp

Logic\_Gate\_Simulator\Dialogs\widget\_customgates.h

Logic\_Gate\_Simulator\Dialogs\widget\_customgates.ui

Logic\_Gate\_Simulator\Dialogs\widget\_inputgates.cpp

Logic\_Gate\_Simulator\Dialogs\widget\_inputgates.h

Logic\_Gate\_Simulator\Dialogs\widget\_inputgates.ui

Logic\_Gate\_Simulator\Dialogs\widget\_standard.cpp

Logic\_Gate\_Simulator\Dialogs\widget\_standard.h

Logic\_Gate\_Simulator\Dialogs\widget\_standard.ui

Logic\_Gate\_Simulator\Dialogs\widget\_text.cpp

Logic\_Gate\_Simulator\Dialogs\widget\_text.h

Logic\_Gate\_Simulator\Dialogs\widget\_text.ui

Logic\_Gate\_Simulator\Dialogs\widget\_truthtable.cpp

Logic\_Gate\_Simulator\Dialogs\widget\_truthtable.h

Logic\_Gate\_Simulator\Dialogs\widget\_truthtable.ui

Logic\_Gate\_Simulator\GameObjects\gameobject.cpp

Logic\_Gate\_Simulator\GameObjects\gameobject.h

Logic\_Gate\_Simulator\GameObjects\gate.cpp

Logic\_Gate\_Simulator\GameObjects\gate.h

Logic\_Gate\_Simulator\GameObjects\gateand.cpp

Logic\_Gate\_Simulator\GameObjects\gateand.h

Logic\_Gate\_Simulator\GameObjects\gatecollection.cpp

Logic\_Gate\_Simulator\GameObjects\gatecollection.h

Logic\_Gate\_Simulator\GameObjects\gateconstantactive.cpp

Logic\_Gate\_Simulator\GameObjects\GateConstantActive.h

Logic\_Gate\_Simulator\GameObjects\GateConstantInactive.cpp

Logic\_Gate\_Simulator\GameObjects\GateConstantInactive.h

Logic\_Gate\_Simulator\GameObjects\gateeor.cpp

Logic\_Gate\_Simulator\GameObjects\gateeor.h

Logic\_Gate\_Simulator\GameObjects\gatefpga.cpp

Logic\_Gate\_Simulator\GameObjects\gatefpga.h

Logic\_Gate\_Simulator\GameObjects\gatenand.cpp

Logic\_Gate\_Simulator\GameObjects\gatenand.h

Logic\_Gate\_Simulator\GameObjects\gatenor.cpp

Logic\_Gate\_Simulator\GameObjects\gatenor.h

Logic\_Gate\_Simulator\GameObjects\gatenot.cpp

Logic\_Gate\_Simulator\GameObjects\gatenot.h

Logic\_Gate\_Simulator\GameObjects\gatenumberoutput.cpp

Logic\_Gate\_Simulator\GameObjects\gatenumberoutput.h

Logic\_Gate\_Simulator\GameObjects\gateor.cpp

Logic\_Gate\_Simulator\GameObjects\gateor.h

Logic\_Gate\_Simulator\GameObjects\GateReciever.cpp

Logic\_Gate\_Simulator\GameObjects\GateReciever.h

Logic\_Gate\_Simulator\GameObjects\GateSingleOutput.cpp

Logic\_Gate\_Simulator\GameObjects\GateSingleOutput.h

Logic\_Gate\_Simulator\GameObjects\GateTimer.cpp

Logic\_Gate\_Simulator\GameObjects\GateTimer.h

Logic\_Gate\_Simulator\GameObjects\GateToggle.cpp

Logic\_Gate\_Simulator\GameObjects\GateToggle.h

Logic\_Gate\_Simulator\GameObjects\gatetriand.cpp

Logic\_Gate\_Simulator\GameObjects\gatetriand.h

Logic\_Gate\_Simulator\GameObjects\gatetrieor.cpp

Logic\_Gate\_Simulator\GameObjects\gatetrieor.h

Logic\_Gate\_Simulator\GameObjects\gatetrior.cpp

Logic\_Gate\_Simulator\GameObjects\gatetrior.h

Logic\_Gate\_Simulator\GameObjects\gatetypes.h

Logic\_Gate\_Simulator\GameObjects\gatexor.cpp

Logic\_Gate\_Simulator\GameObjects\gatexor.h

Logic\_Gate\_Simulator\GameObjects\textlabel.cpp

Logic\_Gate\_Simulator\GameObjects\textlabel.h

Logic\_Gate\_Simulator\Resources\Button Icons

Logic\_Gate\_Simulator\Resources\Gates

Logic\_Gate\_Simulator\Resources\Ununsed

Logic\_Gate\_Simulator\Resources\Button Icons\gate-collection-delete.png

Logic\_Gate\_Simulator\Resources\Button Icons\gate-collection-delete-all.png

Logic\_Gate\_Simulator\Resources\Button Icons\gate-collection-move-gates.png

Logic\_Gate\_Simulator\Resources\Button Icons\gate-collection-nand.png

Logic\_Gate\_Simulator\Resources\Button Icons\gate-collection-optimize.png

Logic\_Gate\_Simulator\Resources\Button Icons\gate-collection-save.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-application.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-bin.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-drag.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-file-down.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-file-explorer.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-file-up.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-and.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-collection.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-input.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-input-off.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-input-on.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-not.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-or.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-output.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-selection.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-timer.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-tri-and.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-gate-tri-or.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-link-basic.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-link-delete.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-new-page.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-pan.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-redo.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-tasks.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-tool.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-undo.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-zoom-in.png

Logic\_Gate\_Simulator\Resources\Button Icons\icon-zoom-out.png

Logic\_Gate\_Simulator\Resources\Gates\Old Gates

Logic\_Gate\_Simulator\Resources\Gates\gate-and.png

Logic\_Gate\_Simulator\Resources\Gates\gate-eor.png

Logic\_Gate\_Simulator\Resources\Gates\gate-nand.png

Logic\_Gate\_Simulator\Resources\Gates\gate-neor.png

Logic\_Gate\_Simulator\Resources\Gates\gate-nor.png

Logic\_Gate\_Simulator\Resources\Gates\gate-not.png

Logic\_Gate\_Simulator\Resources\Gates\gate-number-output.png

Logic\_Gate\_Simulator\Resources\Gates\gate-or.png

Logic\_Gate\_Simulator\Resources\Gates\gate-tri-and.png

Logic\_Gate\_Simulator\Resources\Gates\gate-tri-eor.png

Logic\_Gate\_Simulator\Resources\Gates\gate-tri-or.png

Logic\_Gate\_Simulator\Resources\Gates\Old Gates\gate-and - Copy.png

Logic\_Gate\_Simulator\Resources\Gates\Old Gates\gate-not - Copy.png

Logic\_Gate\_Simulator\Resources\Gates\Old Gates\gate-or - Copy.png

Logic\_Gate\_Simulator\Resources\Ununsed\box.png

Logic\_Gate\_Simulator\Resources\Ununsed\circle.png

Logic\_Gate\_Simulator\Resources\Ununsed\icon-mag.png

Logic\_Gate\_Simulator\Tests\Tests\.gitignore

Logic\_Gate\_Simulator\Tests\Tests\Tests.cpp

Logic\_Gate\_Simulator\Tests\Tests\Tests.h

Logic\_Gate\_Simulator\Tests\Tests\Tests.pro

Logic\_Gate\_Simulator\Tests\Tests\Tests.pro.user

Building the application

The IDE QT Creator is required to rebuild the software. To build the executable open the qt project file Logic-Gate-Simulator using the IDE and then click build.

Below are the code files (or sections of the code files) referenced within this report.

Referenced Files

1 – GameObject and DragableGameObject classes (Headers/GameObjects/gameobject.h)

2 – The gate class (Headers/GameObjects/gate.h)

3 – The node class (Headers/GameObjects/gate.h)

4 – Widget\_AllGates (Sources/Dialogs/widget\_allgates.cpp)



5 – And gate example override of UpdateOutput function (Source/GameObjects/GateAnd.cpp)

6 – Drawing of nodes and links between nodes (Source/GameObjects/gate.cpp)

7 - Task design verification. (Sources/Dialogs/dlg\_task.cpp)

Appendix Software Images

1 – Example of a GateCollection object

2 – Example instance of DLG\_CircuitTaskDesigner



3 – DLG\_TruthTableTaskDesigner

4 – DLG\_TaskManager



Appendix Diagrams

1 - Process flow diagram of challenge task creation.



2 – Diagram of DLG\_Home interactions with GateFeild, DLG\_GateInfo, Widget\_AllGates and other selection widgets.



Appendix User documentation

When starting the application, the user is presented with the loading dialog.



Once finished loading the main window appears.



To the left of the main window at the top is the gate selection area. This contains a dropdown triggerd by pressing the “All” button. The dropdown displays various pages from which gates can be chosen to add to currently open gatefield (large page in the middle of the main window) named “New Tab” in the example. The all gates selection page has a scrollbar which the user can scroll through to located the gate they wish to add. Other gate selection pages include, standard, custom, input and advanced.



The custom gates page has some additional features. Here the user can asses previously saved custom circuits to use. A button for deleting these is on the top right of this page, and a button for creating a new custom circuit is on the top left.



Below the gate selection page, is the gate information page. Here information of a currently seleceted gate in a gatefield is displayed. Various options are displayed depending on the gate selected. These include, enabling/disabling the gate, changing signal values deleting the gate, and for timer gates, setting their frequency.



When using gates on the gatefield, there are some advanced features to considor. Using the highlight feature, by dragging the mouse over a selection of gates places these gates into a gatecollection object, seen below.



Gate-collections have four buttons. The first switches the ‘drag mode’ of the collection represented by a change in background colour, meaning the user can switch between dragging the entire gate collection as a single gate, or moving the individual gates within it. As seen below moving these gates causes the boundaries of the gate selection to update. The following button is to save the gatecollection object to disk, which adds it the custom circuits page shown above. Finally, are the two delete buttons, one which removes the entire gatecollection and its contents, and the other which removes only the gatecollection itself, adding its contained gates back onto the gatefield.



Another gate type which requires further explanation is the label gate. Selecting the grey box by the side of the gate opens a dialog from which the text of the label gate can be changed. This entire dialog is also custom designed and includes a custom scrollbar for changing the size of the label’s font.

On the right-hand side of the main window are a series of buttons used for manipulation of the gates. Their descriptions are shown below in the image.



These operations are also accessible under to tools tab at the top of the main window.



Buttons also exist at the top of the main window; their descriptions are shown below.



A new challenge task can be created using the File -> New Task menu options. The user is presented with two task creation options.



Once one is clicked, the user is navigated to the dialog shown below. Here the number of inputs and outputs is set for the task the user has chosen to create.



Once the user presses the “Ok” button, if they chose to create a truth table task, they are navigated to the dialog shown below. Here the user creates a circuit that is compared with the truth table during the challenge. The inputs and outputs are preplaced by the application.



If the user chooses a circuit creation task to create, once they have entered the number of inputs and outputs of the required circuit, they are navigated to the dialog below. Here the user enters the values of the truth table that will be compared against the circuit during the challenge. The user does this by clicking on the right-hand values turning them either to a 1 or 0.



Once tasks creation is completed, the task is saved to a tasks folder located next to the executable and the user is navigated back to the main window. Pressing on the tasks button, detailed above, opens the task manager shown in the image below. This loads all tasks from the task folder and displays them for the user to do.



Buttons representing completed tasks are coloured green, shown below.



When performing a task, a new dialog opens. Below is what a circuit task looks like. The truth table on the right-hand side is what should match the circuit.



Below is what a truth table task looks like. The user must select the values on the right-hand side of the truth table so that it represents that of the ‘Or’ circuit shown.



Pressing the submit button at the bottom right of the tasks dialog submits the task for review, and the user is navigated back to the task manager dialog. If correct the colour of the challenge button of the task just completed will change green. If incorrect, a dialog message box will appear notifying the user.