NEA Project

ANALYSIS

# Background Information

At AF School the computer science lessons are intended to encourage students and show them everything the topic has to offer. One section of computer science is Ciphers, it entails the enciphering and deciphering process of many different ciphers, including; Caesar, Substitution, and Transposition. As well as including, frequency analysis, and brute force methods for decryption. The section is often enjoyed by students as it can be fun and puzzling, giving students the opportunity to compete solving problems, whilst using algorithms. It is this side of the lessons that the students respond well to as it pushes them to think about the solution from a computational and mathematical point of view.

Mr. C thinks that some of the younger students would also benefit from learning ciphers due to its benefits in promoting critical thinking. With the older students Mr. C gets them to program some of the algorithms required in the ciphering and deciphering. This is often in done in a language like python that the older students have an understanding of. The students may use a library like pycipher to speed up the process, especially when layering multiple ciphers together.

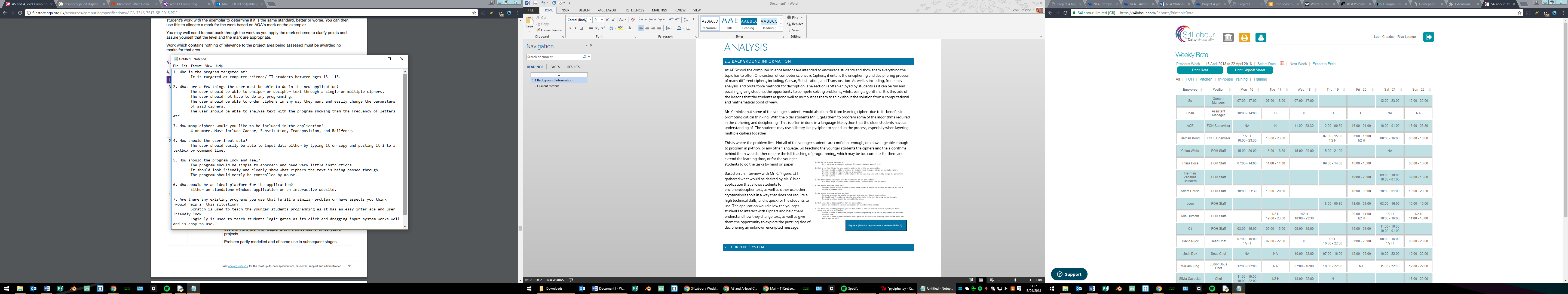
This is where the problem lies. Not all of the younger students are confident enough, or knowledgeable enough to program in python, or any other language. So teaching the younger students the ciphers and the algorithms behind them would either require the full teaching of programming, which may be too complex for them and extend the learning time, or for the younger students to do the tasks by hand on paper.

Figure 1. (Solution requirements interview with Mr. C)

Based on an interview with Mr. C (Figure. 1) I gathered what would be desired by Mr. C is an application that allows students to encipher/decipher text, as well as other use other cryptanalysis tools in a way that does not require a high technical skills, and is quick for the students to use. The application would allow the younger students to interact with Ciphers and help them understand how they change text, as well as give them the opportunity to explore the puzzling side of deciphering an unknown encrypted message.

# 1.2 Current System

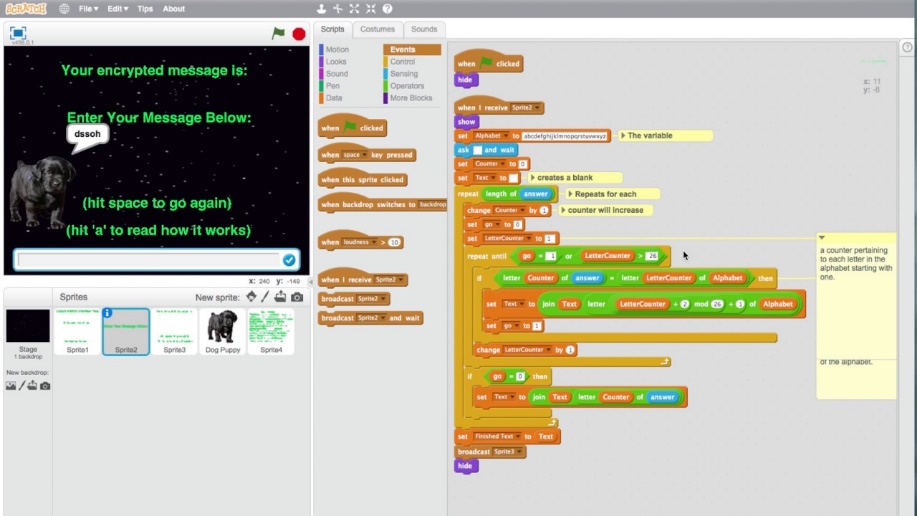
Currently Mr. C plans on teaching the younger students Ciphers by having them complete the algorithms and encryption processes on paper. By having the students follow along with him completing the algorithms on the class whiteboard to begin with, then having them complete handed out questions by themselves.

Completing the algorithms by hand would lengthen the teaching process as writing everything out is far slower than typing it into a computer. Furthermore manually writing algorithms can leave a lot of room for errors and mistakes. Mistakes can leave a younger student feeling demotivated and confused about the algorithm, so by avoiding these a student would have a far more fulfilling learning experience.

The alternative would be teaching the younger students python and using the pycipher package to aid the students. The benefit of using the pycipher library is that the younger students would not have to program the ciphers themselves, making the teaching of the programming a bit easier as only the basics could be covered. The disadvantage to this solution is that a programming interface can seem daunting to someone who does not fully understand it, and has very little experience with the concept. Since it may be the students first introduction to programming Mr. C will be having to deal with any problems they have with the programming side as well as the understanding of the Ciphers. This complicates his job as well as the students’ involvement.

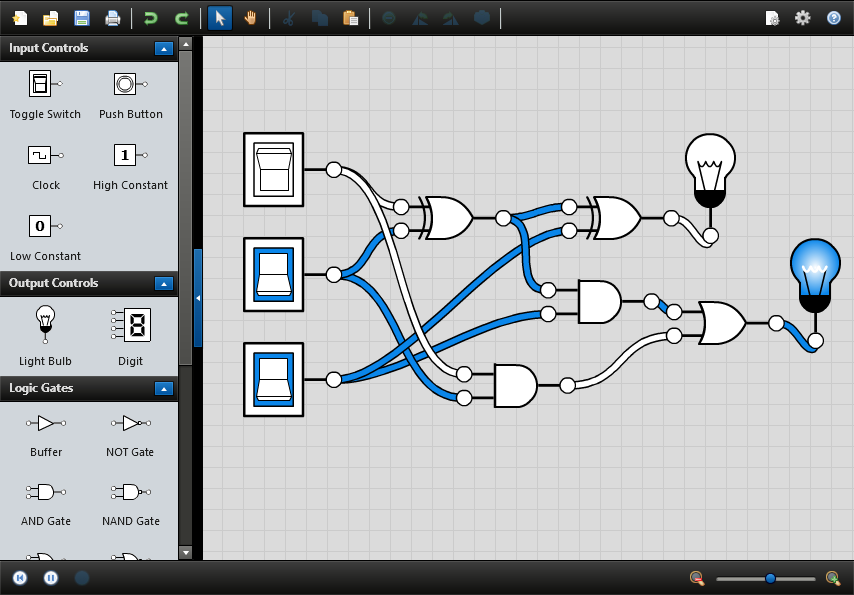
# 1.3 Research

In the interview with Mr. C (Figure 1) he references two programs that he commends the interfaces and input systems for. By analysing these programs it may help with the design of the solutions user interface.



The figure to the left shows an example of a Caesar cipher in Scratch. As you can see Scratch uses a block based building structure to create a runnable script. This is favoured by Mr. C for teaching young students as the interface is friendlier than standard programming and limits what the user can do, reducing the likelihood of them making mistakes. By having blocks that can be dragged around and re-ordered, the user can clearly see what order things are happening in and how the output was achieved. It also gives the user freedom to create what they want to get the output they require.

A similar block building could work for the new application with each block being a cipher and having it start from a text input block. Also the concept of the user being able to move all the functions around and order them works well in showing them which order things are happening and the flow of the program.



Next is a program called Logic.ly, it allows users to create virtual circuits using logic gates which it connects by drawing wires between the plugs on each gate. The gates can be controlled by a set of switches and buttons, and the output can be displayed on bulbs. This turns what can be quite a difficult topic to understand into something very easy to use. The wire and plug method is intuitive in showing how inputs and outputs connect, and how data is flowing. Additionally the by having gates that can be added, moved, and connected in so many ways the user has freedom in what they want to do with the program.

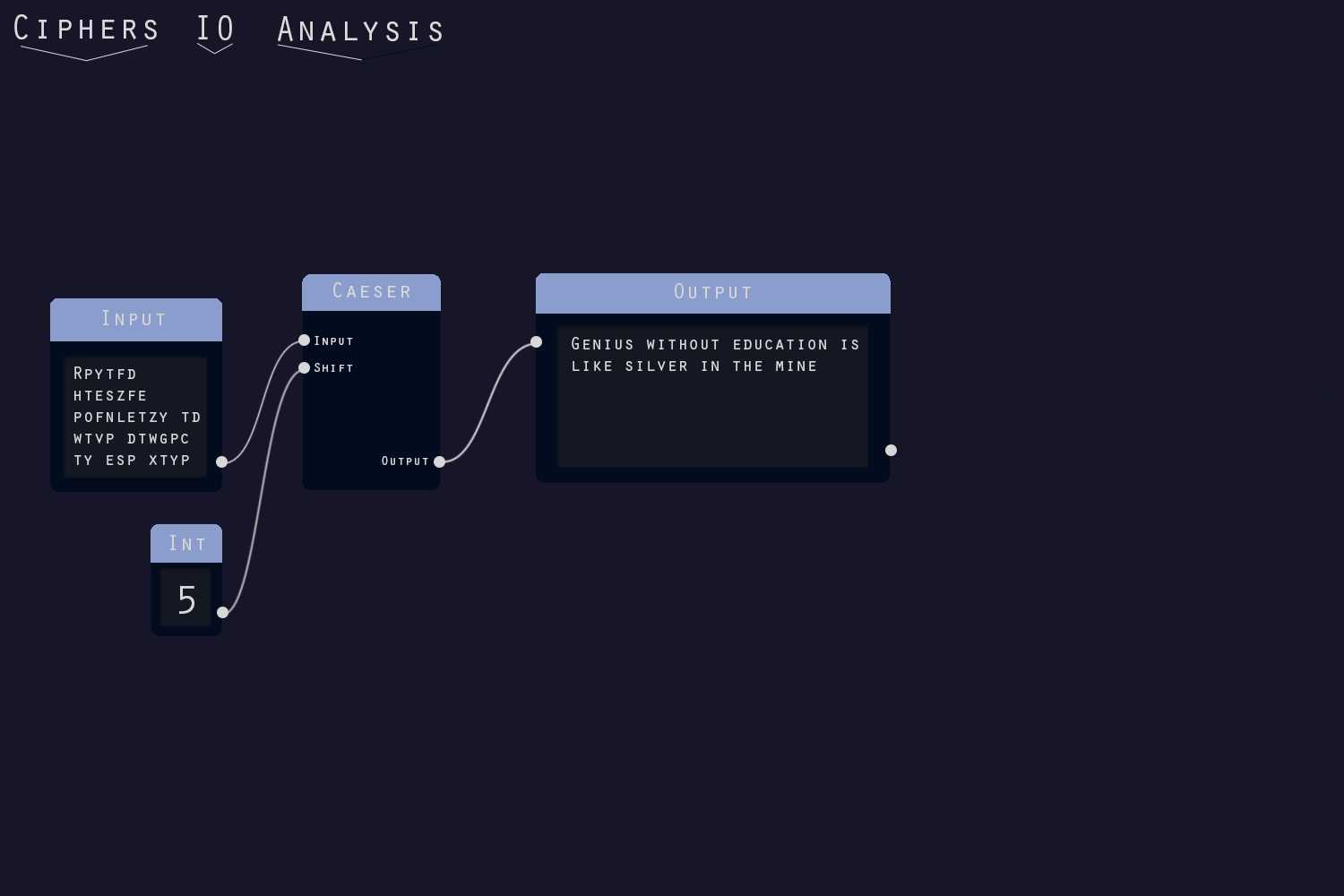
The concept of having modules with plugs that wire up together could work well in the new application as it shows clearly where data is being transferred, and it allows the users graph to be dynamically expandable, giving them the opportunity to make more complex things.

# 1.4 Specification of user needs

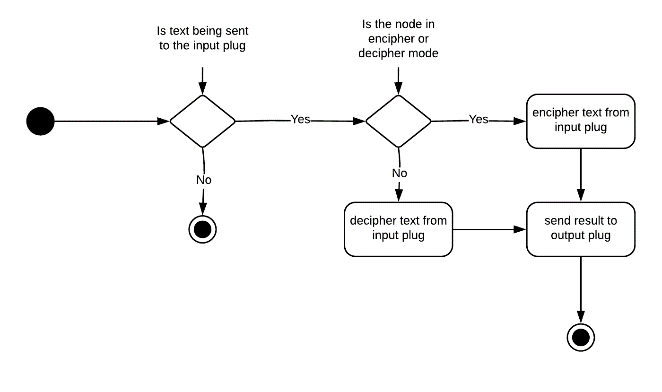
Since the new application is a replacement for either, hand written enciphering/deciphering or python scripting, the new application must not limit the user within the layers of ciphering/deciphering they may want. E.g. it should not be a simple program that just takes user input runs it through one output and displays the output. The User should be able to layer on as many ciphers as they please. Just like they could if they were writing them all out by hand, or running multiple functions in python.

Main requirements  
1. The program can accurately encipher and decipher text using; Caesar cipher, substitution cipher, transposition cipher, and rail fence cipher.  
2. The user should be able to input their own text into the ciphers  
3. All of the ciphers parameter can be changed and set by the user  
4. There is a way for the user to analyse text for the most frequent letters   
5. The user can add ciphers and order them in any way they want, and have them feed into each other  
6. There are tools to help the user brute force an unknown cipher  
7. The applications interface should be easy to use. Meaning most of the user interface is controlled by mouse, so the application works like a drag and drop editor. The user should only have to type their text input and parameters into the program.  
8. The user can copy and paste text into and out of the program  
9. The program should clearly display any errors the user makes when setting up there cipher, and informs them what the problem is.  
10. Each cipher should be independent of the other ciphers in the program including other instances of the same cipher. By this, I mean the parameters can be changed independently of the other ciphers.  
11. The user should be able to see how the text has changed between each cipher.

# 1.5 planned solution

For the new application I have chosen to create a system where the user can create boxes that contain functions, and connect them together with wires by plugging one boxes output into another boxes input. The functions these boxes contain will be the enciphering and deciphering process, and there will be a box for each cipher. The parameters also be plugged into the box, and whether or not it is enciphering or deciphering will be toggled on the box. Text can be input into the program by another set of boxes that only have output plugs and contain text boxes for the users input. Similarly the output will be shown by an output box that only has an input plug, the box simply displays the plaintext of whatever is connected to the box. Outputs can also be viewed on the wires, so that each wire can show whatever values it is connecting. Many boxes can be linked together and branched to the users’ needs, allowing them to create any sort of cipher they require. The boxes work like nodes and the wires are connecting them so they are able to share data.

The figure to the left is mock-up of what the UI should look like. The connections between nodes is shown by a Bezier curve being drawn between the connected plugs. The Caesar cipher does not require a encipher/decipher toggle since the algorithm is the same all that changes is the shift amount, which anyone familiar with the Caesar cipher knows to decipher you simply set the count to 26 – the enciphering count.

This diagram shows how the basic logic of a cipher node will work. It demonstrates how the node will check for valid inputs and alter the inputs accordingly.

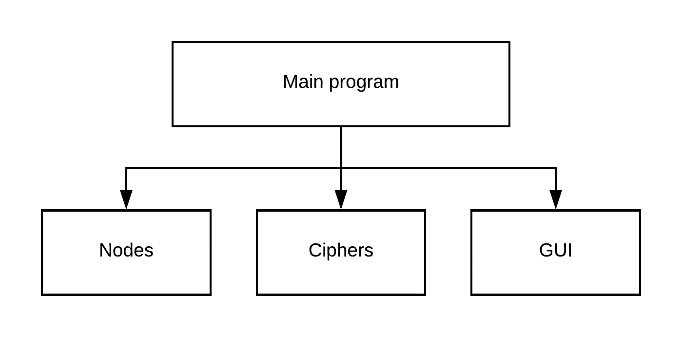
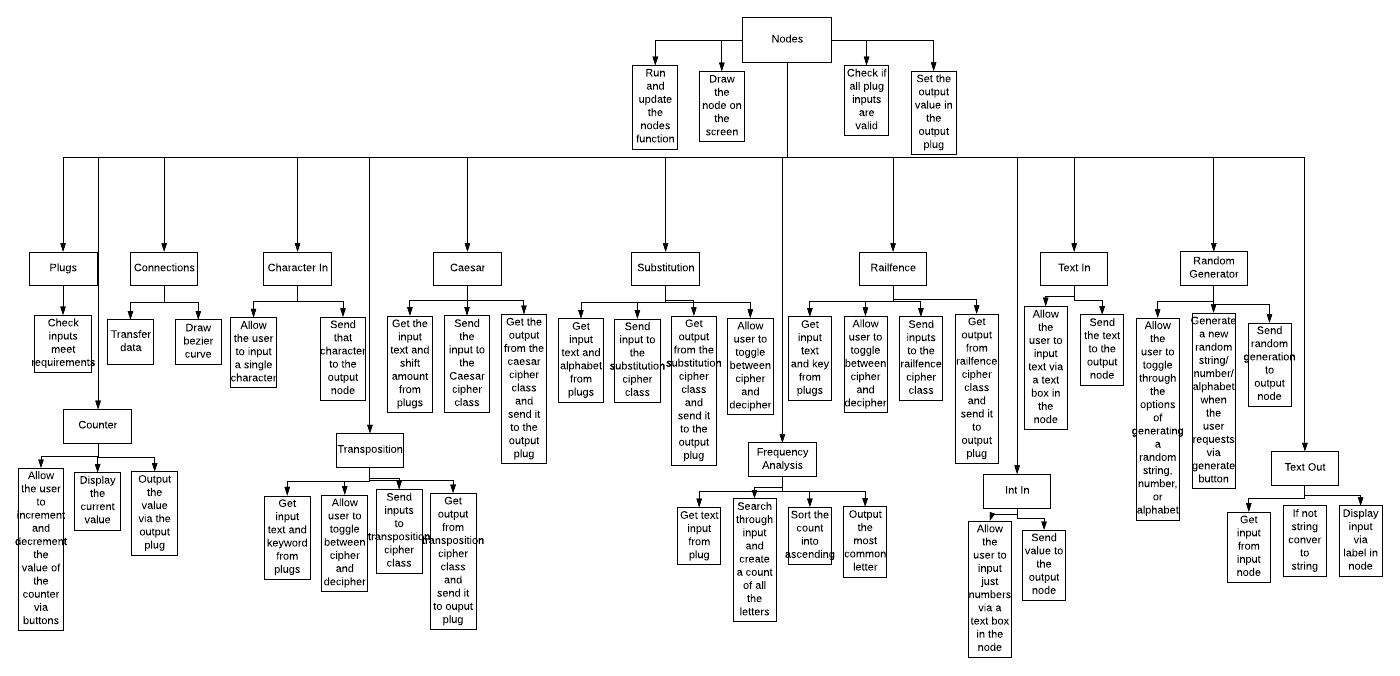
design

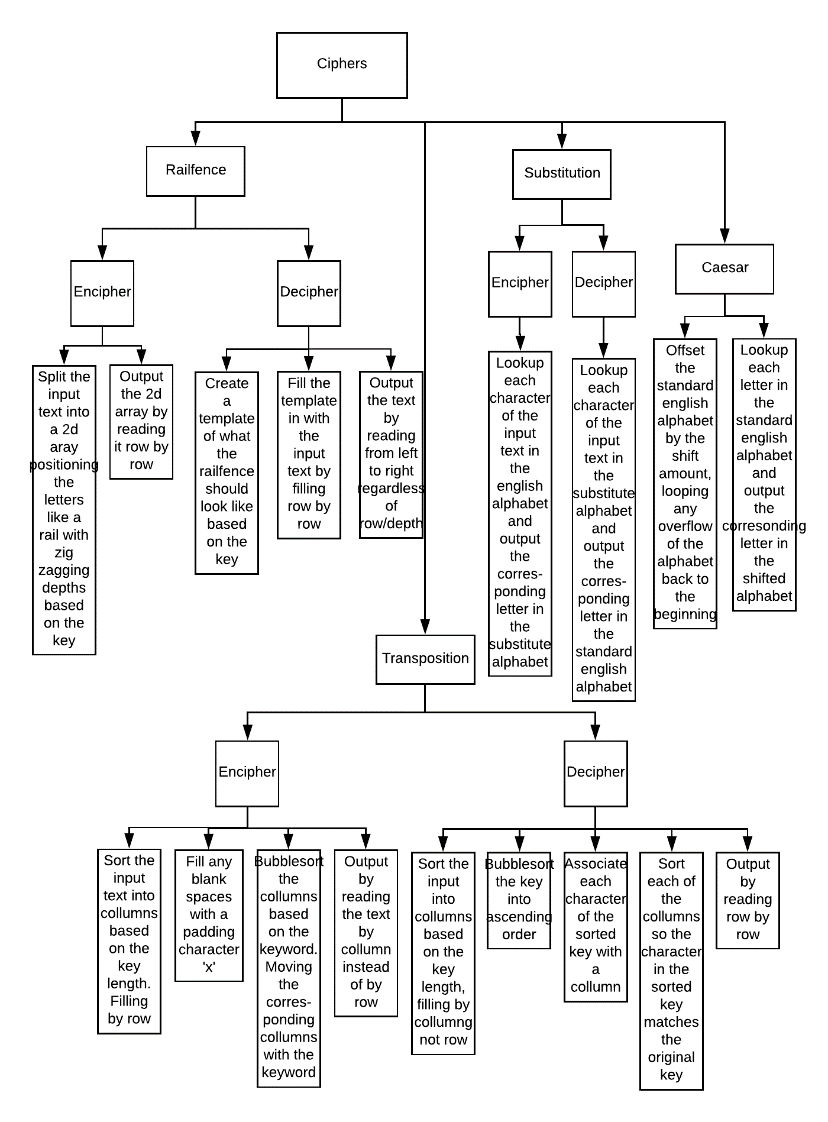
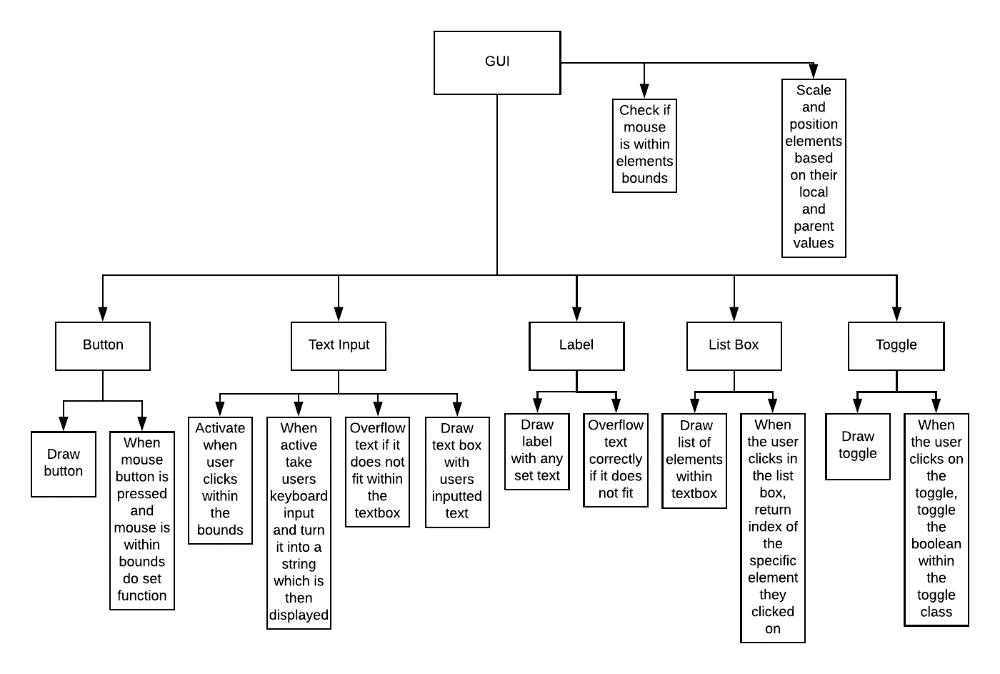
# 2.1 System specification

The application will be written in Java with use of the Processing 3 library and IDE. The reason I chose Java for this application is mainly because of its object orientated structure. Since the user needs to be able to dynamically create and structure custom objects an object orientated structure is a necessity so the objects can be instantiated into the program. Another benefit to Java is its automated memory management. Since objects are being created and destroyed throughout the programs runtime, an automated memory management would speed up development and make the program more reliable, only at the cost of performance.

The Processing 3 library is a graphics library. Not a UI library it has no built in buttons or text boxes, its main use is for drawing graphics on screen. It makes drawing boxes and text on screen a lot easier as it bridges between the application and OpenGL, simplifying the rendering process. I chose this over a standard UI library since no current UI library has support for a node editor, so either I would have to adapt one or build one from scratch. Processing makes it easier to build one from scratch as I don’t have to interface with OpenGL or any other graphics framework, instead I can focus on the functionality of the UI. Processing also simplifies mouse and keyboard inputs, further easing one of the main challenges of this application. Other that processing I will use some of the built in Java libraries, mainly for the use of getting and setting the computers clipboard, allowing for the implementation of a copy and paste functionality in the programming.

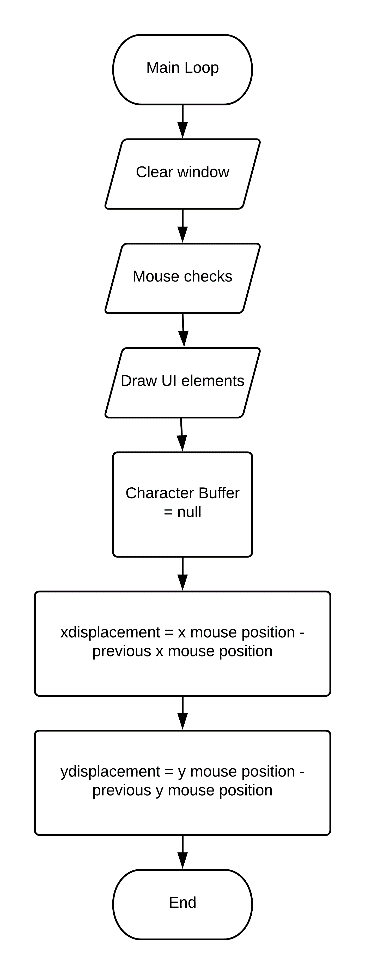
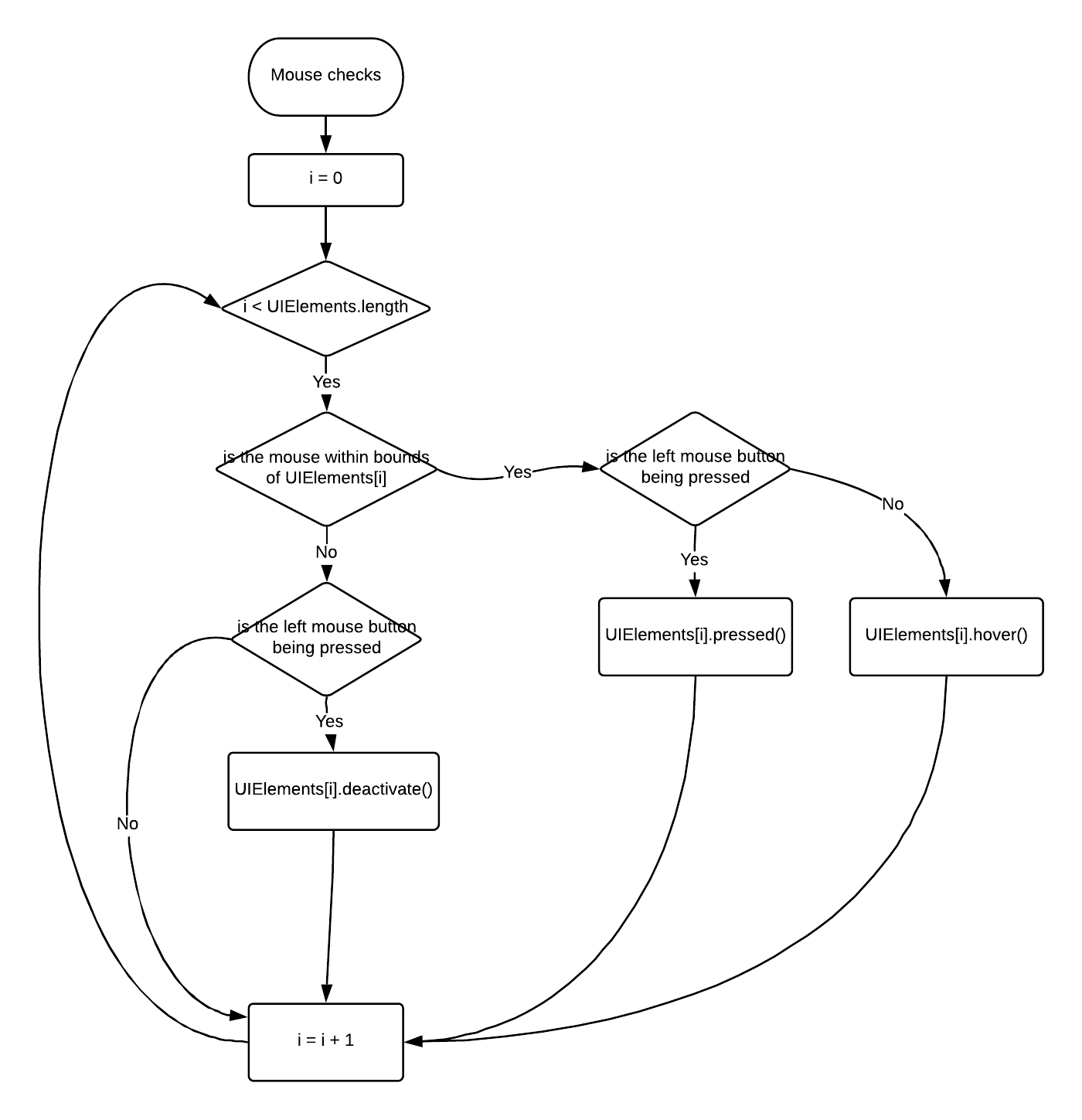
# 2.2 Program overview

The program is split into 3 mains sections. The Ciphers which contains all of the algorithms and information for the ciphers. The UI which consists of logic and objects for each of the required UI elements. Lastly there is The Nodes which encompasses all of the objects and methods surrounding the Node editor, including; Connections, Plugs, and The canvas which all the nodes are contained in at runtime. The only parts of the program outside of these three sections are the parts that handle the updating and drawing of the elements on screen and manage the input of the user.



The diagrams above shows the programs functionality from a top down perspective for each section of the program. It also outlines each a lot of the objects required for the program.

The actual program combines each of these parts by having each node contain the cipher and/or UI elements it requires, and then having the inner workings of the node interface the the elements together.The nodes are then instantiated by the user pressing buttons that are stored in the main class of the program. The pressing of those buttons are controlled by the main loop running mouse checks on all the UI elements.

These two flow charts show the main loop, which is the function that is called when each frame is drawn in the program, and any functions that branch from that. The main loop starts with the window being cleared. This is so the UI elements are drawn on to a blank canvas and prevents visual artifacts. Next the mouse check function is called. The function loops through all the UI elements in the program and calls functions on the element based on the mouses behaviour. Next all the UI elements are drawn. This is done by calling the update function within each element and having it draw itself. Then the character buffer is cleared. The character buffer is a list of all the characters that were input during the frame before. By using a buffer if more than one character is input during a single frame then the buffer ensures no characters are missed. After that the x and y displacements are set. These are just the difference in pixels between the current mouse position, and the previous one. These values are used for moving UI objects like nodes with the mouse. Finally at the end the loop just repeats itself, this continues until the program is halted.

# 2.3 Specification of objects

The only new data type I require is an Alphabet type. This is used in the substitution cipher. It’s quite simply an array of 26 characters, no more or less. The data type will simply have a get and set character that will take an index of the alphabet and change or return the character accordingly.

## 2.3.1 GUI ELEMENT SPECIFICATIONs

## 2.3.2 Specification of Nodes

# 2.4 Algorithms used