Graphical Enigma Simulator

Majed Monem
Final Year Project
BSc (Hons) Applied Computing
University of Dundee, 2015
Supervisor: Prof. J. Arnott

Abstract - *Enigma machines were used in the main during World War II by the German military. It was a device which scrambled plain text into ciphered text. This project demonstrates both the encryption and decryption processes carried out by the Enigma machine in a graphical simulation. The simulation demonstrates the movement of the rotors within the machine, which presents a 3D graphical visualisation of the process of encrypting plain text into cipher text and the process of decrypting cipher text into plain text.*

1 Introduction

With the outbreak of wireless communication in the early 1900s, there was a necessity for secure communication, particularly for military use. With this came the invention of the Enigma machine in 1918, invented by a German engineer, Arthur Scherbius. Later the Enigma machine patented in 1919. In the 1920s, early models were used commercially, and later adopted by Nazi Germany before and during World War II. The Enigma machine was an electro-mechanical device which scrambled plain text messages into ciphered text using a letter substitution system. This enabled the military forces to communicate with their allies using coded messages.

In this project a Graphical Enigma Simulator was developed, which demonstrated the inner workings of the rotors during the process of encryption (plain text to ciphered text) as well as the process of decryption (ciphered text to plain text). The aim of the project was to provide a greater detail of the processes carried out by an Enigma machine in a 3-dimensional graphical format, specifically the rotors. The simulator, developed in C++ programming language, also allows users to encrypt and decrypt their own text while viewing the current path within a rotor.

There are several sections in this report which cover various aspects of the project. In the background section, historical information about the Enigma machine is presented, which enhances the understanding of Enigma machines. In addition, information about the rotor wiring and related work is discussed. In the specification section the requirements of the project will be illustrated, as well as project management related information. Following on shall be the design section which will outline the design of the simulator. The implementation and testing section will provide an overview of the technologies considered for use before development was underway. Testing methods and results shall also be outlined. In the evaluation section, results from evaluation shall be discussed. The final product shall be described after that and finally the summary and conclusion drawn from this project, including recommendations for future work shall be discussed.

2 Background

2.1 History

The Enigma machine was invented by a German engineer in 1918, Arthur Scherbius and later adopted by Nazi Germany before and during World War II. The Enigma machine was a device used by the German army to communicate with their allies using encrypted messages. The Enigma machine consisted of a keyboard of 26 letters in the pattern of the normal German typewriter, but with no keys for numeric or punctuation characters. Behind the keyboard was a lamp board made up of 26 small circular windows, each bearing a letter in the same pattern as the keyboard, which could light up one at a time. Behind the lamp board was the scrambler unit, consisting of a fixed wheel at each end and a central space for three rotation wheels. Messages were limited to a maximum of 250 letters to avoid the inner mechanism returning to the same position, because the sequence would repeat itself after 17,576 (26x26x26) notches. Had the messages not been limited, then British code-breakers may have been able to crack the encrypted messages. Thus potentially the number of cipher text alphabets was vast and this led German military authorities to believe that this cipher system was unbreakable. [1] The first Enigma machine was heavy and bulky, which did not appeal as such to the German military for use. It was not until Enigma D was introduced, by then a reflector was added to increase the complexity level, that the military considered the Enigma machine.

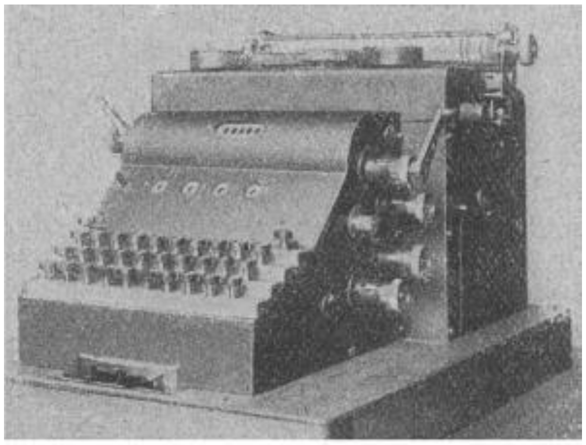


Figure 1 – Enigma A (Crypto Museum, 2008)

Various versions of Enigma machines were developed, each with varying rotors. In 1926, a commercial version of the Enigma machine was purchased by the German Navy and adapted for military use. A special Enigma machine was developed by Chiffriermaschinen-AG (a company formed by Scherbius), which contained rotors that consisted of the same contact alignment as the rotors in Enigma D, but with teeth, multiple notches and were advanced cog wheels instead of pawls and ratchets. This model lead to the Enigma G. The Enigma G had different rotors with a 'zig-zag' pin placement and the counter on its right. Its rotors, which also had multiple notches, were moved by a system of gears. In 1932 the Wehrmacht (German armed forces) revised the commercial Enigma D version and added the plugboard at the front of the machine. This version, known as Enigma I, became known as the Wehrmacht Enigma and was introduced on a large scale in the army and public authorities. Initially, this Enigma machine came with three rotors, however, from 1939 onwards they were equipped with five rotors. The Wehrmacht model was later adopted by the German Navy, with its securer plugboard and the extended set of rotors of eight. [2]

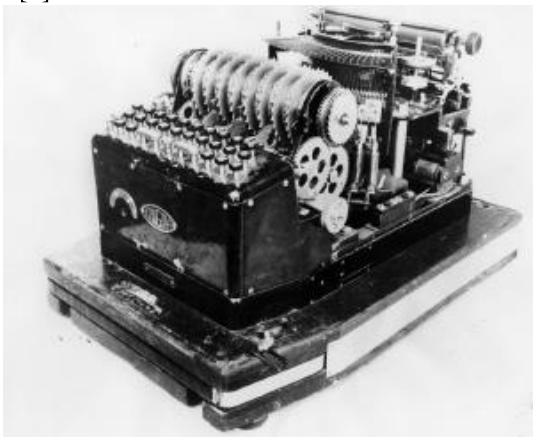


Figure 1-1 – The Wehrmacht Model (Crypto Museum, 2009)

2.2 Breaking the code

The Germans believed that the messages being sent to their allies were not breakable. However, the code breakers based at Bletchley Park cracked the secret messages being broadcasted, which played a crucial role in the defeat of Germany. The Polish were the first people to come close to cracking the Enigma code. Marian Rejewski, Henryk Zygalski and Jerzy Rozicki were three mathematicians who successfully cracked the Enigma. They also developed an electro-mechanical machine, called the Bomba, to speed up the code breaking processing. [2] With the invasion of Poland looming, the Poles shared their information with the British, who in turn established the Government Code and Cipher School at Bletchley Park. However, it was only in 1941 when their work began to pay off meaningfully, when they were able to gather evidence of the planned invasion of Greece. [3]

2.3 Bletchley Park

Bletchley Park is the home of the Government Code and Cipher School (GC&CS) based in Milton Keynes, UK, where the Enigma machine was initially broken. This location was chosen as it is 45 miles north of London with direct railway connections to London, as well as connections to Cambridge and Oxford, which allowed scientists and army personnel to travel secretly. Alan Turing developed the Bombe, not to been confused by the Bomba which it was in fact based on. He developed a more universal method based on cribs (pieces of predicted plain text), due to the Polish method of exploiting the German vulnerability of the double-enciphered message indicator, no longer being valid. The value of this codebreaking machinery was recognised by the then British Prime Minister, Winston Churchill, who introduced a new level of secrecy to surpass all other levels, known as ULTRA. Three additional rotors used exclusively by the Navy and not shared with any other parts of the army. In 1941 Turing discovered the procedure of the additional wheels. [4]

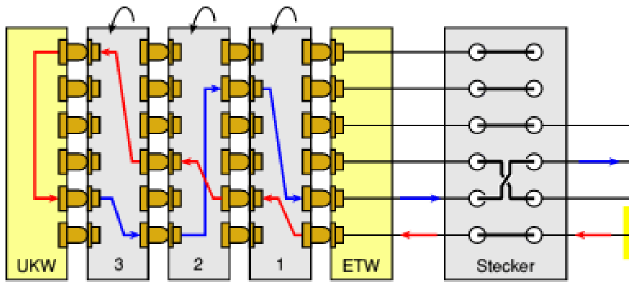


Figure 1-2 Circuit Diagram (Crypto Museum, 2009)

2.4 Rotor Wiring

Each rotor had 26 positions, one for each letter of the alphabet. After a key had been pressed the rotor would rotate so the next letter is visible. If, for example, the letter 'A' was active on the key press, then on the next key press the letter 'B' would be active on that rotor. Once a full cycle had been completed the next rotor would rotate one step. Once the first rotor reaches the letter 'Z' it would ensure that the letter 'A' is active for the next key press, but also if the next rotor was on the letter 'Q' then the letter 'R' would be active during the next key press. The same would occur for the third rotor, once the second rotor has completed a cycle. This results in 17,576 (26 x 26 x 26) possibilities. In addition to the rotors there was a reflector (Umkehrwalze) added on the end and a plugboard (Steckerbrett or Stecker) was introduced to the first Wehrmacht version of the Enigma machine. The reflector redirected the current back to the rotors by a different route. With the exception of the beta and gamma reflectors, each letter was paired with one another. For example, 'E' and 'Q' were paired together on reflector B, so that when the rotor passed the current to 'E', 'Q' would be passed back to the rotor. The rotor wiring was dependant on the rotor itself, each had different internal wiring. The plugboard added an extra layer of complexity to the Enigma machine. It was situated at the front of the machine and enabled the key press to map to a different letter on the rotor.

Figure 1-3 – Rotor V and VI
(Ilord [5], 2010)

2.5 Related Work

In terms of related work, the author was not aware of any simulators which represented the inner workings of an Enigma machine in a 3D graphical representation. However, various simulators which encrypt and decrypt text are available, widely on the internet. One simulator in particular caught the attention of the author. [6] It demonstrates the current path when a key is pressed through three rotors, in a simple, 2-dimensional format. This simulator provided a deeper understanding on how a simulator for this project could be developed in terms of the logic for the encryption and decryption processes.

3 Specification

3.1 Project Management

3.1.1 Project Specification

The purpose of this project was to develop a Graphical Enigma Simulator which would demonstrate the process of encryption and decryption of an Enigma machine. One of the particular aims is to visually demonstrate the principle of poly-alphabetic substitution in operation in the rotors.

3.1.2 Project Plan

A plan of how much time to allocate to each aspect of the project was essential to facilitate good time management and to provide an understanding of how the project was progressing. The aspects of the project were as follows:

- Background research
- Requirement analysis
- Project design
- Ethical approval
- Code implementation
- Testing
- Interim report
- Evaluation
- Final report and portfolio

A Gantt chart, shown in Appendix G, was created with task lists and timescales to ensure that the author could the complete tasks without delay.

3.1.3 Methodology

Throughout the development of the project, an iterative development cycle was utilised. Waterfall and agile methodologies were considered but neither were suitable for this project because the waterfall approach did not allow revision to the project and as for the agile approach, not enough information on the structure of the simulator was known at the stage of commencing the project. In addition, users of the simulator were not available for requirements to be gathered and furthermore formal documentation was required. The iterative approach allowed revision of aspects of the project throughout.

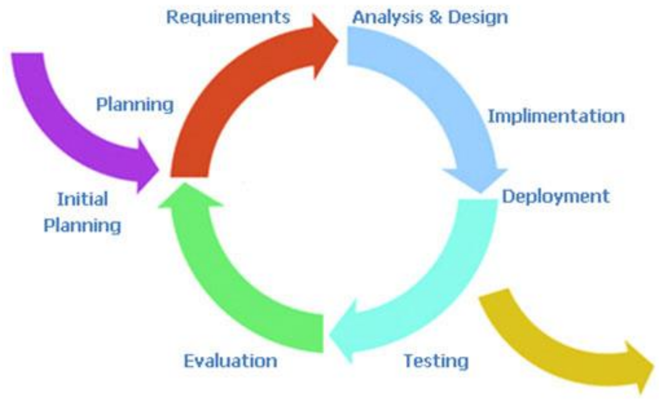


Figure 2 – Iterative model (Voltreach [7], 2011)

3.1.4 Source Control

To reduce the risk of file corruption and deletion from the author’s local machine, GitHub was used. GitHub is a repository hosting service, which also offers revision control. This allowed regular backups to be made and record project progression.

3.1.5 Ethical Approval

User participation was required to perform evaluation, therefore ethical approval from the School of Computing Ethics Committee was required. The ethics form and approval letter can be found in Appendix A and Appendix E, respectively.

3.1.6 Meeting with Supervisor

Weekly meetings were arranged with the project supervisor. During meetings, the tasks that had been carried out the previous week were discussed. The supervisor provided suggestions and improvements to the author about the project to ensure that the project could be accomplished. Insight was also provided on difficulties encountered by the author and suggestions for next steps to be taken for the following week.

3.2 Requirement Elicitation

The first step in the development lifecycle was to gather requirements. The main source for gathering requirements were from meetings with the supervisor.

3.3 Functional Requirements

A set of functional requirements were established, detailing each of the functions the simulator should facilitate.

3.3.1 Main Menu

The simulator shall have a main menu.

3.3.2 Main Menu - options

The main menu shall contain three options: Encrypt, Decrypt and Exit.

3.3.3 Encrypt Option

The simulator shall allow the encryption process to be simulated after selecting Encrypt from the Main Menu.

3.3.4 Decryption Option

The simulator shall allow the decryption process to be simulated after selecting Decrypt from the Main Menu.

3.3.5 Exit Option

This option shall allow the simulator to close once after selecting Exit from the Main Menu.

3.3.6 Encryption

The simulator shall scramble plain text into cipher text.

3.3.7 One Rotor - Encryption

The simulator shall demonstrate the operation of encryption in one rotor.

3.3.8 Three Rotors Encryption

The simulator may demonstrate the operation of encryption in three rotors.

3.3.9 Decryption

The simulator shall unscramble cipher text into plain text.

3.3.10 One Rotor - Decryption

The simulator shall demonstrate the operation of decryption in one rotor.

3.3.11 Three Rotors - Decryption

The simulator may demonstrate the operation of decryption in three rotors.

3.3.12 Visual Representation

The simulator shall visually demonstrate the principle of poly-alphabetic substitution in operation in the scrambling unit of an Enigma machine.

3.3.13 Animation

The simulation shall be demonstrated using animation.

3.3.14 Attack Method

The simulator may include an attack method, which could become a game, where the user would guess the encrypted plain text.

3.4 Non-Functional Requirements

A set of non-functional requirements were established, detailing requirements of the implementation.

3.4.1 Graphical User Interface

The interface shall be presented in a graphical format.

3.4.2 Operating System - Windows

The simulator shall be compatible on Windows Operating Systems.

3.4.3 Operating System – Mac/Linux

The simulator may be compatible on Mac/Linux Operating Systems.

3.4.4 Development

The simulator should be developed using C++ and Visual Studio IDE.

4 Design

4.1 Design Prototypes

Several user interface prototypes were developed during the initial stages of development. The main focus was to ensure a high level of usability could be achieved in terms of ease of use for the users. As part of the iterative development lifecycle, throughout development, prototypes were revised.

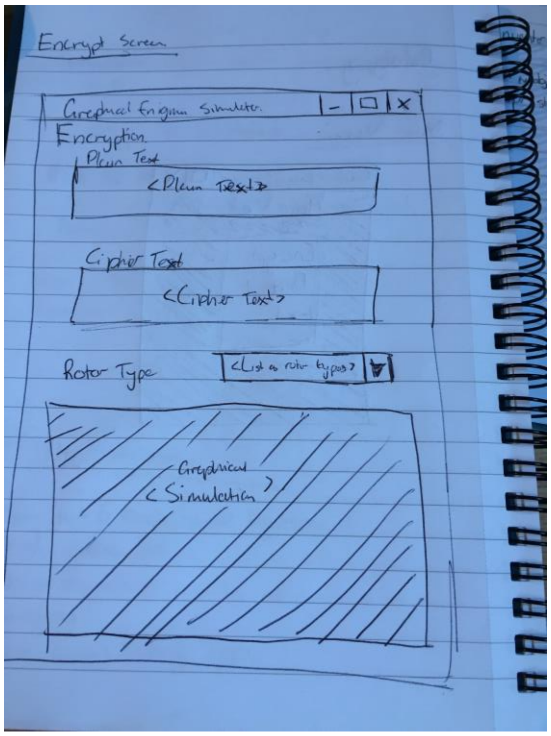


Figure 3 – Hand drawn prototype

After the hand drawn prototypes were complete, high fidelity prototypes were designed for the first iteration.

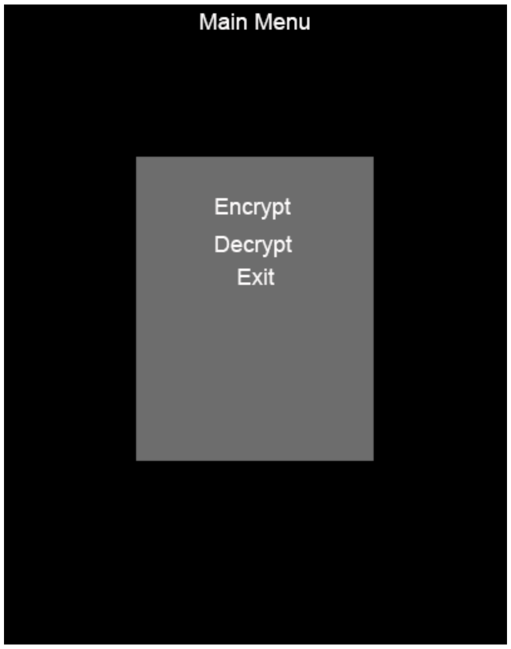


Figure 3.1 – Main Menu prototype design (Portrait orientation)

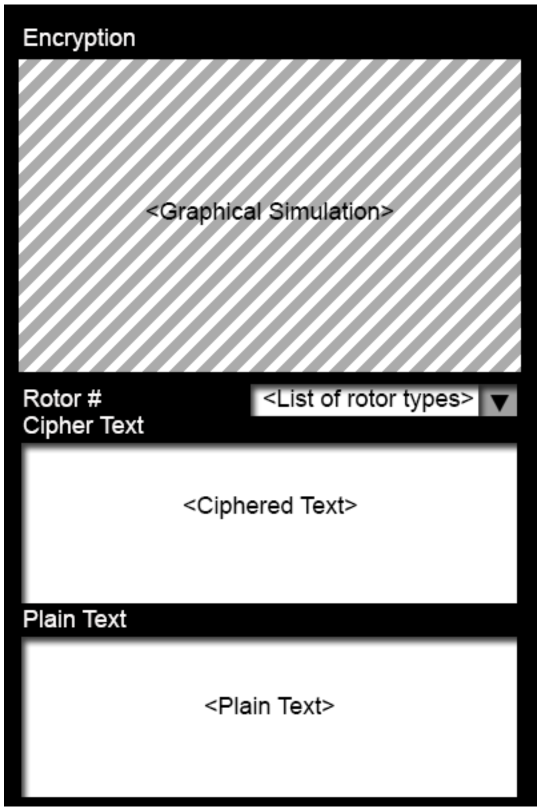


Figure 3.2 – Encryption screen prototype design (Portrait orientation)

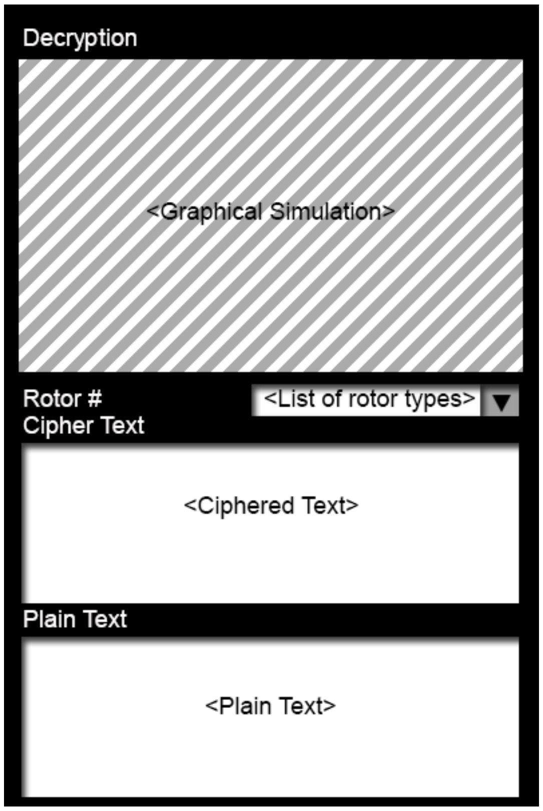


Figure 3.3 – Decryption screen prototype design (Portrait orientation)

The major change during this iteration was the repositioning of the graphical simulation area. The author decided that placing the graphical simulation area towards the top of the simulator would be more beneficial because of aesthetic appeal and ease of use. Landscape versions of the prototypes were also designed. These can be found in Appendix H.

4.2 Final Design

During development the screen felt cluttered with portrait orientation while the simulation was running. At that stage, a design decision was made to set the orientation on the simulation screens (Figure 3.5 & Figure 3.6) to landscape. The main menu was not affected by this.

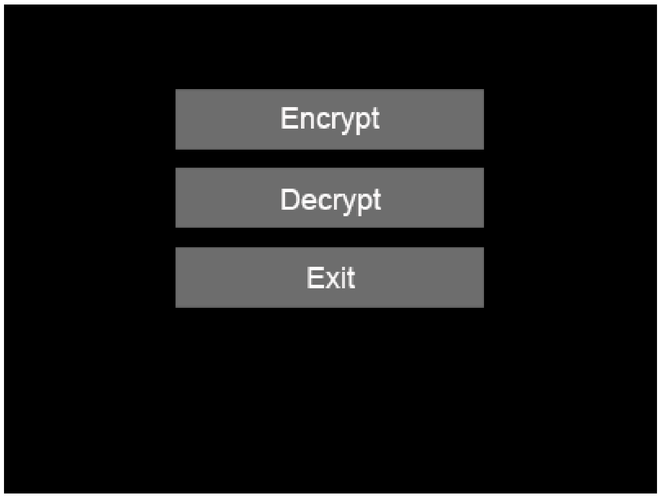


Figure 3.4 – Main Menu (Final prototype)

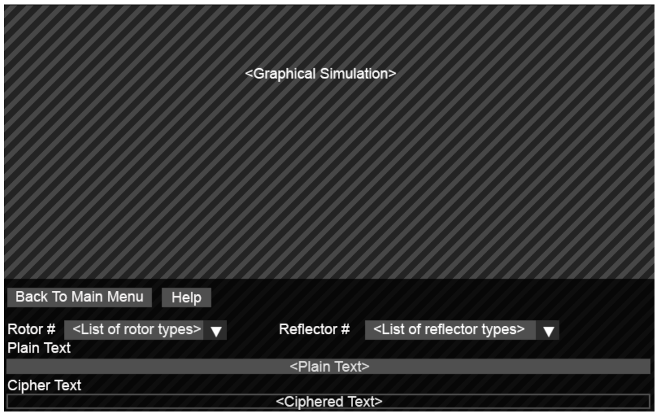


Figure 3.5 – Encryption Screen (Final Prototype)

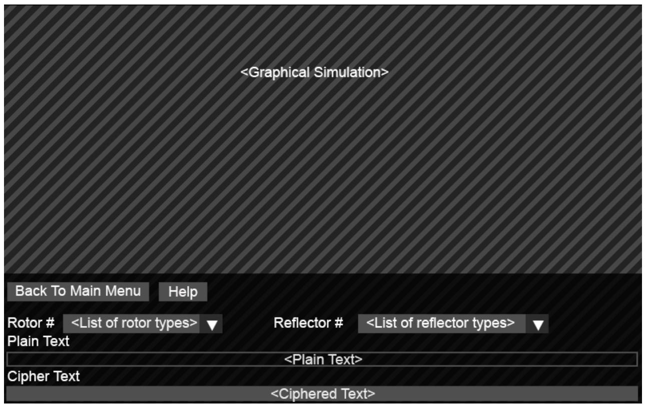


Figure 3.6 – Decryption Screen (Final Prototype)

During the final iteration of design, a number of features to the user interface were added. One of the features added was the selection of reflector types as well as rotor types. A help button was also added to give additional guidance to the user. Initially the idea was to have a menu bar, but it was then decided a button which would take the user back to the main menu should be implemented instead, since the menu bar would not contain enough options to be relevant. In order to allow the users to understand the processes of encryption and decryption, the processes are done live. For example, during the encrypting process, when a user enters a letter into the plain text field, the encrypted letter would be processed and outputted instantaneously to the cipher text field as well as the rotor animating. Further details can be found in Appendix H.

4.3 Rotor Design

While the purpose of this project was to demonstrate a detailed perspective of the encryption and decryption processes, each part of the rotor was modelled using Blender. [8] Each component was modelled as closely as possible to the real life counterparts, where feasible.

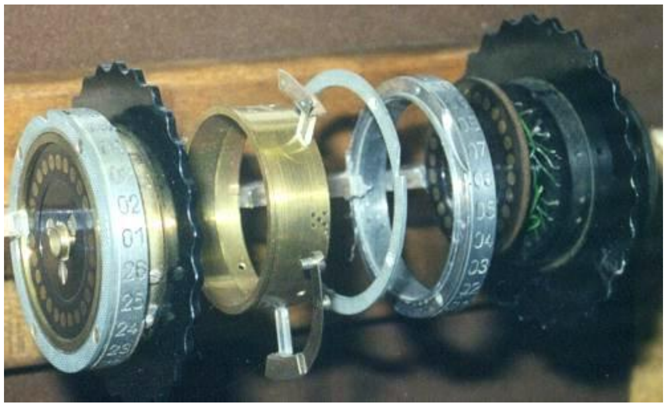


Figure 4 – Exploded View of Rotor
(Picture by Jerry Proc [9])

Once modelled using Blender, the model was exported as a Wavefront file format (.obj). Colours and textures were not included in the Blender models.

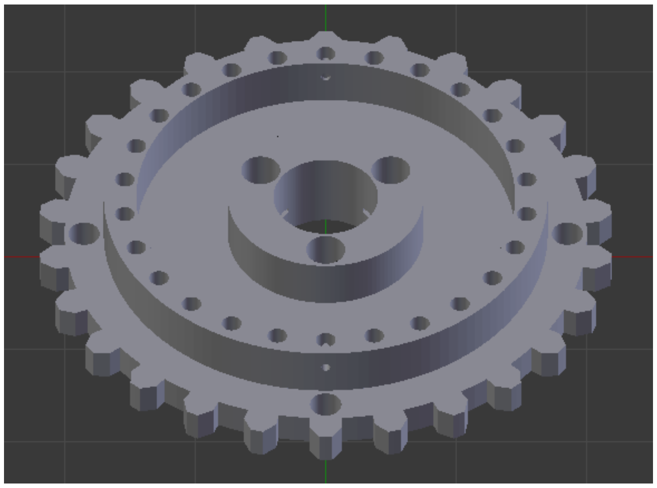


Figure 4.1 – Ratchet Wheel Model

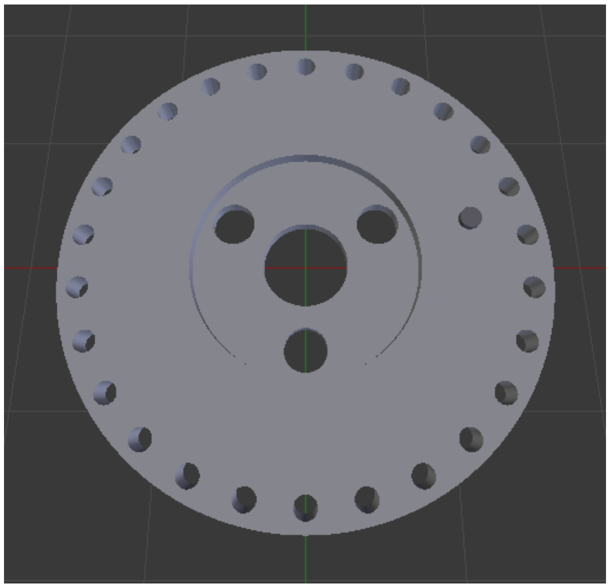


Figure 4.2 – Contact Wheel Model

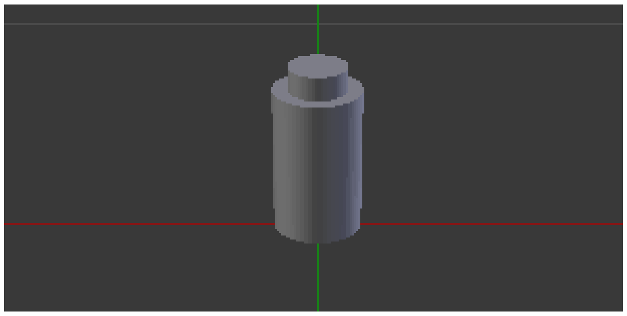


Figure 4.3 – Short Pin Model

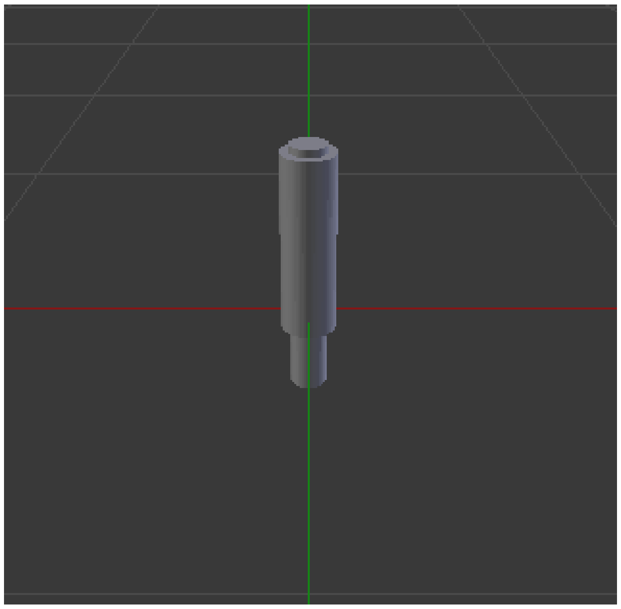


Figure 4.4 – Long Pin Model

The scale of these models were not an important factor because they could be modified during coding, therefore the author decided to model them without too much importance, given this factor. It was important, however, that all the components were proportional to each other while modelling, in order save time during coding to align them correctly. With this factor taken into account, the diameter of the holes in each component was consistent throughout. Figure 4.1, which shows the ratchet wheel model, was constructed from a gear. Creating a gear from scratch would take an unnecessary amount of time therefore a Blender extension was acquired. [10] Figures 4.3 and 4.4, show the pins which the simulator will use to represent the path of the letters during the encryption and decryption processes. These were to be imported only once, where then the simulator could calculate the positioning of each individual pin. 26 pins would be drawn, one to represent each letter of the alphabet. The positions were to be calculated using Equation (1) for the *x* coordinate and Equation (2) for the *z* coordinate. The *y* coordinate was irrelevant.

$$x = (1.51 \div 4) \times \cos((2\pi \div 26) \times i) \tag{1}$$

$$z = (1.51 \div 4) \times \sin((2\pi \div 26) \times i) \tag{2}$$

Where *i* is iterator variable within a loop and *x* and *z* are the coordinates. *1.51* represents the radius of the circle of which the outer holes were plotted onto the contact wheel (Figure 4.2). The divisor of *4* was chosen to ensure a small output was given to scale to the scene appropriately. *26* was the number of points required, one for each letter of the alphabet. The coordinates of the pin objects were to be treated as *x* and *z* coordinates rather than *x* and *y*. This was due to the simplicity of applying transformations to the objects using OpenGL transformation functions.

5 Implementation and Testing

5.1 Considerations

A variety of technologies were researched before coding commenced. The key areas researched were programming languages, APIs (application programming interfaces) used to render the graphics and graphical user interface libraries.

5.1.1 APIs

A choice of programming language and graphical user interface library revolved around on the choice of this. The two choices researched were Direct3D and modern OpenGL.

Direct3D
Direct3D is a graphics API which can be used to create 2D and 3D graphics but it is proprietary software. Also it is not cross-platform compatible and is only available on Microsoft Windows operation systems.

OpenGL
OpenGL is a graphics API which can be used for rendering 2D and 3D vector graphics. Modern OpenGL can run across multiple platforms and is an open standard rather than proprietary software.

5.1.2 Graphical User Interface

Various graphical user interface libraries were researched. This would be the component allowing the users to interact with the simulator.

CEGUI [11]
Crazy Eddie's GUI system. This is a free library providing windowing and widgets for graphics API and engines where such functionality is not natively available or is severely lacking. The library is written in C++, it is object orientated and is primarily targeted at game developers. It is licensed under the MIT license.

LibRocket [12]
LibRocket is a C++ interface middleware package designed for game applications. At its core it is based on the popular HTML and CSS specifications. It implements the Model-View-Controller (MVC) design pattern. It is licensed under the MIT license.

LibUFO [13]
LibUFO is a C++ core library for developer graphical user interfaces. It is mainly used as an OpenGL GUI toolkit. It is based upon an abstract layer which must be implemented by a native backend. It is licensed under the GNU LGPL version 2.1 license.

AntTweakBar [14]
AntTweakBar is a small and easy to use C/C++ library that allows programmers to quickly add a light and intuitive graphical user interface into graphical applications based on graphical APIs such as OpenGL and DirectX to interactively tweak parameters on-screen. It is design to be fast, clean and intuitive while minimizing the programmers work. It is released under the zlib/libpng license.

ImGui [15]
Immediate Mode Graphical User Interface (ImGui) is a bloat-free graphical user interface library for C++. It outputs vertex buffers that you can render in your 3D-pipline enabled application such as OpenGL. It favours simplicity and productivity rather than certain features normally found in more high-level libraries. A main feature

of ImGui is that it is designed to be efficient and fast. It is licensed under the MIT license.

Qt [16]
Qt is a cross-platform application and UI framework for developers using C++, a CSS and Javascript like language. It is licensed under a commercial and open source license.

FLTK [17]
Pronounced “full tick”, FLTK is a cross-platform C++ GUI toolkit. It provides modern GUI functionality without the bloat and supports 3D graphics via OpenGL and its built-in GLUT emulation. It is licensed under the GNU Library General Public License.

5.1.3 Programming Language

An important factor which impacted the selection of language was cross-platform compatibility. C#, C++ and Java were taken into consideration given the author’s previous experience with these.

C#
C# is an object-oriented programming language. It allows development for applications that run on the .NET framework. The syntax is simple to learn so it was an easy choice for consideration. Also the author had vast experience using this language and felt comfortable.

C++
An extension of the C language, C++ is an object-oriented programming language. It encapsulates both high and low level language features. It is seen by many as the best language for creating large scale applications.

Java
Designed to have the look and feel of C++, Java is simpler and easier to learn. It is object-orientated and robust. In addition, it is platform independent.

5.2 Technologies Used

Decisions about which technology to use were finalised early on during development, since they were based on the requirements. Visual Studio was chosen as the development environment because it can be easily accessed by the author.

5.2.1 Blender [8]

The individual components of the rotor had to be modelled in 3D. Blender was the obvious choice since it was free to use, compared to most popular modelling tools which required paid licensing. A major disadvantage of this, however, was the steep learning curve and the fact that the author had not worked with this software before. With this factor taken into account the components were not modelled to complete precision but the important details, which would be required to enable the user to view the rotor, were successfully modelled. A feature of Blender is that it can export models to a variety of file formats. The file format used to export the models was a Wavefront (.obj) format. This was because the vertex positions of the object could easily be parsed into any program, from the outputted file.

5.2.2 OpenGL

The main reason behind choosing OpenGL was because the Graphics module was well underway, therefore it made sense to choose OpenGL instead of having to learn a brand new language. In addition, OpenGL is an open standard not proprietary software and it is cross-platform, which means it is not restricted to only being run on Microsoft Windows operating system. This would also meet one of the requirements.

5.2.3 GLSL [18]

Once the objects were imported into our OpenGL environment, lighting was required to provide a more realistic effect as well as being aesthetically pleasing. GLSL (OpenGL Shading Language) was used to achieve this effect. It was also taught during the Graphics module, alongside OpenGL. It is used so that code can be run on the GPU. It is made up of four shaders; vertex shaders, fragment shaders, geometry shaders and tessellation shaders. GLSL is a similar to C programming language with C++ mixed in.

Vertex Shaders
The vertex shader processes each vertex separately so they run once per vertex, passed to the graphics processor. This is required for all OpenGL programs and must have a shader to pass to, usually the fragment shader. The purpose of this shader is to define vertex positions and other vertex attributes such as position, colour and texture coordinates.

Fragment Shaders
This is actually the last part of the shader pipeline. It processes the individual fragments generated by OpenGL’s rasterizer and its main purpose is to compute the colour and depth of pixel fragments. It must also have a shader bound to it, normally the vertex shader.

Tessellation Shaders
This shader is optional. It receives its inputs from the vertex shader. It processes patches, a type of geometric primitive specifically for tessellation shaders. Its purpose is to tessellate (spilt mesh into smaller geometric primitives such as triangles) the mesh patches.

Geometry Shaders
This shader is also optional. Its inputs are either from the vertex or tessellation shader. It processes each geometric primitive and defines geometric primitives. It can modify the type and number of geometric primitives by emitting altered or new primitives, or discarding them.

For this project only the vertex and fragment shaders were required because greater detail was not required while rendering the graphics, therefore these would be sufficient enough. Also, geometry and tessellation shaders were not taught during the Graphics module.

5.2.4 C++

Given that a user interface was a necessity C# was an obvious choice for consideration due to the fact that Visual Studio allows drag and drop of user interface components using this programming language. One major advantage of this would be that it would save time on researching and implementing a graphical user interface library, as it would not be required. However, modern OpenGL implementation did not seem feasible given that experience implementing OpenGL was with C++.

5.2.5 ImGui

A graphical user interface library which was compatible with modern OpenGL was essential. It would also be beneficial if the library supported GLFW [19] windowing system. ImGui, Immediate Mode Graphical User Interface, was the chosen library as it was compatible with both. It is maintained on GitHub, meaning updates for this library are made regularly. Integrating a graphical user interface library with OpenGL was found to be a more difficult task than initially thought, however, ImGui was the only library which the author was able to integrate successfully with OpenGL and GLFW. While a lack of support for this library exists on the internet, the library provides code which is easy to understand and documentation, which made it the correct decision to use this as a graphical user interface library.

5.2.6 SOIL [20]

In order to load images, SOIL, Simple OpenGL Image Loader, an image loading library was used. Its primary function enables OpenGL to load images and use them as textures. The author used this to load in an image of the rotor details to present to the users. The library provided didn't seem to work with the latest version of OpenGL, therefore the version used in this project may differ from versions downloadable. However, the modifications made were not implemented by the author, but rather the Graphics module coordinator.

5.3 Classes

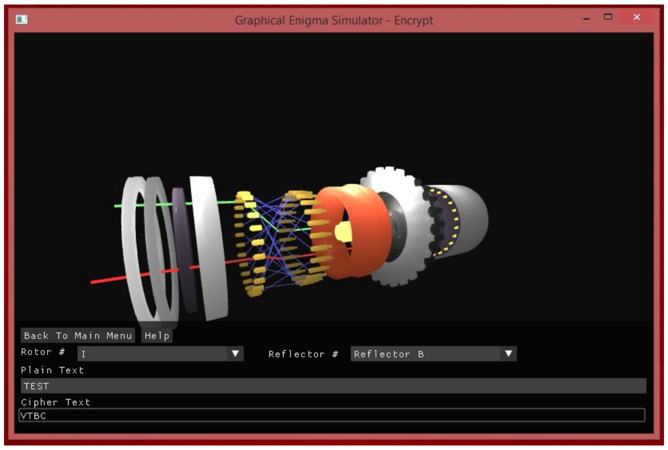
Main – This class contains the main entry point of the program. It initializes the ImGui library and carries out the main operations required for rendering the OpenGL graphics such as drawing the components to the screen.

Wrapper_glfw – This class was required to handle most of the user interface operations and create the GLFW context to enable OpenGL to be rendered. It also links to the Enigma class and partially handles the encryption and decryption logic.

Object_ldr – This class has been derived from the Graphics module, where it was used to load in Wavefront (.obj) file types. It processes Wavefront files and parses each line of code and stores the vertex positions in a 3 dimensional vector. This made use of the GLM mathematics library available for OpenGL.

ImGui – This class is the graphical user interface library. All of the user interface widgets are linked to this.

Enigma – Although this class was not essential, it assisted in keeping the logical flow of the program in order, rather than having masses amount of code in one class. This class was initially created with the implementation of three rotors in mind to meet the requirements.



Rotor # I
Reflector # Reflector B
Plain Text
TEST
Cipher Text
VTBC

Figure 5 – Final Outcome

5.4 Testing

Various methods of testing were carried out. Black box testing was first, followed by white box testing and finally user testing. From testing it can be concluded whether the simulator works or not.

5.4.1 Black Box Testing

Black box testing involves testing the inputs and outputs without being concerned about the process of how the output is achieved. Using this method, the actual output was then compared with the expected output the author had assumed. With this type of testing it was difficult to view the process of the outputs. This was used, primarily, to test that the encryption and decryption functions were able to successfully encrypt and decrypt the input text provided. Simple words were provided as input, the author then done the encryption and decryption by hand and compared results. None of the tests failed. A specific criteria which was tested, was that entering the same letter consecutively 26th times should result in the 27th encrypted or decrypted character being the same as the first character. The Enigma machine did not allow letters to be encrypted into itself so this would test that the letter, provided as input, would never encrypt to itself. The same was done for decryption. Each rotor setting was tested together with the maximum available combinations of reflector settings. The results can be seen in Appendix B.

5.4.2 White Box Testing

Sometimes known as glass box or clear box testing, white box testing tests the reliability of a system. It allows developers to understand the interaction between the input and output, providing a more detailed view of the process. The author achieved this by making use of the console output provided by Visual Studio. While the expected output and the actual output were still tested, in addition the letters which were translated in between the process were printed to the console. Each rotor setting was tested together with the maximum available combinations of reflector settings, same as with black box testing. However, expanding on that, the author also recorded the process of achieving the outputs. For the encryption test, the following data was recorded:

- Current key
- Index of key press
- Key mapped to rotor
- Rotor key mapped to reflector
- Index of key passed back to rotor
- Output key

For the decryption test, the same data as above was recorded, but in addition the index value of initial key is passed to the reflector was recorded. See Appendix T for further information.

5.4.3 User Testing

Before evaluation, how well the simulator works, was carried out, the author had to test whether it functioned in the hands of other users. This proved to be successful, as minor bugs were reported immediately.

Copy and Paste bug
Before the user tested the simulator this bug was already known to the author. Due to the fact that the simulation was occurring live, pasting data into the textbox did not encrypt or decrypt the given text. Although this did not cause an error of any sort, it was still incorrect for it to behave in this manner. This was resolved after the evaluation process however, after feedback was provided to the author.

Backspace bug
When the user held the backspace key for a prolonged period of time, the simulator would crash. The root of this problem was resolved relatively quickly, however. But in solving the problem another bug arose in the text which was being outputted. On pressing the backspace key each character in the rotor string, the index value was incremented by a value of 1. Due to the sensitivity of the backspace callback, it would occur only once. This resulted in the rotor string not being aligned correctly. This bug was resolved after evaluation had been undertaken due to feedback. In addition, the author did not take into consideration to cater for erasing letters other than the last one. As a result of this, when erasing letters other than the final one, the last character of the opposite text field is erased. In order to solve this problem the index of which the cursor was positioned in the text field was required. However, ImGui did not cater for this so the author reported this issue on the GitHub repository for assistance from the creator of ImGui, unsuccessfully however. The author was able to successfully resolve this bug but in doing so, the processes of encryption and decryption had to be recoded.

Current paths disappear
If the user was to enter a long array of letters or even letters towards the end of the alphabet, the current paths would disappear. This was obviously a major concern to the author as this was one of the main features of the simulator. This issue was resolved swiftly as the author found the root of the problem. The problem was that, after each key press, a variable was being used to store the number of letters entered. This variable is then used to calculate the index of the current path to illuminate. However to ensure the variable stayed within the limit of 26 a conditional statement was required to handle this. This conditional statement contained the problem.

Main Menu Resized
If a user was to maximize the main menu window and then enter the Encrypt or Decrypt functions, that window would be resized to the specified size by the simulator. If the user was then to maximize and the restore down that window, it would resize to the size of the main menu. The root of this problem was not found, however the user could easily resize the window themselves using the window handles, without any loss of experience using the simulator.

Key on hold
If a user was to hold a key down, specifically the letters, then the simulator would crash. The author found that this

was due to the fact that on each key press, the simulator would attempt to encrypt/decrypt the input, but when a key was held down, input was entered at such a rate, the simulator simply could not cope. Therefore it was decided that the simulator should not encrypt/decrypt when a key is held but only encrypt/decrypt once, on the initial key press.

Another bug was one that rarely occurred. If the user was to type rapidly and repeatedly and then keep on holding the backspace key, repeating these series of events would cause the text fields and sometimes the rotor key to become out of sync and on the rare occasion, caused the simulator to crash.

Overall the testing process provided valuable feedback to the author to enhance the functionality of the simulator to increase robustness and reliability.

6 Evaluation

The participants are the expert users, it is their product and they know their requirements better than anyone, therefore it was important to find participants to evaluate the simulator. The main factor being assessed was the usability of the simulator, in terms of how easy is it to use, the learning curve required and the intuitiveness of the user interface.

6.1 Usability

6.1.1 System Usability Scale

The System Usability Scale (SUS) [21] provides a quick, reliable tool for measuring usability. It consists of a ten item questionnaire with five response options for the respondents in the form of a Likert Scale. Created in 1986 by John Brooke, it allows evaluation of a wide variety of products and services, including hardware, software, mobile devices, websites and applications.

This provided the author with valuable feedback with a small sample size of participants while providing reliable results. However, SUS is not a diagnostic tool, it classifies the usability of applications. With this in mind the author was able to measure the ease of use of the simulator. Along with ease of use, the learning curve of using the simulator and how well functions of the simulator were integrated were also measured among others.

Participants were required to complete a short demographic questionnaire before commencing evaluation and then the SUS questionnaire, once they had completed evaluation. The participants spent approximately twenty minutes using the simulator, after they were given a short introduction about the project and what an Enigma machine was.

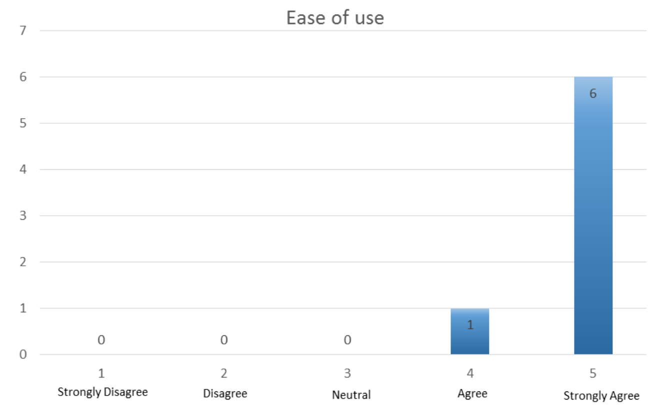


Figure 6 – Ease of Use

The response from the participants regarding the ease of use, Figure 6, was overwhelming. All of them agreed that it was easy to use. This was further demonstrated while observing the participants, where little or no input from the author was required. These results also reflected the simplicity of the interface.

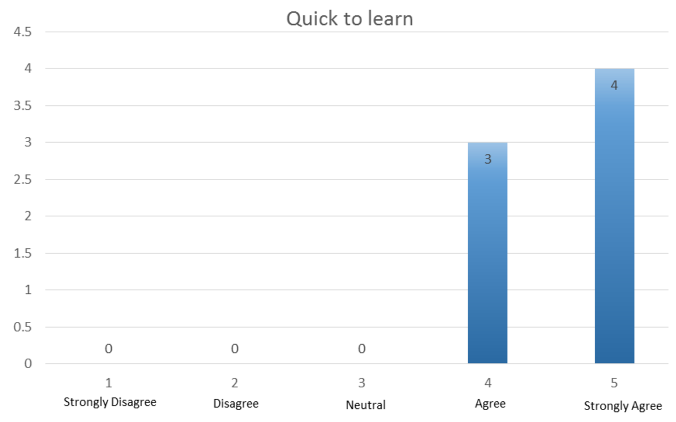


Figure 6.1 – Quick to Learn

The learning curve of the simulator was an important factor to evaluate because it would reflect how much assistance a user would require to use the simulator. All the participants agreed that the simulator was quick to learn. Again this reflects the simplicity of the interface, as well as demonstrating the well implemented design of simulating the processes of encryption and decryption.

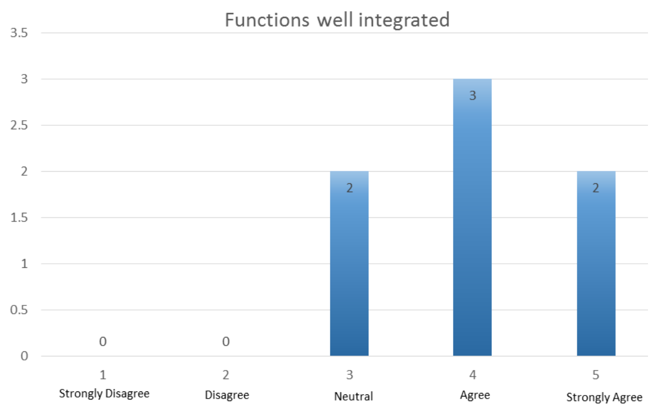


Figure 6.2 – Functions well Integrated

The majority of the participants felt that the functions of the simulator were well integrated. This was an important factor for this project because it reflected how well the simulation provided information on the encryption and decryption processes.

Overall evaluation of the simulator, undertaken by the participants, provided valuable feedback to the author in terms of the current state of the simulator as well as possible features which could be implemented in the future. It is believed that the general census among the participants was that the simulator was well implemented and designed. One participant in particular did not expect the simulation to be 3D but instead 2D. The participant was, however, impressed with the author's ability to be able develop a 3D version. A specific problem which did arise with various participants was that they wanted to copy the encrypted or decrypted text so they could paste the text to perform the opposite process. Due to the simulator being unable to facilitate for a copy and paste functionality at the time, the participants found themselves writing the text down by hand which became a nuisance. An interesting observation made was that none of the participants seemed to navigate to the help section. This may have been because they were already told about the simulator, but even then they were not informed that the simulator had the capabilities to view the rotor from different angles. The participants almost stumbled upon the controls by accident rather than navigating to the help section or even asking during the evaluation. This was interesting because having never used the simulator before, they were able to quickly grasp an understanding of it which demonstrated the usability of the simulator.

As described later in this section, the author decided to take on board the feedback and redesign the interface to facilitate a copy feature.

6.1.2 Nielsen's Heuristics

Throughout development of the simulator the author was conscious of Nielsen's Heuristics, [22] which are broad rules of thumb for design rather than guidelines. These aided the author in making design decisions regarding the user interface to ensure the simulator met these considerations.

6.1.2.1 Visibility of system status

Feedback was provided to the user about the current mode, main menu, encryption or decryption. This was done by renaming the title bar to reflect the appropriate mode. Also, to ensure that users knew that the text box was active, an insertion cursor appears. Upon entering a letter to process, the simulator would animate and produce the current path while rotating the required steps. This provided feedback to the users to make them aware that the simulator is performing the process.

6.1.2.2 Match between system and the real world

The choice of wording was chosen carefully, especially in the help section, to ensure not too much technical jargon was included. The wording did not confuse any participants as they were able to understand the phrases used.

6.1.2.3 User control and freedom

This aspect was important because if the user was to click on the decrypt function instead of the encrypt function they would require a function to allow them to go back to the previous menu. This was taken into consideration immediately as it formed part of the menu navigation system. This provides the user with an emergency exit if they were to enter a function by mistake, without having to go through an extended interaction, such as restarting the simulator. To facilitate this a button, which redirects the user back to the main menu, was implemented. Another consideration taken into account was what if the user was to erase a letter from the text field, should the rotor reverse one step? The answer was yes because this demonstrates to the user that they have erased a letter as well as the corresponding letter being erased from the opposing text field. This essentially supports an undo function. More consideration could have been given to assist the users with accidental interactions.

6.1.2.4 Consistency and standards

The theme of the simulator remains consistent throughout the interface. The font remains white. Buttons are consistently the same shade of grey. The interface layout for the encryption and decryption functions remains consistent. Another important factor in terms of consistency was the wording. This was to ensure the same terminology was used throughout, which aided to decrease the cognitive load on users while describing certain aspects in the help section. None of the participants provided feedback on this aspect, however.

6.1.2.5 Error prevention

Rather than displaying error messages, it was important to prevent the problem from occurring in the first place. An important scenario to address was what if the user was to press backspace, and no text was present in the text field. Instead of displaying a message, the simulator does nothing in this scenario. Another scenario would be when viewing the rotor details. Since it is connected with the help section, closing the help section should close, if open, the rotor details also, and return the user to the previous state.

6.1.2.6 Recognition rather than recall

It is important to ensure minimum cognitive load on users. They should be able to recognise the interface rather than try to remember it. To comply with this consideration the encryption and decryption screens remain consistent. This was reflected in the evaluation results with all the participants agreeing the simulator was easy to use and quick to learn. This ties in with section 6.1.2.4.

6.1.2.7 Flexibility and efficiency of use

Due to the simplicity of the user interface the simulator caters for novice and experienced users equally. Unfortunately, consideration was not given to shortcuts to optimize certain tasks, which could have aided expert users. This is an aspect which the author feels could definitely be improved upon.

6.1.2.8 Aesthetic and minimalist design

The interface presented to the user upon entering the encryption or decryption screens included only the necessary information. The button, which presents the user with the image of the rotor details, which was implemented in the help section, was initially designed to be included in the main simulation screen, beside the help button. However, the author decided that this should be included in the help section rather than having an additional button to clutter the interface.

6.1.2.9 Help users recognize, diagnose and recover from errors

The interface and user input fields have been designed to ensure that no errors occur. Because the user can only enter letters and not numbers, symbols or utilize the spacebar, no feedback as such was provided as nothing is entered. This was clearly an aspect which could be improved upon, perhaps by providing a tooltip with information.

6.1.2.10 Help and documentation

Every effort was carried out ensure that the simulator remained intuitive and simple, especially the help section to ensure it was lightweight. However, users may require further assistance and a help section may not be enough. A user guide has been created to provide further assistance. Whether a novice user or an expert user, they can still refer to this guide for assistance. It provides a more in-depth guidance than the help section and more explanation on certain aspects. Furthermore a video tutorial, which can be found in Appendix S, was also created to demonstrate a live run of the processes.

6.2 Evaluation of Requirements

The set of requirements specified during the start of the project were mostly met. All the requirements which had high priorities were all met.

Please refer to Appendix O for numbering details.

6.2.1 Functional Requirements

R8, *R11* and *R14* were not met, mainly due to time constraints. However, these were not essential to the simulator, only desirable.
R8 and *R11* were the requirements that were stated to include three rotors, in order to emulate the Enigma machine. However, after much discussion with the supervisor, it was realized that implementing three rotors would be difficult to fit on to one screen within a reasonable size for the user to view. In addition, time constraints did not allow for further development for these requirements to be met.
R14 was designed to add a type of problem-solving game to the simulator, acting like a teaching tool to decrypt the text manually. The author thought that this may detract from the idea of the simulator, therefore this requirement was not fulfilled. In addition, time constraints did not allow further development.

6.2.2 Non-Functional Requirements

From these set of requirements only *R17*, making the simulator compatible for Mac and Linux operating systems, was not met. However, during user evaluation this did not seem an issue as all the participants used the simulator on a Windows operating system. This was mainly due to the author unable to access a Mac or Linux operating system.

Further results and information can be found in Appendix F.

6.3 Re-Design

Once evaluation was complete, the author decided to redesign the interface to facilitate a copy and paste feature. This was highly sought of from many participants during evaluation. While minimal changes were made to the interface, only a button was added (See Appendix F), the code required further modifications. In the process of facilitating copy and paste, the backspace bug (described in section 5.4.3) was resolved. Before this, the encryption and decryption processes occurred upon a key press, the simulator now processes the entire string of the input box in the event loop. Processing this repeatedly for each iteration of the loop was thought to be inefficient therefore a boolean variable was added. This was to ensure after the input string was processed upon a key press that it was not done again until the next key press occurred, where the whole input string would be processed. As this was a minor change, testing was not required from the participants but the author did check for robustness and reliability.

7 Description of the final product

The aim of the project was to develop a simulator which would graphically demonstrate the encryption and decryption processes which occurred within an Enigma machine. This was achieved with the aid of C++ and OpenGL, which were used to develop a 3D graphical simulator along with ImGui to implement a graphical user interface and Blender to model the components of the rotor.

8 Summary and Conclusions

8.1 Future Work

There are various improvements and extensions which could be made to the simulator. Meeting the unfulfilled requirements would present a good starting point for future development. The highest priority of those requirements would be to add two additional rotors so the simulator can display three rotors. This would further explain the processes of encryption and decryption to users. As well as adding further detail, the simulator would become more appealing simply due to the fact it could emulate a real Enigma machine with three rotors and more graphics would be present on the screen.

Supporting multiple platforms can also be seen as an improvement. Since the author was not able to test on any of the desired operating systems stated in the requirements (Appendix O) except Microsoft Windows, supporting multiple platforms would allow a wider range of users to run the simulator on which ever platform they require.

The other requirement, which was not met, was to implement an attack method, which would almost create a game where the user would attempt to solve the original state of the output text. This would add an additional element to the simulator. However, this could also be created as a separate application rather than being a part of the simulator.

When the simulator starts up, all the objects must be loaded in one by one. Currently this takes a few seconds, due to the number of vertices which make up the objects. This is certainly an area the author would like to improve upon because of the delay caused in starting up the simulator. In order to enhance loading times, the class which loads in the objects could perhaps be altered or the objects themselves be modified, so that they are constructed of less vertices without losing their quality.

The rotor components are not the most aesthetically appealing graphics. This is certainly an area the author feels could be improved upon. The alphabet tyre cannot be distinguished clearly without the aid of the diagram in the help section. A texture could have be added with each letter of the alphabet or, like many of the real rotors, numbers ranging from 1 to 26. However, on the real rotors the numbers are engraved or in some cases they are embossed. In order to simulate this effect, the easiest solution would be to modify the existing model in Blender. In addition to the alphabet tyre, various other components would benefit from remodelling, mainly the spring-loaded ring adjusting lever, finger wheel and ratchet wheel. Due to the lack of experience using Blender, combined with the complexity of these components, the author was unable to model these components precisely, in comparison to their real life counterparts. The spring-loaded ring adjusting lever could not be modelled to precision due to the sources which were unable to provide a complete reference of the component. Most of the finger wheel and ratchet wheel were modelled to an acceptable level, however, only the outer teeth of the components were unable to be modelled correctly. Given more time, the author feels that this may have been achieved. To add more aesthetic appeal, each individual component could have been textured to produce a rough metal look. This was not implemented in this project for two reasons, mainly time constraints. In addition, these types of changes seemed out of the scope of the project. However, all the components could be remodelled to increase precision because if a user was to zoom in on a large scale they may see some incorrect alignments, in particular with the current path. In addition to being remodelled the components could be completely overhauled and redesigned entirely. The author sketched each component with rough estimations, revision in this area could increase the precision of the components.

The theme of the simulator could also be revised. While the author assumed that the dark theme looked sleek, some may feel it looked dull and boring. Also, the font used in the simulator felt old. The author did search for fonts which were smooth, however, OpenGL did not render them smoothly but instead they were blurry and did not offer any appeal, but enabling anti-aliasing could solve this, but time constraints prevented this from being implemented. In addition to the theme, a transition of menu navigation could be implemented. When a user selects an option which deviates from the current screen, animation could occur to hide the other items. A form of transition was implemented, however, but only when the rotor details section was displayed. Adding this effect to further menu navigations could enhance the users' experience of the simulator. As the view of the rotor is exploded, allowing the user to view the rotor implode back into form would be interesting.

Adding shortcuts to quick views would also be useful. Instead of holding down one of the controls to move the view to a certain angle, a user could press a combination of key presses to initiate a quick view change. Upon these key presses, the view would instantly change to, for example, a front view of the rotor. This would save the user time from having to manually change the view. In addition, the mouse could be used to move the rotor. A final suggestion for improvement would be the interaction with the rotor. To find details about the rotor, the user must go into the help section and then click another button. This seems like too much of an interaction to just view an image which occupies a vast majority of the screen. A solution to improve this could be to present the user with the component names when they hover over them. This would introduce a new level of aesthetic appeal to the simulator as well as adding dynamic interactions.

In addition to these improvements, an extension to the project would be to develop an application for mobile devices or to embed into a website for universal access.

In order to run the simulator successfully OpenGL 4 or greater is required and Visual Studio must be installed. It would have been ideal to deploy the simulator to enable users to install it on their computer.

8.2 Conclusion

Overall, the project can be deemed a success as it demonstrates the processes of encryption and decryption within an Enigma machine. The simulator can be seen as a prototype version, which demonstrates the use of one rotor by visually illustrating the paths of the current and providing the user with output text simultaneously. With this in mind future development for a complete three rotor simulator is in place, with the code being easily extendable. With a lack of resources to demonstrate the processes of an Enigma machine, this project can be seen as unique, especially given that the demonstration is represented graphically with a 3D rotor. One of the main reasons behind the success of this project, was that time was managed carefully with the highest priority requirements met first.

In terms of design, the author ensured the interface was simplistic as possible. This was to ensure that the rotor was able to be implemented to a reasonable size which allowed users to understand the processes of encryption and decryption upon a key press.

As the main purpose was to demonstrate the processes of encryption and decryption, a high level of importance was given to ensure that the simulator was robust and reliable in terms of the providing the correct output. With this in mind, the author chose to concentrate on the logical side of the simulator first. Once the logic was complete, the graphics and user interface were then optimized to improve aesthetic appeal and usability.

Throughout testing and user evaluation, the code was revised constantly to improve certain aspects. After testing and user evaluation were complete, new features were added to the simulator along with several bug fixes. The two notable features added, were a guide for users in the simulator itself, which illustrates the component names and the other feature being the ability to facilitate copy and paste, instead of users attempting to remember or write by hand the output text they wish to operate on. The main factors of the inclusion of these features was due to the feedback received during evaluation and the observations made.

8.3 Critical Appraisal

There was a relatively steep learning curve to undertake in this project. Prior to the undertaking the project, the author had no knowledge of OpenGL or any type of 3D graphical programming language. In addition to this learning curve, Blender was also a new experience. Nevertheless previous work with modelling software somewhat eased the learning curve. These learning curves were eased as the project progressed. This was due to the author undertaking the Graphics module, which provided a template to help commence with implementation. The choice of programming language was rather constrained however. This was because during the Graphics module, C++ was used during the assignments and labs, therefore it felt wasteful to explore additional options with no experience using other programming languages combined with OpenGL. In addition to the constraint on choice of programming language, a graphical user interface was required, which was not used during the Graphics module, hence the need to research graphical user interface libraries compatible with modern OpenGL and C++. With a variety to choose from, the author felt that choosing the library which would take the least amount of time to integrate with OpenGL was vital, therefore ImGui was chosen. The choice of resources proved to be the correct decisions. Basic knowledge was acquired about the Enigma machine prior to the project as it was briefly taught in a previous module in university.

Throughout implementation, the author was cautious of time. Due to this, three rotors were not able to be implemented, which can be concluded as a negative aspect of this project. However, the implementation of one rotor demonstrates the intake of knowledge consumed by the author, taking into consideration no knowledge of OpenGL or 3D graphics programming was present prior to this project. Most of the tasks outlined in the Gantt chart were completed on schedule with the exception of implementation, which took slightly more time to complete.

During evaluation, the author was able to observe users using the simulator for the first time, which provided to be a valuable experience. From observations and feedback received, the author was able to revise and implement new features to the simulator. These features enhanced the usability of the simulator. Only 7 participants were able to be gathered for the evaluation. The author felt that being able to gather more participants would have provided a wider range of feedback.

The simulator could be progressed further by simulating all three rotors, which given more time the author feels could have been achieved. The structure of the code is in place to be extended without too much disruption to the current flow of events.

Overall, the project can be considered a success because the main requirements were met. This type of programming was a brand new experience, where a lot of valuable skills were learnt, as well as learning about the Enigma machine. The main skills learnt were the use of Blender to create 3D models, OpenGL and 3D graphics programming language. In addition to learning new skills, the author's C++ skills were strengthened. A valuable lesson taken from this project can be to complete the logical side of an application first, and then enhance the aesthetics. The fundamentals of any application is to ensure the logistics are functioning correctly. This is definitely a lesson the author will take forward after graduation.

Acknowledgments

The author would like to thank the supervisor, Professor John Arnott, for supervising this project. His guidance and support played a vital role to ensure that the project was completed to an acceptable standard and on schedule, as well as sharing his knowledge of the Enigma machine with the author.

Also, a special gratitude must be given to the participants who underwent evaluation of the simulator and in providing the author with valuable feedback.

References

[1] Royal Naval Museum. (2002). *The Enigma Machine.* Available: www.royalnavalmuseum.org/info_sheets_enigma.htm. Last accessed 28/04/2015.

[2] Rijmenants D. (2005). *The German Enigma Cipher Machine.* Available: http://users.telenet.be/d.rijmenants/en/enigma.htm. Last accessed 28/04/2015.

[3] History. (2013). *Code Breaking.* Available: http://www.history.co.uk/study-topics/history-of-ww2/code-breaking. Last accessed 28/04/2015.

[4] Crypto Museum. (2014). *History of the Enigma.* Available: http://www.cryptomuseum.com/crypto/enigma/hist.htm. Last accessed 28/04/2015.

[5] ILord. (n.d.). *German Enigma Machine.* Available: http://www.ilord.com/enigma.html. Last accessed 28/04/2015.

[6] Frank Spiess. (2009). *Three Rotor Enigma Simulation.* Available: http://www.Enigmaco.de. Last accessed 28/04/2015.

[7] Voltreach. (n.d.). *Development Methodologies.* Available: http://www.voltreach.com/Development_Methodologies.aspx. Last accessed 28/04/2015.

[8] Blender Foundation. (n.d.). *Blender.* Available: http://www.blender.org. Last accessed 28/04/2015.

[9] Jerry Proc. (2011). *Enigma.* Available: http://www.jproc.ca/crypto/enigma.html. Last accessed 28/04/2015.

[10] BlenderWiki. (n.d.). *Extensions: 2.6 Add Gear Object.* Available: http://wiki.blender.org/index.php/Extensions:2.6/Py/Scripts/Add_Mesh/Add_Gear. Last accessed 28/04/2015.

[11] CEGUI. (n.d.). *CEGUI.* Available: http://cegui.org.uk. Last accessed 28/04/2015.

[12] LibRocket. (n.d.). *LibRocket.* Available: http://www.librocket.com. Last accessed 28/04/2015.

[13] LibUFO. (n.d). *LibUFO.* Available: http://libufo.sourceforge.net. Last accessed 28/04/2015.

[14] AntTweakBar. (n.d.). *AntTweakBar.* Available: http://anttweakbar.sourceforge.net. Last accessed 28/04/2015.

[15] ImGui. (n.d.). *ImGui.* Available: http://github.com/ocornut/imgui. Last accessed 28/04/2015.

[16] Qt. (n.d.). *Qt.* Available: http://qt-project.org. Last accessed 28/04/2015.

[17] FLTK. (n.d.). *Fast Light Toolkit.* Available: http://fltk.org. Last accessed 28/04/2015.

[18] D. Shreiner, G. Sellers, J. Kessenich and B. Licea-Kane (2013), *'OpenGL Programming Guide 8th Edition'* Version 4.3, pp. 34-36.

[19] GLFW. (n.d.). *GLFW – An OpenGL Library.* Available: http://www.glfw.org. Last accessed 28/04/2015.

[20] Lonesock. (n.d.). *Simple OpenGL Image Library.* Available: http://www.lonesock.net/soil.html. Last accessed 28/04/2015.

[21] Usability. (n.d.). *System Usability Scale (SUS).* Available: http://www.usability.gov/how-to-and-tools/methods/system-usability-scale.html. Last accessed 28/04/2015.

[22] Jakob Nielsen. (2005). *10 Heuristics for User Interface Design.* Available: http://www.nngroup.com/articles/ten-usability-heuristics. Last accessed 28/04/2015.

Appendices

Appendix A – Ethics Approval

Appendix B – Black Box Testing

Appendix C – Blender associated files

Appendix D – Design Sketches and Prototypes

Appendix E – Ethics Application

Appendix F – Evaluation Summary

Appendix G – Gantt chart

Appendix H – Interface Design

Appendix I – Mid-Project Progress Report

Appendix J – Minutes of Meetings

Appendix K – Poster

Appendix L – Requirements

Appendix M – Screenshots

Appendix N – Source Code

Appendix O – Specification Requirement

Appendix P – SUS Results

Appendix Q – Use Case

Appendix R – User Guide

Appendix S – Video Tutorial

Appendix T – White Box Testing