# Pi Keyboard

**May Cook**
**Exam Number: [Removed]**
**Centre: [Removed]**

# Contents

# 1 - Analysis

## 1.1 - Problem Identification

Desktop computers are commonplace, found in almost every home and workplace in the modern day. Although these devices lend themselves to a variety of specialist applications, these often require specialist peripheral hardware such as a Musical Instrument Digital Interface (MIDI) keyboard (shown in figure 1.1.1.1), in addition to or in place of the keyboard and mouse, that are typical of a desktop computer.

I would like to explore the alterations that could be made to the hardware and software used in keyboards and in specialist peripheral devices, to create a device that can be customised to perform specialised functions, whilst retaining as much of the functionality of a conventional keyboard as possible.


**Figure 1.1.1.1 - Example of a MIDI keyboard**

## 1.2 - Stakeholders

Anyone who uses a computer on a regular basis, particularly those who require specialist functions such as:

- **Musicians**

  Musicians use a wide range of specialist hardware such as mixing desks and MIDI input devices. These can require large amounts of physical space that may not be available, making these devices inconvenient.

- **Gamers**

  There are many keyboards and keyboard replacements that are marketed specifically for gamers, as there are many functions, shortcuts and keystrokes that are needed for specific games. These features could be achieved through the use of Application Programming Interfaces (APIs).

- **Typists**

  Although there are few features requires by typists that are not already present on a conventional keyboard, typists often use layouts such as Dvorak that differ from that of a conventional keyboard.

- **Programmers**

  Similarly to typists, programmers do not typically use any peripherals that are not typical of a desktop computer. Despite this, programmers frequently use hotkeys to carry out functions such as copying text. The ability to map these functions to dedicated key, rather than combinations could be a significant benefit.

I plan to use my friend Callum as a representative of these groups as he is both a gamer and programmer.

## 1.3 - Research

### 1.3.1 Corsair K95 RGB Platinum

URL:
https://www.corsair.com/eu/en/Color/Keyboard-Model/Key-Switches/Keyboard-Layout/k95-rgb-platinum-config-na/p/CH-9127014-ND
Date accessed: 12/6/18

The Corsair K95 RGB Platinum is a mechanical keyboard that allows the user to change the function of any key through the supplied software tools. Every key of the K95 contains a Light Emitting Diode (LED) that can display a variety of colours, which can be calibrated into varying profiles using corsair's iCUE software. The keyboard can be calibrated so that specific keys cause the lighting profile to change. The K95 was created primarily for gamers and therefore prioritises ergonomics over customisability, such as the inclusion of a reversible wrist rest.


**Figure 1.3.1.1 - Annotated Image of K95 Keyboard**

| Features that may be applicable for my project | Explanation | Features that are unlikely to apply to my project | Explanation |
|---|---|---|---|
| Customisable lighting | Many of the features implemented by corsair for customising the lighting could be applied to my project as optional features of the interface. | Physical keys | I intend to use software in combination with a touch screen to to create a virtual keyboard, rather than physical keys. |
| The ability to change the input of each key | The ability to customise the functionality is the focus of my project, making the ability to alter the input of each key is an essential feature for my project. | Textured keys | As I will not be implementing physical keys, It is unlikely to be realistically possible to implement textured keys. |
| Dedicated macro keys | The ability for the user to create dedicated keys for macros (macro instructions) is a feature that will increase the number of ways the | LED strip | While I could implement an LED strip into my project, it would add time and cost to the |

| | | | |
|---|---|---|---|
| | user can customise the functionality of the device. | | development of the keyboard, without adding any improvements to functionality or customizability. |
| Software tools | The K95 comes with a software tool known as “iCUE”, which allows the user to apply more detailed customisation to the device, this is a feature that I will likely also implement, as an additional way for the user to customize the keyboard. | Variety of Keyswitches | While the variety of keyswitches is a useful feature of the K95, the nature of my keyboard prevents the necessity for key switches altogether. |
| Dedicated media keys | While I do not plan to add dedicated media keys to the default layout, it makes sense to include optional dedicated media keys as a feature. | | |
| Dedicated wrist rest | Although the focus of my project is not the ergonomics of the device, it’s improvement would greatly aid the usability of my device. For this reason I may consider implementing a wrist rest in my device | | |

### 1.3.2 Project Precog

URL: https://www.asus.com/Project-Precog/
Date Accessed: 12/6/18

Project Precog is a device that is currently in development (as of June 2018). The device will use artificial intelligence (AI) to create a dual screen laptop that can fill a variety of purposes, such as a tablet, photo editing device or a conventional laptop. While the project focuses on the use of AI, the instillation of a second touch screen rather than a second keyboard makes the device proposed by Project precog similar in design to my project.

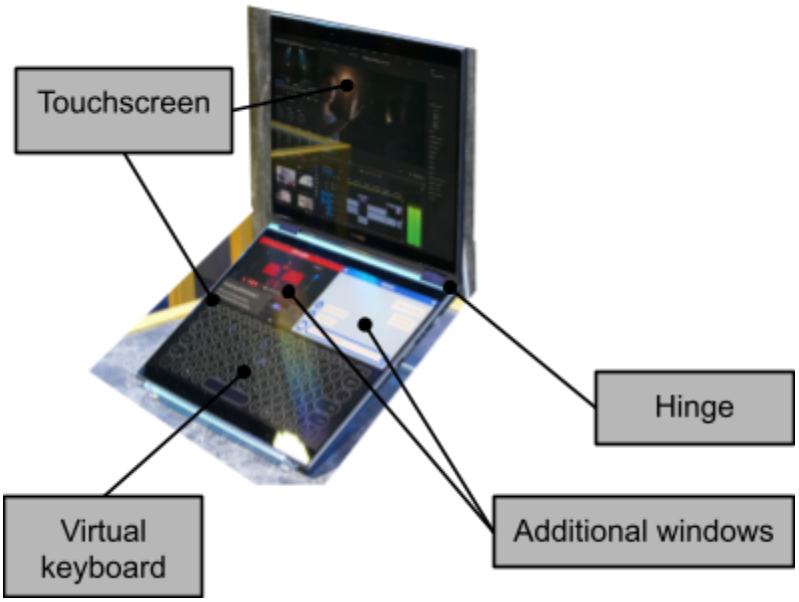


**Figure 1.3.2.1 - Annotated image of a prototype of Project Precog**

| Features that may be applicable for my project | Explanation | Features that are unlikely to apply to my project | Explanation |
|---|---|---|---|
| Use of touch screen to create a keyboard | This is the method I intend to use to power my keyboard | Collapsible design | My device is intended as a desktop peripheral rather than a portable device, making this feature less applicable to my project |
| Capacity to use as pen and touch device | This is an interesting feature proposed by ASUS, and is a feature I have the possibility of including | Two touch screens | As my device is intended to serve as a keyboard replacement, rather than a stand alone device, there is |

| | | | |
|---|---|---|---|
| | | | little need for multiple touch screens. |
| Application of AI | While most of the applications of AI proposed by project precog are irrelevant to my keyboard, the capacity for AI to be used for text prediction could potentially be a useful feature. | | |

### 1.3.3 101touch

URL: http://www.101touch.com/
Date Accessed: 12/6/18

The 101touch is a touch screen device that operates by using software to create a virtual keyboard that can be used as if it were physical. The 101touch has a variety of presets and features, each created with a specific demographic in mind. The large number of presents, in addition to a high degree of customizability is intended to create a device that can be used by almost anyone, to complete tasks that would typically require specialist hardware. The 101touch also utilises the APIs of varying applications in order to further increase the scope of it's application, allowing it to include application specific functions.

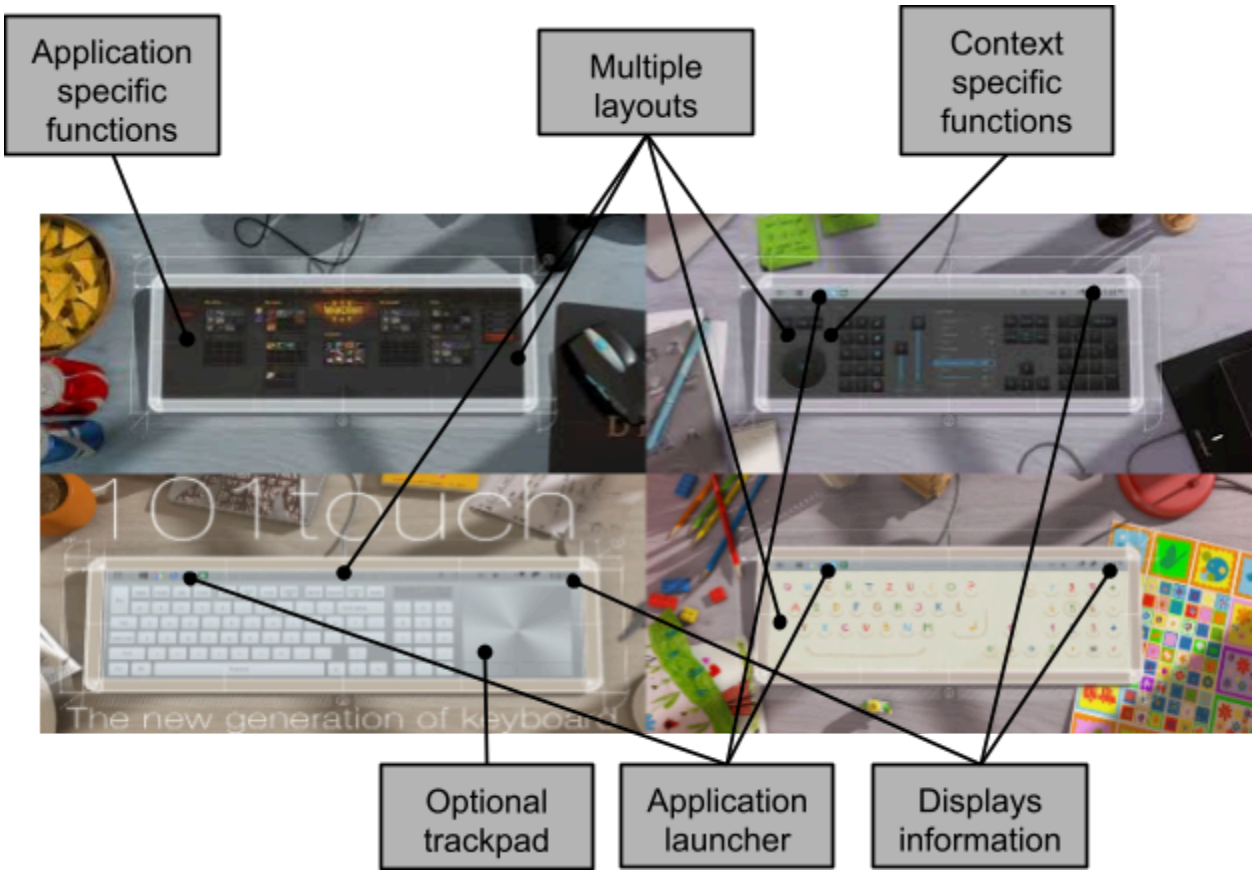


**Figure 1.3.3.1 - Annotated image of multiple presents of the 101touch**

| Features that may be applicable for my project | Explanation | Features that are unlikely to apply to my project | Explanation |
|---|---|---|---|
| Multiple layouts | As my project is intended to fill the role of a variety of specialist equipment, it is important for it to have a variety of layouts to accommodate the number functions. | Android platform | The device uses an android platform to power the device so as to allow for the initial setup of the 101touch to be as fast as possible, differs from my focus on the degree of |

| | | | |
|---|---|---|---|
| | | | customizability. |
| Trackpad | This feature would add further functionality to my device, by eliminating the need for a mouse. | | |
| Use of APIs to create keys with program specific functions | This would allow my device to include application specific functions, increasing the number of possible uses of the device. | | |
| Customisable layout. | The capacity for the user to customise the appearance increases the potential usability of the device by allowing the user to use a layout they find clearer. | | |
| Haptics | The 101touch contains small vibratory engines the provide haptic feedback when typing on the device, while this helps to mitigate the disadvantages of a touch screen over a mechanical keyboard by providing a response from keystrokes, this is a feature I may include in my device. | | |

## 1.4 - Proposed Solution

Based on my research above I have decided to create a peripheral device that uses a touch screen in place of physical keys. This would allow it to serve the function of other devices when necessary.

### 1.4.1 Outline of Solution

A touch screen with on board memory displaying a fully customisable keyboard, that can be plugged into a Universal Serial Bus (USB) port and used as a peripheral keyboard. The device will have the capacity to support functions such as hotkeys and all unicode characters, in addition to an interface through which to customise it.

The keyboard will also have the capacity to serve as a replacement for a variety of specialist peripheral devices such as a MIDI device by simulating their functionality through the use of software.

### 1.4.2 Requirements of Solution

#### 1.4.2.1 Functionality

Essential

| Feature | Justification | Reference |
|---|---|---|
| Unicode support | By supporting the full range of unicode characters, it allows the keyboard to be used in a wider range of contexts, such as for typing in multiple languages. | |
| Touch screen interface | It is essential for me to use a touch screen interface as it will allow me to receive inputs from the user when simulating hardware devices. | Section 1.3.3 |

Additional

| Feature | Justification | Reference |
|---|---|---|
| API support | By providing support for a variety of APIs, the keyboard will be able to perform program specific inputs. | Sections 1.2 and 1.3.3 |
| Software Development Kit (SDK) | By creating an SDK, third parties will be able to create software that is compatible with the device, allowing for additional features. | |

| | | |
|---|---|---|
| MIDI inputs | One device for which a virtual replacement would be made is a MIDI keyboard. | Section 1.2 |
| Macros (Macro instructions) | By allowing the user to create keys that perform a predefined set of instructions, the functionality of the keyboard could be improved. | Section 1.3.1 |

### 1.4.2.2 Usability

Essential

| **Feature** | **Justification** | **Reference** |
|---|---|---|
| GUI | It is essential that I provide a GUI, so that the user has a visual representation of the device being simulated. | Section 1.5.2 |

Additional

| **Feature** | **Justification** | **Reference** |
|---|---|---|
| Customisable keyboard layouts | The usability of the keyboard would be improved by providing the user with the capacity to customise the layout. | Section 1.2 |
| Wrist rest | One issue with many keyboards, is the ergonomics. Although this is not the primary focus of my project, the inclusion of a wrist rest would | Section 1.3.1 |

| | | |
|---|---|---|
| | reduce the strain on the wrists. This would improve usability when the device is used for extended periods of time. | |
| Customisation software | The usability could be improved by creating a piece of software that allows the user to customise the device. This would make the customisation of the device simpler and more accessible to those with less experience. | Section 1.3.1 |

### 1.4.3 Hardware and Software

My project has both a hardware and a software element. This is because my project needs the software to simulate a variety of hardware peripherals and the hardware the allow for physical interaction with them.

Although I am not entirely sure of the hardware I intend to use at this stage of the project, It will be as outlined below. I will clarify the hardware used during the development phase of my project.

My project will require a touchscreen, so that it can display the images and take the inputs required to simulate devices such as keyboards. My project will also require a device to run my software, such as a Raspberry Pi or an Arduino. My project would also benefit from a housing to prevent the device from breaking. My project may also benefit from additional sensors such as a microphone, as it will increase the number of devices that could be simulated by third parties.

My project will require a piece of software that will simulate a keyboard, My project is also likely to require an SDK so as to allow third parties to create software that simulates varying peripheral devices. My project may also require software that allows the user to customise the layout of the device.

## 1.5 - Computational Methods

### 1.5.1 Computational Solution

This problem lends itself to a computational solution as it requires a device that will serve as a means to provide inputs to a desktop computer, this is only achievable through computational methods.

### 1.5.2 Application of Computational Methods

There are many computational methods that I will need to apply to my project in order to be successful.

One computational method that applies to my project is visualizsation. This is because creating a robust Graphical User Interface (GUI) is one of the key requirements of my project.

Another computational method that will play a significant role in my project is abstraction. This is because in order to create a platform by which to customise the device,I must apply abstraction so as to give the impression that a given function or keystroke is a single task, when in reality it may be comprised of a combination of commands.

Reusable program components will also be important for my project, as the device will likely have to run on relatively low powered hardware. This means it is important for me to maximise the efficiency of any software I create and reusing program components will help me achieve this.

### 1.5.3 Success criteria

I will use the features specified in section 1.4.2 as the criteria by which I will measure the success of my project. I will ensure all features listed as "essential" are included and will implement as many other features as possible. In addition to this, there are three essential factors that must be met when constructing the hardware for my device.

- The device must be able to serve as a peripheral device
    - This means it must be capable of sending a signal to a computer over a USB connection
- The device must appear to be a single piece of Hardware from it's exterior
    - This means any additional hardware must be contained in the same case.
- The device must operate without the end user needing to remove the device from its case
    - This means ensuring all necessary features, such a power switch, it accessible from the exterior

### 1.5.4 Limitations

One significant limitation is the lack of specialist hardware. The degree to which the devices can be simulated is greatly inhibited by the general purpose hardware used in my project.

Another limitation is security, particularly in regards to keyloggers. By providing software tools for my device, I am increasing the number of methods that could be used by malware to exploit my device. Keyloggers in particular are a risk for my project, as the features of my device it particularly vulnerable to keylogging. Keyloggers are hackers that use spyware to detect the keyboard inputs made to a computer in order to obtain sensitive data such as passwords.

# 2 - Design Outline

## 2.1 - Break down of solution

I have decided to use a hierarchical diagram to illustrate the decomposition of my solution. This is due to the many aspects of my solution.

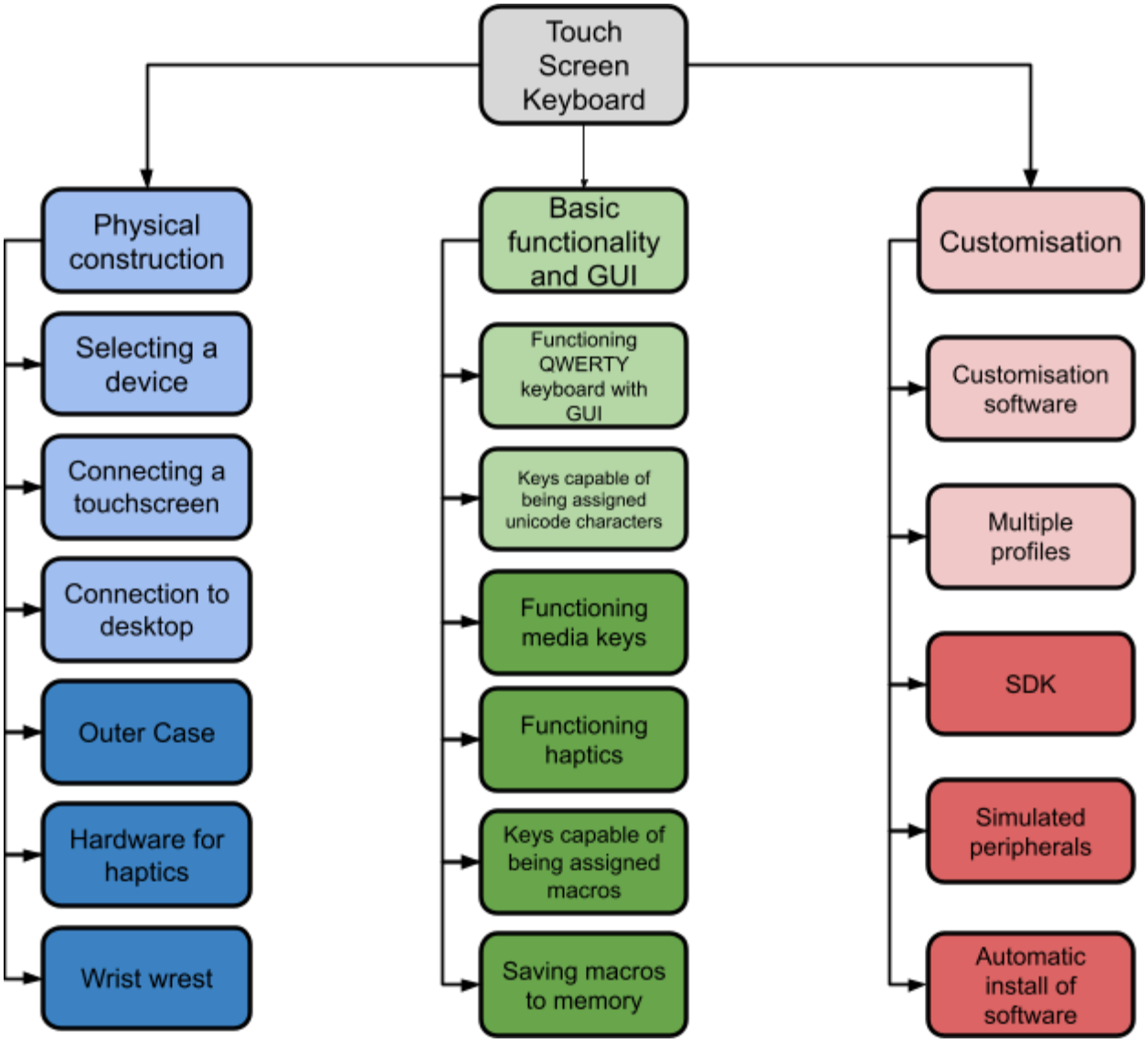


**Figure 2.1.1 - Hierarchical diagram of my solution**

Figure 2.1.1 is an illustration of the three cycles of my solution. The first cycle is depicted in blue, the second in green and the third in red. Each node is an aspect of the problem I intend to tackle in each cycle of my project. The nodes at the bottom are depicted in a darker shade, this indicates that these are optional features that I may add if I have the time and resources.

The first cycle will focus on the physical construction of the device. This involves selecting the hardware to use in my device and the act of assembling them so as to make the remainder of my project possible.

The second cycle will focus on implementing the basic functionality and GUI of my device. This involves getting the device to function as a simple QWERTY keyboard, with the addition of extra key capable of being assigned Unicode inputs. If possible I will also implement additional features such as media keys and functioning haptics.

The third cycle will focus on the customisation software and other tools that will support my device. This involves the creation of software tools to allow for the creation of different keyboard layouts, as well as the design of additional preset keyboards. If possible I will also create additional features such as an SDK and the automatic install of any drivers required by the device.

Due to the hardware centered nature of the first cycle, the objectives I aim to complete differ from the other two cycles. Because of this, I will specify what I aim to achieve during cycle 1 in section 3.0. For each of the other two cycles, I aim to complete:

- Cycle outline- in which I intend to produce a brief overview of my intentions for the cycle, in addition to defining the success criteria for its completion.
- Algorithms definition - in which I will define the algorithms which I intend to use in the cycle
- Data structures - in which I intend to specify the main data structures used in the cycle with example data and justification as to why they are appropriate.
- Testing - in which I intend to define varying tests to confirm the functionality of the software produced in the cycle, as well as any changes made to rectify these
- Evaluation - in which I intend to evaluate the cycle according to the criteria listed in the cycle's outline in addition to specifying any changes to the final design made during the cycle's completion.

# 3 - Cycle 1: Physical construction

## 3.0 - Aims for Cycle 1

In this section I intend to complete:

- Cycle outline - in which I intend to produce a brief overview of my intentions for the cycle, in addition to defining the success criteria for its completion.
- Intended approach - in which I intend to specify how I anticipate completing the cycle
- Cycle notes - in which I intend to document the progress of completing the cycle
- Testing - in which I intend to specify any tests which can be performed at this stage along with any findings and any modifications made in response
- Evaluation - in which I intend to evaluate the cycle according to the criteria listed in the cycle's outline in addition to specifying any changes to the final design made during the cycle's completion.

## 3.1 - Cycle 1 outline

### 3.1.1 Intentions for Cycle 1

In this cycle I will focus on the physical construction of the device. This involves getting the device to a stage where it has the hardware to facilitate the software created in later cycles and could theoretically function as intended if the correct software were present.

### 3.1.2 Success Criteria

The criteria I aim to complete in this cycle are as follows:

- The device must have the capacity to send graphics to the touch screen.
- The device must have the capacity to receive data from the touch screen.
- The device must be able to send ASCII (American Standard Code for Information Interchange) characters to another device.
- The device must have the capacity to send other data to another device.

Where possible I will also attempt to satisfy the following criteria:

- The device should have an outer case that protects the inner hardware and obscures them from the end user.
- The device should have a connection to hardware capable of providing haptic feedback.
- The device should have a wrist rest to improve the usability of the device when used as a keyboard.

## 3.2 - Intended approach

Due to the nature of my first cycle, I am unable to full plan out my process. The diagram below is an illustration of the two stage process that will serve as a guideline for the cycle.

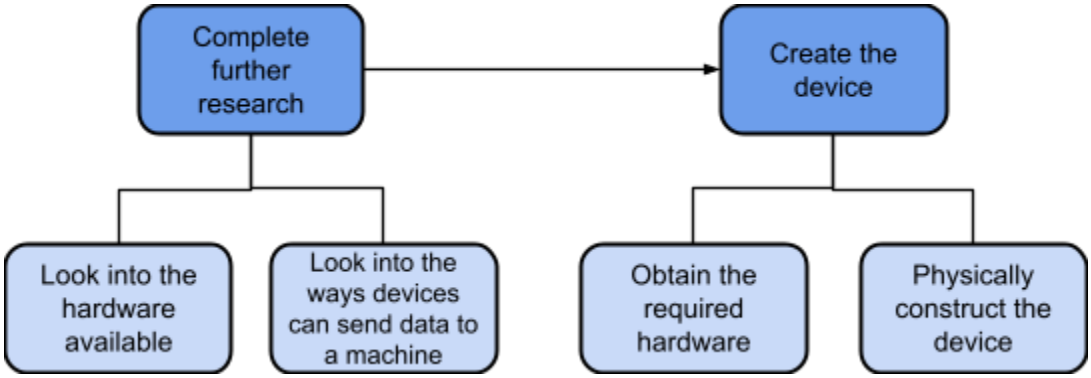


**Figure 3.2.1 - diagram of approach to cycle 1**

## 3.3 - Cycle notes

### 3.3.1 Additional research

When looking further into the hardware available it became clear that I should use a Raspberry Pi ZERO to power my device, this is not ideal as although there are many touchscreens available for the device, I was unable to find one large to function as I had envisioned. This also lead to an important decision as to my approach to sending inputs. This was the decision between the use of two technologies to send inputs, USB and bluetooth.

#### 3.3.1.1 USB

The USB standard would not normally allow for the type of connection needed for my project as it does not allow for my device to act as both host and slave.The OTG (On The Go) variant of the standard resolves this issue allowing it to be used to connect my device to other machines.

| Pros | Cons |
| --- | --- |
| The device likely be more secure as there is no risk of the packets being intercepted. | The OTG standard is a wired standard, meaning I would be unable to use the device wirelessly. |
| The connection would likely be easier to implement than a bluetooth connection as the device does not need to pair with the intended machine. | The device would need a cable that specifically supports the OTG standard. |
| | The OTG cable would require the use the Pi's only USB port not needed to power the Pi, which may limit number of additional features I am able to implement . |

#### 3.3.1.2 Bluetooth

The bluetooth standard is a standard that uses wireless connections to build PANs (Personal Area Networks). The bluetooth standard has many variants known as profiles. I believe the HID (Human Interface Device) profile is most appropriate for my project as it is designed to power peripheral devices such as mice, keyboard and game controllers.

| Pros | Cons |
| --- | --- |
| A bluetooth connection would allow the | The device would be less secure as packets |

| | |
|---|---|
| device to be used wirelessly, improving usability. | could be intercepted. Encryption would not resolve this issue as the device would still be susceptible to man-in-the-middle attacks . |
| A bluetooth connection is less likely to require the use of the Pi's only free USB port, allowing it to be used to power a better touchscreen or to support additional features. | The device would need additional hardware to allow it to transmit bluetooth packets. |
| | The connection would likely be more challenging to implement than a USB connection as it may be difficult to allow the device to differentiate between bluetooth devices. This also creates the problem of managing multiple paired devices. |
| | The device would only be compatible with machines capable of a bluetooth connection, impairing the practicality of my device. |

#### 3.3.1.3 Raspberry Pi ZERO vs Raspberry Pi ZERO W

After looking into the pros and cons of both bluetooth and USB connections, I decided to begin by implementing a USB connection. This was primarily due to the benefits to security and the lesser complexity of its implementation. This meant I was able to focus on implementing the device over USB, while leaving the possibility of implementing a bluetooth connection in the future, allowing the device to connect over both mediums.

The decision to begin by implementing USB but to maintain the possibility of implementing both connections in the future left me with a second decision. This was the decision between the two variants of the Raspberry Pi, the original "Raspberry Pi ZERO" or the newer "Raspberry Pi ZERO W". The only difference between the devices besides the price was that the ZERO W has on board bluetooth and wireless capabilities. Although the ZERO W's additional features first appeared to be strictly beneficial, the additional connection may serve as an additional vector for the device to be hacked.

| Benefits of Pi ZERO | Benefits of Pi ZERO W |
|---|---|
| The device would be more secure as it would have fewer attack vectors for hackers to use. | I would be able to use wireless peripherals when developing the device, making development easier |
| The Pi ZERO is available at a lower price, making it cheaper to replace if broken. | The device would have the hardware to transmit bluetooth packets, making it easier to implement a bluetooth connection in the future. |
| | The device would have access to the internet, which may allow my device to support additional features in the future |

Source https://www.modmypi.com/blog/raspberry-pi-comparison-table
Date accessed: 10/10/18

After comparing the Raspberry Pi ZERO to the Raspberry Pi ZERO W, I decided that the potential for developing further functionality provided by the Pi ZERO W justified the additional security consideration.

### 3.3.1.4 Selecting a touchscreen

The final decision I had to make was to select a touchscreen for use in my project. As stated at the beginning of section 3.3.1, the Raspberry Pi ZERO only has support for a limited number of touchscreens. This is in part due to it having less powerful hardware when compared to other models in of Raspberry Pi. The Raspberry Pi ZERO also lacks a DSI (Display Serial Interface) port which is used to power many of the touchscreens designed for the Raspberry Pi, such as the original first party 7” touchscreen. The number of usable touchscreens is further diminished by the need to use the Raspberry Pi ZERO’s only spare USB port to connect to send inputs, as explained in section 3.3.1.1. This leaves only the Raspberry Pi’s GPIO (General Purpose Input/Output) pins.

The Raspberry Pi's GPIO pins support the SPI (Serial Peripheral Interface) protocol, allowing them to be used to detect inputs made on the touchscreen. The largest touch screens readily available that does not require a USB or DSI connection use the Raspberry Pi's GPIO pins in conjunction with the Pi's  Mini HDMI (High-Definition Multimedia Interface) port. These touch screens are only 5 inches, which greatly limits the functionality of the device. These screens would be large enough to support a small keyboard, macro keys, MIDI input device and more, but not without major compromises as to their size.

### 3.3.2 Construction

Once I had decided on the hardware I was going to use, I was left with the task of physically constructing the device and configuring the software to send ASCII characters for testing.

#### 3.3.2.1 Initial construction

I started by hooking it up to a monitor over HDMI via a VGA (Video Graphics Array) adapter and used a USB to microUSB OTG converter to connect a wireless keyboard and mouse. This allowed me to verify that the Pi was working and gave me a way to interact with the operating system. This allowed me to install the drivers required by the touchscreen

I then followed the guide created by rmed to allow the pi to send data to other machines. After completing the first of the three sections of the guide, I had managed to run the bash code written by rmed , defining a keyboard called "mykeyboard" in within ConfigFS. I achieved this by editing the rc.local file of my raspberry pi.

https://www.rmedgar.com/blog/using-rpi-zero-as-keyboard-setup-and-device-definition
Date accessed (6/11/18)

### 3.3.2.2 Change to serial connection

Although I was able to successfully follow the first section of rmed's guide, it had taken significantly longer than I had anticipated, and it would not be practical to continue. For this reason I decided to attempt to use a serial connection to send signals to a raspberry pi 3B+ instead of a USB OTG connection to a desktop machine. This would allow me to begin to develop the software for the keyboard, whilst allowing me to implement a USB OTG connection to other devices in the future.

### 3.3.2.3 Use of BBC Microbit

Due to an unforeseen limitation of the hardware, it was not practical for me to establish a serial connection between the two raspberry pis. Because of this, I moved all of the data from the pi zero onto a second raspberry pi 3B+ and connected the touchscreen. I then connected a BBC Microbit to each pi 3B+. This allows me to set up a serial connection between each pi and it's connected microbit. I can then transfer the data wirelessly between the two microbits.

### 3.3.2.4 Use of Virtual Network Computing (VNC) software

After connecting the devices as stated in section 3.3.2.3, I used the micropython library to write a small program to allow the microbit connected to the first pi to send any data it receives from the pi to the other microbit over wireless. I then created another program to allow the microbit to send any data it receives wirelessly to the connected pi.

This did not allow the data to be sent, as the programs had to be timed so accurately as to be unrealistic. Because of this, I planned to utilise VNC software to send keyboard inputs. This would allow the device to send data to the other pi wirelessly, without the need for any serial connection. It also has the benefit that the device could be configured to connect to other machines.

I began by following the guide on the raspberry pi website and was able to successfully enable VNC server, but was unable to install VNC viewer on the other pi as the website used to download the application was blocked on the school internet. Due to time constraints, I was unable to continue with the VNC implementation.

https://www.raspberrypi.org/documentation/remote-access/vnc/
date accessed (24/4/19)

I will attempt to implement a system using VNC at a later date.

## 3.4 - Testing

Due to the nature of my project at this stage, there are only a limited number of meaningful tests that can be performed. The number of meaningful tests are further inhibited as the device is not currently able to send data. For this reason, tests 1.1.1 to 1.1.4 will focus on the functionality that is present on the device.

| Test ID | Test | Reasoning | Expected Outcome | Actual Outcome | Evidence |
|---|---|---|---|---|---|
| 1.1.1 | Turn on the raspberry pi without any monitor connected | To verify that the raspberry pi is able to power on | The red LED on the pi will turn light up, indicating the pi has powered on | Pass - The red LED lit up, indicating that the pi was powered on | Photo Evidence 1.1 |
| 1.1.2 | Connect a monitor to the raspberry pi | To verify that the pi is able to display graphics via its HDMI port and that the Operating System (OS) has been installed | The monitor will display the desktop | Pass - The monitor displayed the desktop | Photo Evidence 1.2 |

| | | | | | |
|---|---|---|---|---|---|
| | | correctly | | | |
| 1.1.3 | Connect the raspberry pi to the touchscreen | To verify that raspberry pi is able to send graphics to the touch screen | The touch screen will display the desktop | Pass - The touch screen displayed the desktop | Photo Evidence 1.3 |
| 1.1.4 | Tap the touchscreen | To verify that the raspberry pi is receiving inputs from the touchscreen | The cursor will move to the appropriate location | Pass - The cursor moved to the location where it was pressed | Video Evidence 1.1 |

## 3.5 - Evaluation

### 3.5.1 Evaluation of success criteria

Criteria I had aimed to complete:

| Criteria | Evaluation | Explanation | Response |
|---|---|---|---|
| The device must have the capacity to send graphics to the touch screen. | Pass | The device has a HDMI connection to the touch screen, allowing it to use it as a display. | This criteria has been met. |
| The device must have the capacity to receive data from the touch screen. | Pass | The device is able to receive all necessary data from the touchscreen, allowing it to detect touch. | This criteria has been met. |
| The device must be able to send ASCII (American Standard Code for Information Interchange) characters to another device. | Fail | The device is currently unable to transfer any meaningful data to other devices. | In the future, I plan to utilise VNC software to allow for the transfer of ASCII characters. |
| The device must have the capacity to send other data to another device. | Fail | The device does not currently have the capacity to transfer any meaningful data to other devices. | In the future, I plan to look into the possibility of using VNC software to transfer other data. |

Criteria I also attempted to satisfy where possible:

| Criteria | Evaluation | Explanation | Response |
|---|---|---|---|
| The device should have an outer case that protects the inner hardware and obscures them from the end user. | Fail | The case that shipped with the raspberry pi cannot be used with the touchscreen and I have not had time to look into alternatives. | In the future, I plan to utilise 3D printing to manufacture a case that is unique to my device. |
| The device should have a connection to hardware capable of providing haptic feedback. | Fail | I have not had time to source hardware capable to providing haptic feedback. The fact that the device does not have a case also means implementing haptic feedback may damage the other hardware components. | In the future, I plan to purchase and connect a simple motor, allowing of the device to vibrate. |
| The device should have a wrist rest to improve the usability of the device when used as a keyboard. | Fail | The device does not currently have a case, making it difficult to attach a wrist rest. The touchscreen is also much smaller than initially planned, meaning a wrist rest would have little effect on the usability of the device. | In the future, I plan to incorporate a small wrist rest into the 3D printed case |

# 4 - Cycle 2: Basic Functionality and GUI

## 4.1 - Cycle 2 Outline

### 4.1.1 intentions for Cycle 2

In this cycle I will focus on creating the software required for basic operation as a keyboard, in addition to creating a GUI to accompany this. I will also implement a key that with the ability to send unicode characters. If possible I will also implement additional functionality to this keyboard such as macros and media keys.

### 4.1.2 Success Criteria

The criteria I aim to complete in this cycle are as follows:

- Create the software required for a basic ASCII keyboard
- Create a GUI for a basic QWERTY keyboard
- Create a button capable of inputting unicode characters

Where possible I will also attempt to satisfy the following criteria:

- Implement functioning media keys to the keyboard
- Create keys capable of being used as macros
- Implement a system to save macros to memory

## 4.2 - GUI design

In order to effectively implement a GUI, it is important to first plan the layout. The appearance of the GUI is also important for my project as its design will greatly impact the user experience. Because of this, I have decided to create a draft of the design using the image editor paint.net, before implementing the GUI. I used an image of the standard UK windows keyboard layout from https://en.wikipedia.org/wiki/British_and_American_keyboards, shown in Figure 4.2.0.1 as a template for my GUI.


**Figure 4.2.0.1 - Template of the standard UK keyboard layout**

### 4.2.1 First Draft of GUI Design

The touch screen display I am using for the device only has a resolution of 800 by 480 pixels. The template I used had a size of 400 by 133 pixels. In order to remedy this, I enlarged the layout and stretched it vertically, giving a size of 800 by 321 pixels, providing a compromise between the size of the keys and the degree they have to be stretched vertically. This also left enough space above the keyboard to allow for some of the additional features planned, such as media and macro keys. I also replaced the “Alt Gr” key with a unicode key, so as to meet the success criteria states in section 4.1.2. I then removed the secondary labels, as these are no longer needed. Finally I created some media keys and changed some of the labelling to match the aesthetic of the media keys I had created.

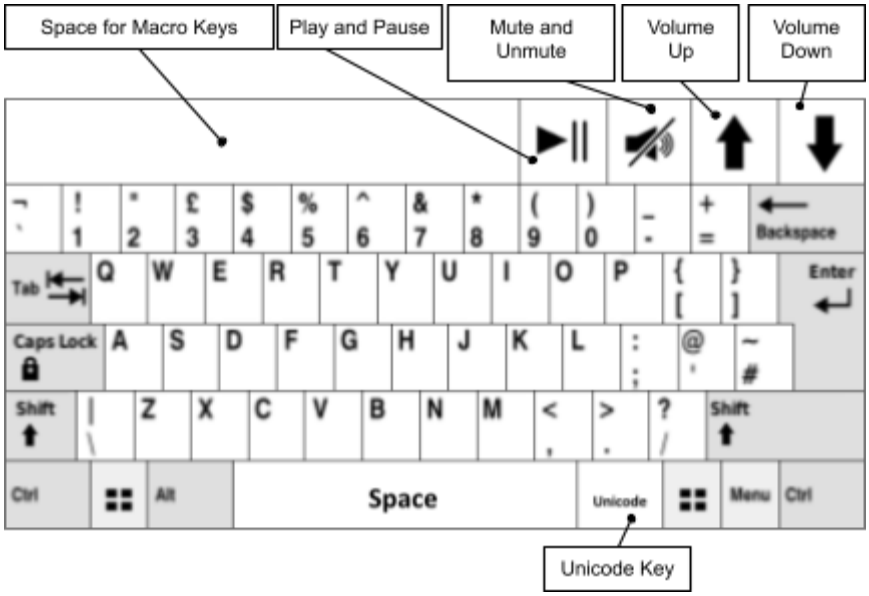


**Figure 4.2.2 - First draft of GUI design**

I decided not to include the macro keys in my first draft as I am unsure whether I will have time to implement them. I also decided to keep the colour scheme largely the same as the template, as the black and white character keys are easy to read and the grey backgrounds on keys such as tab and enter help to differentiate them without deviating too far from the aesthetic of the remainder of the keyboard. This also meant is would be easier create additional colour schemes in the future, as I could use different shades and saturations of the same colour to create a similar layout in a separate colour.

I chose a QWERTY layout as this is the most common keyboard layout. Although it is a relatively inefficient layout, it is by far the most widely used and will likely be the most familiar to the user. I chose a compact layout as opposed to a larger layout such as a  full-size or tenkeyless layout. This is due to the lack of space provided by the touchscreen, which would have made these layouts require impractically small keys. Alternative layouts are also generally much wider while only being slightly taller. This is problematic as it means implementing them would require the keys to be further stretched vertically, which could make them difficult to use.

I decided to make the row containing the additional features larger than the other rows. This was partially done to allow me space to implement any further additional features in cycles 2 and 3. This was also done to utilise the remaining space above the main keyboard. I made the media keys roughly square as this makes them large and therefore easy for the user to identify. This is important as they are not typically present on keyboards of this size, so the user may be initially unaware of them. I also made the media keys slightly wider as this makes them better aligned with the rest of the keyboard.

## 4.3 - Algorithm definition

Once I had constructed an initial draft of the GUI, I began to break down how I would implement my GUI. Below is a pseudocode algorithm for the keyboard.

```
import tkinter

for button in buttons:
      button.load(button.position)
end for

while true:
      for button in buttons:
            if button.isDown:
                  if shiftKey.isDown:
                        send(button.shiftValue)
                  else if capsKey.isDown:
                        send(button.capsValue)
                  else:
                        send(button.lowerValue)
                  end if
            end if
      end for
end while
```

**Figure 4.3.0.1 - pseudocoded algorithm for my GUI**

The first line of code imports the tkinter library. This library will allow me to more easily implement my GUI.

```
import tkinter
```

The next section of my algorithm is a for loop that repeats for every item in the array buttons, which contains all buttons that should be placed onto the interface. The program will store the

current item using the variable button. Each button is loaded using the method button.load(). The x and y values of the location where the button should be placed are stored in the array button.position. This array is passed into the method button.load() using the parameter position.

```
for button in buttons:
      button.load(button.position)
end for
```

The final section of my algorithm is a while loop that will repeat the same for loop indefinitely. This for loop will repeat for each item in the array buttons, storing each item using the variable button. The for loop will check the value boolean attribute button.isDown, if the value of this attribute is true, this will indicate that the button is being pressed. When a button is pressed, the program will check whether the shift or caps keys are being pressed, using shiftKey.isDown and capsKey.isDown respectively. The appropriate value of the key is will be passed into the subroutine send(value), using either button.shiftValue, button.capsValue or button.lowerValue. This subroutine will send the value passed into it to the other device.

```
while true:
      for button in buttons:
            if button.isDown:
                  if shiftKey.isDown:
                        send(button.shiftValue)
                  else if capsKey.isDown:
                        send(button.capsValue)
                  else:
                        send(button.lowerValue)
                  end if
            end if
      end for
end while
```

## 4.4 - Data Structures

| Name | Data Structure | Contents |
|---|---|---|
| buttons | Array | All keys that will be included within the keyboard, implemented as objects using the tkinter library |
| button.position | Array | Two integers, representing the x and y values at which the button should be placed |
| button.isDown | Boolean | Will be set to true when the button is pressed and set to false when the button is released |
| button.shiftValue | Character | Contains the character that should be entered when the button is pressed whilst the shift key is held down |
| button.capsValue | Character | Contains the character that should be entered when the button is pressed whilst the caps key is held down but the shift key is not |
| button.lowerValue | Character | Contains the character that should be entered when the button is pressed and both the shift and caps keys are not held down |

## 4.5 - Testing

### 4.5.1 Test Plan

Below is are the tests I will use to verify that the interface is functional. Due to the nature of the cycle, no erroneous or boundary tests are needed as all data will be input using the buttons on the interface.

| Test ID | Test | Reasoning | Expected Outcome |
|---|---|---|---|
| 2.1.1 | Run the program on a separate machine | To check the program is able to correctly load the GUI | A small window to appear containing a keyboard interface |
| 2.1.2 | Run the program on the pi | To check that the GUI will load at the correct size for the touchscreen | A window containing the GUI should appear, with the whole of the window visible but covering as much of the screen as possible |
| 2.1.3 | Using a mouse connected to the pi, click the "A", key | To check that the program registers the "A" key is pressed | The "A" key should temporarily change colour whilst held down and should send the character "a" when the key is pressed |
| 2.1.4 | Using the touchscreen, press the "A" key | To check that the program is registering touch screen inputs correctly | The "A" key should temporarily change colour whilst held down and should send the character "a" when the key is pressed |
| 2.1.5 | Press the "4" key | To check that the program can correctly input numerical values | The "4" key should temporarily change colour whilst held down and should send the character "4" when the key is pressed |
| 2.1.6 | Press the "#" key | To check that the program can correctly input values other than letters and numbers | The "#" key should temporarily change colour whilst held down and should send the character "#" when the key is pressed |
| 2.1.7 | Press the "Q" key | To check that the program registers the "Q" key is | The "Q" key should temporarily change colour whilst held down and should |

| | | | |
|---|---|---|---|
| | | pressed | send the character "q" when the key is pressed |
| 2.1.8 | Press the "F" key | To check that the program registers the "F" key is pressed | The "F" key should temporarily change colour whilst held down and should send the character "f" when the key is pressed |
| 2.1.9 | Press the "J" key | To check that the program registers the "J" key is pressed | The "J" key should temporarily change colour whilst held down and should send the character "j" when the key is pressed |
| 2.1.10 | Press the "2" key | To check that the program registers the "2" key is pressed | The "2" key should temporarily change colour whilst held down and should send the character "2" when the key is pressed |
| 2.1.11 | Press the "6" key | To check that the program registers the "6" key is pressed | The "6" key should temporarily change colour whilst held down and should send the character "6" when the key is pressed |
| 2.1.12 | Press the "Tab" key | To check that the program registers the "Tab" key is pressed | The "Tab" key should temporarily change colour whilst held down and should send the value "Tab" when the key is pressed |
| 2.1.13 | Press the "Space" key | To check that the program registers the "Space" key is pressed | The "Space" key should temporarily change colour whilst held down and should send the character " " when the key is pressed |
| 2.1.14 | Press the "=" key | To check that the program registers the "=" key is pressed | The "=" key should temporarily change colour whilst held down and should send the character "=" when the key is pressed |
| 2.1.15 | Press the "Mute" key | To check that the program registers the "Mute" key is pressed | The "Mute" key should temporarily change colour whilst held down and should send the value "Mute" when the key is pressed |

| | | | |
|---|---|---|---|
| 2.1.16 | Press the "Play Pause" key | To check that the program registers the "Play Pause" key is pressed | The "Play Pause" key should temporarily change colour whilst held down and should send the value "PlayPause" when the key is pressed |
| 2.1.17 | Press the "Volume Up" key | To check that the program registers the "Volume Up" key is pressed | The "Volume Up" key should temporarily change colour whilst held down and should send the value "VolUp" when the key is pressed |
| 2.1.18 | Press the "Volume Down" key | To check that the program registers the "Volume Down" key is pressed | The "Volume Down" key should temporarily change colour whilst held down and should send the value "VolDown" when the key is pressed |
| 2.2.1 | Press and hold the "Caps Lock" Key | To check that the program can identify when the "Caps Lock" key is pressed | The "Caps Lock" key should change colour when held down and should return to its whilst colour when released. The program should also send the value "Caps Lock" when pressed |
| 2.2.2 | Press and hold the "Shift" Key | To check that the program can identify when the "Shift" key is pressed | The "Shift" key should change colour when held down and should return to its whilst colour when released. The program should also send the value "Shift" when pressed |
| 2.2.3 | Press the "A" key whilst holding the "Caps Lock" key | To check that the program will send capitalised letter characters when "Caps Lock" is held down | The "A" an "Caps Lock" keys should change colour whilst held down. The program should also send the value "A" when the "A" key is pressed |
| 2.2.4 | Press the "4" key whilst holding the "Shift" key | To check that the program will send the appropriate characters when "Shift" is held down | The "4" and "Shift" keys should change colour whilst held down. The program should also send the value "$" when the "4" key is pressed |
| 2.2.5 | Press the "A" key | To check that the | The "A" an "Shift" keys |

| | | | |
|---|---|---|---|
| | whilst holding the "Shift" key | program will send capitalised letter characters when the "Shift" key is pressed | should change colour whilst held down. The program should also send the value "A" when the "A" key is pressed |
| 2.2.6 | Press the "4" key whilst holding the "Caps Lock" key | To check that pressing the "Caps Lock" key will not cause the program to send the shift values for keys where this is not appropriate | The "4" and "Caps Lock" keys should change colour whilst held down. The program should also send the value "4" when the "4" key is pressed |

### 4.5.2 Changes to Testing

Due to my implementation, I had to change the way my program handles modifier keys, such as "Caps Lock" and "Shift". Tests 2.2.1 to 2.2.6 have been replaced by the following tests

| | | | |
|---|---|---|---|
| 2.3.1 | Press the "Caps Lock" Key | To check that the program can identify when the "Caps Lock" key is pressed and to check that the program can identify that "Caps Lock" is currently disabled | The "Caps Lock" key should change colour whilst held down. The program should also output "CapsOn" to the console when the key is pressed. |
| 2.3.2 | Press the "Caps Lock" Key a second time | To check that the program can enable caps lock and can identify that "Caps Lock" is currently enabled | The "Caps Lock" key should change colour whilst held down. The program should also output "CapsOff" to the console when the key is pressed. |
| 2.3.3 | After test 2.3.3, press the "Caps Lock" Key a third time | To check that the program can disable "Caps Lock" | The "Caps Lock" key should change colour whilst held down. The program should also output "CapsOn" to the console when the key is pressed. |
| 2.3.4 | Press the "Shift" Key | To check that the program can | The "Shift" key should change colour whilst held |

| | | | |
|---|---|---|---|
| | | identify when the "Shift" key is pressed and to check that the program can identify that "Shift" is currently disabled | down. The program should also output "ShiftOn" to the console when the key is pressed. |
| 2.3.5 | After test 2.3.4, press the "Shift" Key a second time | To check that the program can enable caps lock and can identify that "Shift" is currently enabled | The "Shift" key should change colour whilst held down. The program should also output "ShiftOff" to the console when the key is pressed. |
| 2.3.6 | After test 2.3.5, press the "Shift" Key a third time | To check that the program can disable "Shift" | The "Shift" key should change colour whilst held down. The program should also output "ShiftOn" to the console when the key is pressed. |
| 2.3.7 | Press the "Ctrl" Key | To check that the program can identify when the "Ctrl" key is pressed and to check that the program can identify that "Ctrl" is currently disabled | The "Ctrl" key should change colour whilst held down. The program should also output "CtrlOn" to the console when the key is pressed. |
| 2.3.8 | After test 2.3.7, press the "Ctrl" Key a second time | To check that the program can enable caps lock and can identify that "Ctrl" is currently enabled | The "Ctrl" key should change colour whilst held down. The program should also output "CtrlOff" to the console when the key is pressed. |
| 2.3.9 | After test 2.3.8, press the "Ctrl" Key a third time | To check that the program can disable "Ctrl" | The "Ctrl" key should change colour whilst held down. The program should also output "CtrlOn" to the console when the key is pressed. |
| 2.3.10 | Press the "Alt" Key | To check that the program can identify when the "Alt" key is pressed | The "Alt" key should change colour whilst held down. The program should also output "AltOn" to the console when |

| | | | |
|---|---|---|---|
| | | and to check that the program can identify that "Alt" is currently disabled | the key is pressed. |
| 2.3.11 | After test 2.3.10, press the "Alt" Key a second time | To check that the program can enable caps lock and can identify that "Alt" is currently enabled | The "Alt" key should change colour whilst held down. The program should also output "AltOff" to the console when the key is pressed. |
| 2.3.12 | After test 2.3.11, press the "Alt" Key a third time | To check that the program can disable "Alt" | The "Alt" key should change colour whilst held down. The program should also output "AltOn" to the console when the key is pressed. |
| 2.3.13 | Press the "A" key after pressing the "Caps Lock" key | To check that the program will send capitalised letter characters when "Caps Lock" is enabled | The "A" key should change colour whilst held down. The program should also send the value "A" when the "A" key is pressed |
| 2.3.14 | Press the "4" key after pressing the "Shift" key | To check that the program will send the appropriate characters when "Shift" is enabled | The "4" key should change colour whilst held down. The program should also send the value "$" when the "4" key is pressed |
| 2.3.15 | Press the "A" key after pressing the "Shift" key | To check that the program will send capitalised letter characters when "Shift" is enabled | The "A" key should change colour whilst held down. The program should also send the value "A" when the "A" key is pressed |
| 2.3.16 | Press the "4" key after pressing the "Caps Lock" key | To check that pressing the "Caps Lock" key will not cause the program to send the shift values for keys where this is not appropriate | The "4" key should change colour whilst held down. The program should also send the value "4" when the "4" key is pressed |
| 2.3.17 | Press the "Unicode" | To check that the | The "Unicode" key should |

| | | | |
|---|---|---|---|
| | key 8 times | unicode key is working correctly | change colour whilst held down and the program should cycle through the predefined list of unicode characters, sending each when pressed. eg: "☺", "≠", "∬", "ɔ", "☺", "≠", "∬", "ɔ" |

### 4.5.3 Prototype Testing


**Figure 4.5.3.1 - A screenshot of the GUI window**

| Test ID | Test | Expected Outcome | Actual Outcome | Evidence |
|---|---|---|---|---|
| 2.1.1 | Run the program on a separate machine | A small window to appear containing a keyboard interface | Pass - the program produced a window containing a keyboard interface as shown in figure 4.5.3.1 | Figure 4.5.3.1 |
| 2.1.2 | Run the program on the pi | A window containing the GUI should appear, with the whole of the window visible but covering as much of the screen as possible | Fail - the program loads a window containing the interface but the buttons are not the correct size and not all of the keys are visible | Video Evidence 2.1.mp4 |
| 2.1.3 | Using a mouse connected to the pi, click the "A", key | The "A" key should temporarily change colour whilst held down and should send the character "a" when the key is pressed | Fail - the "A" key changed colour when the cursor was over the key rather than when the key was pressed. The program also only sent the value "a" when the key was released rather than when it was pressed | Video Evidence 2.1.mp4 |
| 2.1.4 | Using the | The "A" key should | Fail - the "A" key changed | Video |

| | | | | |
|---|---|---|---|---|
| | touchscreen, press the "A" key | temporarily change colour whilst held down and should send the character "a" when the key is pressed | color when pressed, but remained this colour until another key was pressed. The program also sent the value "a" when the key was released rather than when it was pressed. | Evidence 2.1.mp4 |
| 2.1.5 | Press the "4" key | The "4" key should temporarily change colour whilst held down and should send the character "4" when the key is pressed | Fail - the "4" key changed colour but the program sent the character "4" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.6 | Press the "#" key | The "#" key should temporarily change colour whilst held down and should send the character "#" when the key is pressed | Fail - the "#" key changed colour but the program sent the character "#" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.7 | Press the "Q" key | The "Q" key should temporarily change colour whilst held down and should send the character "q" when the key is pressed | Fail - the "Q" key changed colour but the program sent the character "q" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.8 | Press the "F" key | The "F" key should temporarily change colour whilst held down and should send the character "f" when the key is pressed | Fail - the "F" key changed colour but the program sent the character "f" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.9 | Press the "J" key | The "J" key should temporarily change colour whilst held down and should send the character "j" when the key is pressed | Fail - the "J" key changed colour but the program sent the character "j" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.10 | Press the "2" key | The "2" key should temporarily change colour whilst held down and should send the character "2" when the key is pressed | Fail - the "2" key changed colour but the program sent the character "2" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.11 | Press the "6" key | The "6" key should temporarily change colour whilst held down and should send the character "6" when | Fail - the "6" key changed colour but the program sent the character "6" when the key was released, rather than | Video Evidence 2.2.mp4 |

| | | | | |
|---|---|---|---|---|
| | | the key is pressed | when it was pressed | |
| 2.1.12 | Press the "Tab" key | The "Tab" key should temporarily change colour whilst held down and should send the value "Tab" when the key is pressed | Fail - the "Tab" key changed colour but the program sent the value "Tab" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.13 | Press the "Space" key | The "Space" key should temporarily change colour whilst held down and should send the character " " when the key is pressed | Fail - the "Space" key changed colour but the program sent the character " " when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.14 | Press the "=" key | The "=" key should temporarily change colour whilst held down and should send the character "=" when the key is pressed | Fail - the "=" key changed colour but the program sent the character "=" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.15 | Press the "Mute" key | The "Mute" key should temporarily change colour whilst held down and should send the value "Mute" when the key is pressed | Fail - the "Mute" key changed colour but the program sent the value "Mute" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.16 | Press the "Play Pause" key | The "Play Pause" key should temporarily change colour whilst held down and should send the value "PlayPause" when the key is pressed | Fail - the "Play Pause" key changed colour but the program sent the value "PlayPause" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.17 | Press the "Volume Up" key | The "Volume Up" key should temporarily change colour whilst held down and should send the value "VolUp" when the key is pressed | Fail - the "Volume Up" key changed colour but the program sent the value "VolUp" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.1.18 | Press the "Volume Down" key | The "Volume Down" key should temporarily change colour whilst held down and should send the value "VolDown" when the key is pressed | Fail - the "Volume Down" key changed colour but the program sent the value "VolDown" when the key was released, rather than when it was pressed | Video Evidence 2.2.mp4 |
| 2.3.1 | Press the "Caps Lock" | The "Caps Lock" key should change colour whilst held | Fail - the "Caps Lock" key changed colour but the | Video Evidence |

| | | | | |
|---|---|---|---|---|
| | Key | down. The program should also output "CapsOn" to the console when the key is pressed. | program sent the value "CapsOn" when the key was released, rather than when it was pressed | 2.3.mp4 |
| 2.3.2 | Press the "Caps Lock" Key a second time | The "Caps Lock" key should change colour whilst held down. The program should also output "CapsOff" to the console when the key is pressed. | Fail - the "Caps Lock" key changed colour but the program sent the value "CapsOn" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.3 | After test 2.3.3, press the "Caps Lock" Key a third time | The "Caps Lock" key should change colour whilst held down. The program should also output "CapsOn" to the console when the key is pressed. | Fail - the "Caps Lock" key changed colour but the program sent the value "CapsOn" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.4 | Press the "Shift" Key | The "Shift" key should change colour whilst held down. The program should also output "ShiftOn" to the console when the key is pressed. | Fail - the "Shift" key changed colour but the program sent the value "ShiftOn" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.5 | After test 2.3.4, press the "Shift" Key a second time | The "Shift" key should change colour whilst held down. The program should also output "ShiftOff" to the console when the key is pressed. | Fail - the "Shift" key changed colour but the program sent the value "ShiftOff" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.6 | After test 2.3.5, press the "Shift" Key a third time | The "Shift" key should change colour whilst held down. The program should also output "ShiftOn" to the console when the key is pressed. | Fail - the "Shift" key changed colour but the program sent the value "ShiftOn" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.7 | Press the "Ctrl" Key | The "Ctrl" key should change colour whilst held down. The program should also output "CtrlOn" to the console when the key is pressed. | Fail - the "Ctrl" key changed colour but the program sent the value "CtrlOn" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.8 | After test 2.3.7, press the "Ctrl" Key a second time | The "Ctrl" key should change colour whilst held down. The program should also output "CtrlOff" to the console when the key is pressed. | Fail - the "Ctrl" key changed colour but the program sent the value "CtrlOff" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |

| | | | | |
|---|---|---|---|---|
| 2.3.9 | After test 2.3.8, press the "Ctrl" Key a third time | The "Ctrl" key should change colour whilst held down. The program should also output "CtrlOn" to the console when the key is pressed. | Fail - the "Ctrl" key changed colour but the program sent the value "CtrlOn" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.10 | Press the "Alt" Key | The "Alt" key should change colour whilst held down. The program should also output "AltOn" to the console when the key is pressed. | Fail - the "Alt" key changed colour but the program sent the value "AltOn" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.11 | After test 2.3.10, press the "Alt" Key a second time | The "Alt" key should change colour whilst held down. The program should also output "AltOff" to the console when the key is pressed. | Fail - the "Alt" key changed colour but the program sent the value "AltOff" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.12 | After test 2.3.11, press the "Alt" Key a third time | The "Alt" key should change colour whilst held down. The program should also output "AltOn" to the console when the key is pressed. | Fail - the "Alt" key changed colour but the program sent the value "AltOn" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.13 | Press the "A" key after pressing the "Caps Lock" key | The "A" key should change colour whilst held down. The program should also send the value "A" when the "A" key is pressed | Fail - the "A" key changed colour but the program sent the value "A" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.14 | Press the "4" key after pressing the "Shift" key | The "4" key should change colour whilst held down. The program should also send the value "$" when the "4" key is pressed | Fail - the "4" key changed colour but the program sent the value "$" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.15 | Press the "A" key after pressing the "Shift" key | The "A" key should change colour whilst held down. The program should also send the value "A" when the "A" key is pressed | Fail - the "A" key changed colour but the program sent the value "A" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |

| | | | | |
|---|---|---|---|---|
| 2.3.16 | Press the "4" key after pressing the "Caps Lock" key | The "4" key should change colour whilst held down. The program should also send the value "4" when the "4" key is pressed | Fail - the "4" key changed colour but the program sent the value "4" when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |
| 2.3.17 | Press the "Unicode" key 8 times | The "Unicode" key should change colour whilst held down and the program should cycle through the predefined list of unicode characters, sending each when pressed. eg: "☺", "≠", "∬", "ɔ", "☺", "≠", "∬", "ɔ" | Fail - the "Unicode" key changed colour and cycled through the list of unicode characters but began at the second item of the list, "≠" rather than the expected "☺". The program also sent these values when the key was released, rather than when it was pressed | Video Evidence 2.3.mp4 |

## 4.6 - Implementation

I have chosen to implement my interface using a series of functions and procedures, as shown in the Figure 4.6.1

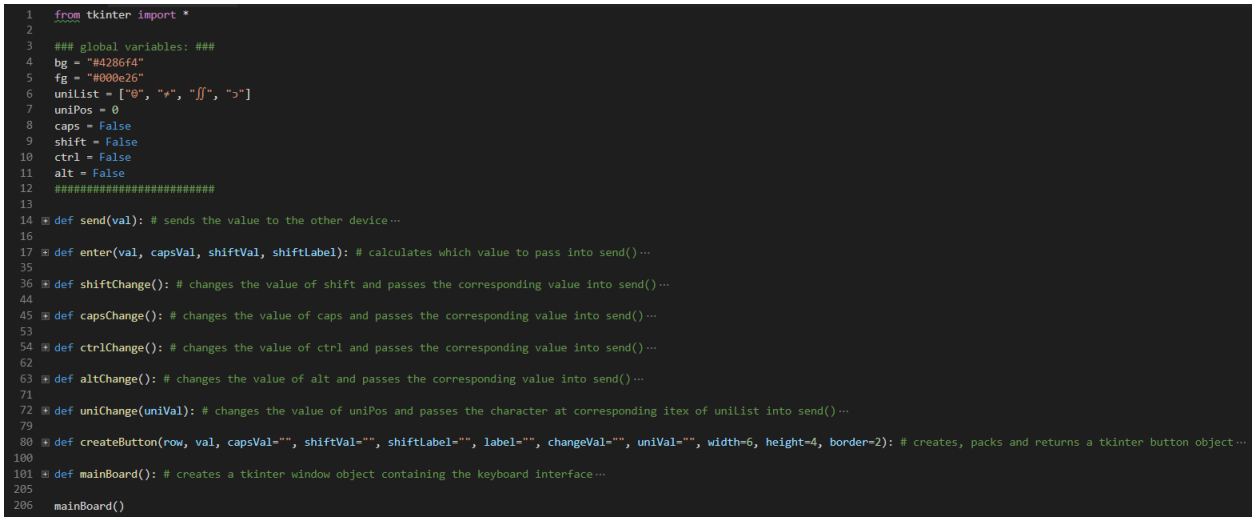


**Figure 4.6.1 - A screenshot of the whole of my implementation, with all subroutines minimised**

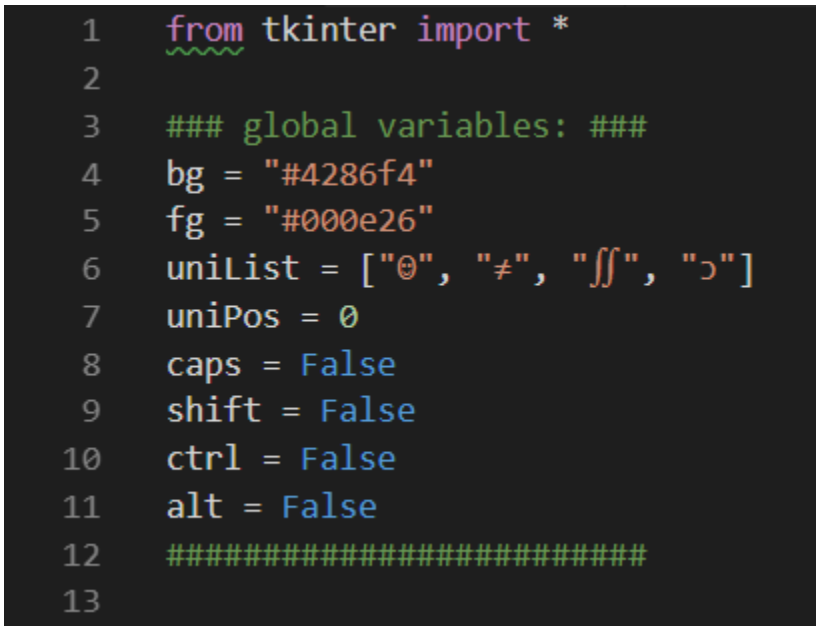


**Figure 4.6.2 - A screenshot of the first section of my implementation**

Line 1 imports the entirety of the tkinter library, which I will use to create my GUI. Lines 4 to 11 contain the global variables that I will use in my program. bg and fg stores the colour of the

background and foreground of the keys as a hexadecimal string. uniList stores the list of characters that the unicode key will cycle between. uniPos stores the position of the previous character in uniList. caps, shift, ctrl and alt are store boolean values indicating whether their respective modifier keys are enabled.

```
13
14  def send(val): # sends the value to the other device
15      print(val)
16
```

**Figure 4.6.3 - A screenshot of the send() procedure**

send() is a placeholder procedure that will send the value passed into it to the other device. To save time implementing my solution, I opted to focus on the interface itself, and thus had the procedure print the passed value rather than sending it to another device. I will change this at a later date.

```
16
17  def enter(val, capsVal, shiftVal, shiftLabel): # calculates which value to pass into send()
18      global shift, caps
19      if shift:
20          if shiftVal!="":
21              send(shiftVal)
22          elif shiftLabel!="":
23              send(shiftLabel)
24          elif capsVal != "":
25              send(capsVal)
26          else:
27              send(val)
28      elif caps:
29          if capsVal != "":
30              send(capsVal)
31          else:
32              send(val)
33      else:
34          send(val)
35
```

**Figure 4.6.4 - A screenshot of the enter() procedure**

The procedure enter() takes in the attributes val, capsVal, shiftVal and shiftLabel of a given key as parameters. The procedure then uses the value of the global variables shift and caps to

determine the most appropriate value to send. This value is then passed into the send() procedure.

```
35
36  def shiftChange(): # changes the value of shift and passes the corresponding value into send()
37      global shift
38      if shift:
39          shift=False
40          send("ShiftOff")
41      else:
42          shift=True
43          send("ShiftOn")
44
```

**Figure 4.6.5 - A screenshot of the shiftChange() procedure**

The procedure shiftChange() changes the value of the global variable shift and passes either “ShiftOn” or “ShiftOff” into send().

```
44
45  def capsChange(): # changes the value of caps and passes the corresponding value into send()
46      global caps
47      if caps:
48          caps=False
49          send("CapsOff")
50      else:
51          caps=True
52          send("CapsOn")
53
```

**Figure 4.6.6 - A screenshot of the capsChange() procedure**

The procedure capsChange() changes the value of the global variable caps and passes either “CapsOn” or “CapsOff” into send().

```
53
54  def ctrlChange(): # changes the value of ctrl and passes the corresponding value into send()
55      global ctrl
56      if ctrl:
57          ctrl=False
58          send("CtrlOff")
59      else:
60          ctrl=True
61          send("CtrlOn")
62
```

**Figure 4.6.7 - A screenshot of the ctrlChange() procedure**

The procedure ctrlChange() changes the value of the global variable ctrl and passes either “CtrlOn” or “CtrlOff” into send().

```
62
63  def altChange(): # changes the value of alt and passes the corresponding value into send()
64      global alt
65      if alt:
66          alt=False
67          send("AltOff")
68      else:
69          alt=True
70          send("AltOn")
71
```

**Figure 4.6.8 - A screenshot of the altChange() procedure**

The procedure altChange() changes the value of the global variable alt and passes either “AltOn” or “AltOff” into send().

```
71
72  def uniChange(uniVal): # changes the value of uniPos and passes the character at corresponding itex of uniList into send()
73      global uniList, uniPos
74      if uniPos < len(uniList)-1:
75          uniPos += 1
76      else:
77          uniPos = 0
78      send(uniList[uniPos])
79
```

**Figure 4.6.9 - A screenshot of the uniChange() procedure**

The procedure uniChange() uses the global variables uniList and uniPos to print a variety of unicode values. It does this by incrementing uniPos if doing so will not cause it to contain an index of uniList that does not exist, in which case uniPos will be set to 0. The character at the new index of uniList is then passed into the procedure send(). This allows uniChange() to cycle through the values stored in uniList, sending the next character each time the procedure is called. uniChange takes in the parameter uniVal, this is unnecessary at this stage but will allow me to implement specific unicode keys in the future if needed.

```
79
80  def createButton(row, val, capsVal="", shiftVal="", shiftLabel="", label="", changeVal="", uniVal="", width=6, height=4, border=2): # creates, packs and returns a tkinter button object
81      if label == "":
82          if capsVal!="":
83              label = capsVal
84          else:
85              label = val
86      if changeVal == "caps":
87          button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border,bg=bg, fg=fg, anchor=N, command=lambda: capsChange())
88      elif changeVal == "shift":
89          button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border,bg=bg, fg=fg, anchor=N, command=lambda: shiftChange())
90      elif changeVal == "ctrl":
91          button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border,bg=bg, fg=fg, anchor=N, command=lambda: ctrlChange())
92      elif changeVal == "alt":
93          button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border,bg=bg, fg=fg, anchor=N, command=lambda: altChange())
94      elif uniVal != "":
95          button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border,bg=bg, fg=fg, anchor=N, command=lambda: uniChange(uniVal))
96      else:
97          button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border,bg=bg, fg=fg, anchor=N, command=lambda: enter(val, capsVal, shiftVal, shiftLabel))
98      button.pack(side=LEFT)
99      return button
100
```

**Figure 4.6.10 - A screenshot of the createButton() function**

The function createButton() takes in a large number of parameters and uses them to create a tkinter button object.

Lines 81 to 85 checks whether the string stored in the parameter label is empty, if so the contents of label are replaced with either the contents of capsVal or val, depending on whether capsVal also contains an empty string.

Lines 86 to 97 check the value of the parameter changeVal and the button created is set to call a procedure when pressed, the procedure called depends on the value of changeVal but the remainder of the definition of the button object is identical. The all other parameters are used to define the attributes of the button object, with the exception of row.

Line 98 calls the pack() method of the button object, which will add it to the tkinter frame object held in the parameter row, allowing it to be loaded to the screen in the corresponding row.

Line 99 returns the button object

```
100
101 def mainBoard(): # creates a tkinter window object containing the keyboard interface
102     global bg
103     height1 = 5
104     width1 = 10
105     width2 = 6
106     height6 = 7
107     home = Tk()
108     home.title("keyboard")
109     home.geometry("800x480")
110     home.resizable(False, False)
111     home["bg"]=bg
112     row1 = Frame(home, bg=bg)
113     row1.pack(side=TOP, anchor=W)
114     row2 = Frame(home)
115     row2.pack(side=TOP, anchor=W)
116     row3 = Frame(home)
117     row3.pack(side=TOP, anchor=W)
118     row4 = Frame(home)
119     row4.pack(side=TOP, anchor=W)
120     row5 = Frame(home)
121     row5.pack(side=TOP, anchor=W)
122     row6 = Frame(home)
123     row6.pack(side=TOP, anchor=W)
124
```

**Figure 4.6.11 - A screenshot of the first part of the procedure mainBoard()**

The procedure mainBoard() creates a tkinter window object called home and uses it to create the keyboard interface.

Lines 103 to 106 contain local variables used to define the dimensions of keys on specific rows, allowing me to change the size of all keys on a specific row, allowing me to more easily adjust the layout of the keyboard.

Lines 107 to 111 create the window object and assign it specific properties, such as making it non-resizable.

Lines 112 to 123 create six tkinter frame objects, which will serve as rows, allowing me to have more control over the layout of my keyboard.

```
124
125     macroFrame = Frame(row1)
126     macroFrame.pack(fill=NONE, padx=240, pady=5, side=LEFT)
127     MuteKey = createButton(row1, "Mute", label="Mute", height=height1, width=width1)
128     PlayPauseKey = createButton(row1, "PlayPause", label="▶/⏯", height=height1, width=width1)
129     VolDownKey = createButton(row1, "VolDown", label="⇩", height=height1, width=width1)
130     VolUpKey = createButton(row1, "VolUp", label="⇧", height=height1, width=width1)
131
```

**Figure 4.6.12 - A screenshot of the second section of the procedure mainBoard()**

Lines 125 and 126 create a new tkinter frame object called macroFrame and places it in the first row of the keyboard.

Lines 127 to 130 use the createButton() function to create the media keys and place them in the first row.

```
131
132     AccentKey = createButton(row2, "`", shiftLabel="¬", width=width2+1)
133     OneKey = createButton(row2, "1", shiftLabel="!", width=width2)
134     TwoKey = createButton(row2, "2", shiftLabel="\"", width=width2)
135     ThreeKey = createButton(row2, "3", shiftLabel="£", width=width2)
136     FourKey = createButton(row2, "4", shiftLabel="$", width=width2)
137     FiveKey = createButton(row2, "5", shiftLabel="%", width=width2)
138     SixKey = createButton(row2, "6", shiftLabel="^", width=width2)
139     SevenKey = createButton(row2, "7", shiftLabel="&", width=width2)
140     EightKey = createButton(row2, "8", shiftLabel="*", width=width2)
141     NineKey = createButton(row2, "9", shiftLabel="(", width=width2)
142     TenKey = createButton(row2, "0", shiftLabel=")", width=width2)
143     MinusKey = createButton(row2, "-", shiftLabel="_", width=width2)
144     EqualKey = createButton(row2, "=", shiftLabel="+", width=width2)
145     BackKey = createButton(row2, "Back", label="←", shiftLabel="", width=20)
146
```

**Figure 4.6.13 - A screenshot of the third section of the procedure mainBoard()**

Lines 132 to 145 use the createButton() function to create and place the keys in the second row.

```
146
147     TabKey = createButton(row3, "Tab", width=13)
148     QKey = createButton(row3, "q", capsVal="Q")
149     WKey = createButton(row3, "w", capsVal="W")
150     EKey = createButton(row3, "e", capsVal="E")
151     RKey = createButton(row3, "r", capsVal="R")
152     TKey = createButton(row3, "t", capsVal="T")
153     YKey = createButton(row3, "y", capsVal="Y")
154     UKey = createButton(row3, "u", capsVal="U")
155     IKey = createButton(row3, "i", capsVal="I")
156     OKey = createButton(row3, "o", capsVal="O")
157     PKey = createButton(row3, "p", capsVal="P")
158     SqBrOKey = createButton(row3, "[", shiftLabel="{")
159     SqBrCKey = createButton(row3, "]", shiftLabel="}")
160     HashKey = createButton(row3, "#", shiftLabel="~", width=10)
161
```

**Figure 4.6.14 - A screenshot of the fourth section of the procedure mainBoard()**

Lines 147 to 160 use the createButton() function to create and place the keys in the third row.

```
161
162     CapsKey = createButton(row4, "Caps Lock", width=16, changeVal="caps")
163     AKey = createButton(row4, "a", capsVal="A")
164     SKey = createButton(row4, "s", capsVal="S")
165     DKey = createButton(row4, "d", capsVal="D")
166     FKey = createButton(row4, "f", capsVal="F")
167     GKey = createButton(row4, "g", capsVal="G")
168     HKey = createButton(row4, "h", capsVal="H")
169     JKey = createButton(row4, "j", capsVal="J")
170     KKey = createButton(row4, "k", capsVal="K")
171     LKey = createButton(row4, "l", capsVal="L")
172     SemiColonKey = createButton(row4, ";", shiftLabel=":")
173     ApostropheKey = createButton(row4, "'", shiftLabel="@")
174     EnterKey = createButton(row4,"\n", "↵", width=14)
175
176
```

**Figure 4.6.15 - A screenshot of the fifth section of the procedure mainBoard()**

Lines 162 to 174 use the createButton() function to create and place the keys in the fourth row.

```
175
176     ShiftLKey = createButton(row5, "Shift", width=11, changeVal="shift")
177     BSlashKey = createButton(row5, "\\", shiftLabel="|")
178     ZKey = createButton(row5, "z", capsVal="Z")
179     XKey = createButton(row5, "x", capsVal="X")
180     CKey = createButton(row5, "c", capsVal="C")
181     VKey = createButton(row5, "v", capsVal="V")
182     BKey = createButton(row5, "b", capsVal="B")
183     NKey = createButton(row5, "n", capsVal="N")
184     MKey = createButton(row5, "m", capsVal="M")
185     CommaKey = createButton(row5, ",", shiftLabel="<")
186     StopKey = createButton(row5, ".", shiftLabel=">")
187     FSlashKey = createButton(row5, "/", shiftLabel="?")
188     shiftRKey = createButton(row5, "Shift", width=20, changeVal="shift")
189
```

**Figure 4.6.16 - A screenshot of the sixth section of the procedure mainBoard()**

Lines 176 to 188 use the createButton() function to create and place the keys in the fifth row.

```
189
190     CtrlLKey = createButton(row6, "Ctrl", width = 11, height=height6, changeVal="ctrl")
191     WinLKey = createButton(row6, "Win", height=height6)
192     AltLKey = createButton(row6, "Alt", height=height6, changeVal="alt")
193     SpaceKey = createButton(row6, " ", label="_____", width=48, height=height6)
194     UnicodeKey = createButton(row6, "Unicode", width=8, height=height6, uniVal=0)
195     WinRKey = createButton(row6, "Win", height=height6)
196     MenuKey = createButton(row6, "Menu", height=height6)
197     CtrlRKey = createButton(row6, "Ctrl", width=13, height=height6, changeVal="ctrl")
198
199     home.mainloop()
200
```

**Figure 4.6.17 - A screenshot of the seventh section of the procedure mainBoard()**

Lines 190 to 197 use the createButton() function to create and place the keys in the sixth row.

Line 199 indicates the end of the infinite loop used by tkinter to run the interface.

```
200
201   mainBoard()
202
```

**Figure 4.6.18 - A screenshot of where mainBoard() is called**

Line 201 calls the procedure mainBoard()

## 4.7 - Evaluation

### 4.7.1 - Changes made during my implementation

During my implementation I made a number of changes to the way my interface operates. These are:

- The use of tkinter frame objects to help specify the layout of the keys.
- The use the createButton() to encapsulate the creation and packing of buttons.
- The “Shift” and “Caps Lock” keys switching between being on and off, rather than being on when held down.
- The “Shift”, “Caps Lock”, “Alt” and “Ctrl” keys all alternate between sending values indicating whether they are on and off, rather than sending one value indicating that the key has been pressed.
- The “Unicode” key now cycles through the characters in the list held in uniList, rather than sending specific values.
- The use of procedures to encapsulate the rest of my program.

### 4.7.2 - Changes made to my layout

The layout of the keyboard also deviated from the one planned in section 4.2, These differences are:

- A blue colour scheme rather than a monochrome colour scheme.
- The “Volume Up” and “Volume Down” keys have been swapped.
- The “Enter” key has been changed to use only a single row.
- The position of the “#” key has been raised by one row to make space for the “Enter” key
- The icons on keys such as “Tab” and “Shift” have been removed.
- The icons on the “Windows” keys have been replaced with the text “Win”.
- The text on the “Space” key has been replaced with a line.
- The bottom row has been stretched vertically to allow for the “Space” key to be used more easily.
- The dimensions of some of the keys have been altered to ensure as much of the window is utilised as possible.

### 4.7.3 - Changes made in response to testing

During testing, my prototype failed every test with the exception of test 2.1.1. In most of the tests, the only reason my device failed is that the button sent it’s value when released rather

than when pressed. This is a small issue that does not affect the performance of the keyboard enough to warrant a solution at this stage.

Another issue identified when testing was that the window containing the keyboard interface did not load correctly, This was due to the homebar of the pi and a difference in the dimensions of the keys. This difference in dimensions is likely to to the tkinter button objects being represented differently on the pi than on the machine used to write the code, as the machines used different operating systems and IDE (Integrated Development Environment) software to each other.

In response to this I modified the mainBoard() procedure to make the window fullscreen, as shown on line 112 of figure 4.7.3.1.

```
101
102 def mainBoard(): # creates a tkinter window object containing the keyboard interface
103     global bg
104     height1 = 5
105     width1 = 6
106     width2 = 3
107     height6 = 7
108     home = Tk()
109     home.title("keyboard")
110     home.geometry("800x480")
111     home.resizable(False, False)
112     home.attributes("-fullscreen", True)
113     home["bg"]=bg
```

**Figure 4.7.3.1 - A screenshot of the first part of the modified version of the procedure mainBoard()**

I also changed the values of the dimensions of the tkinter button objects, to fit the pi, as shown in Photo Evidence 2.1 to 2.9. The new layout is shown in figure 4.7.3.2

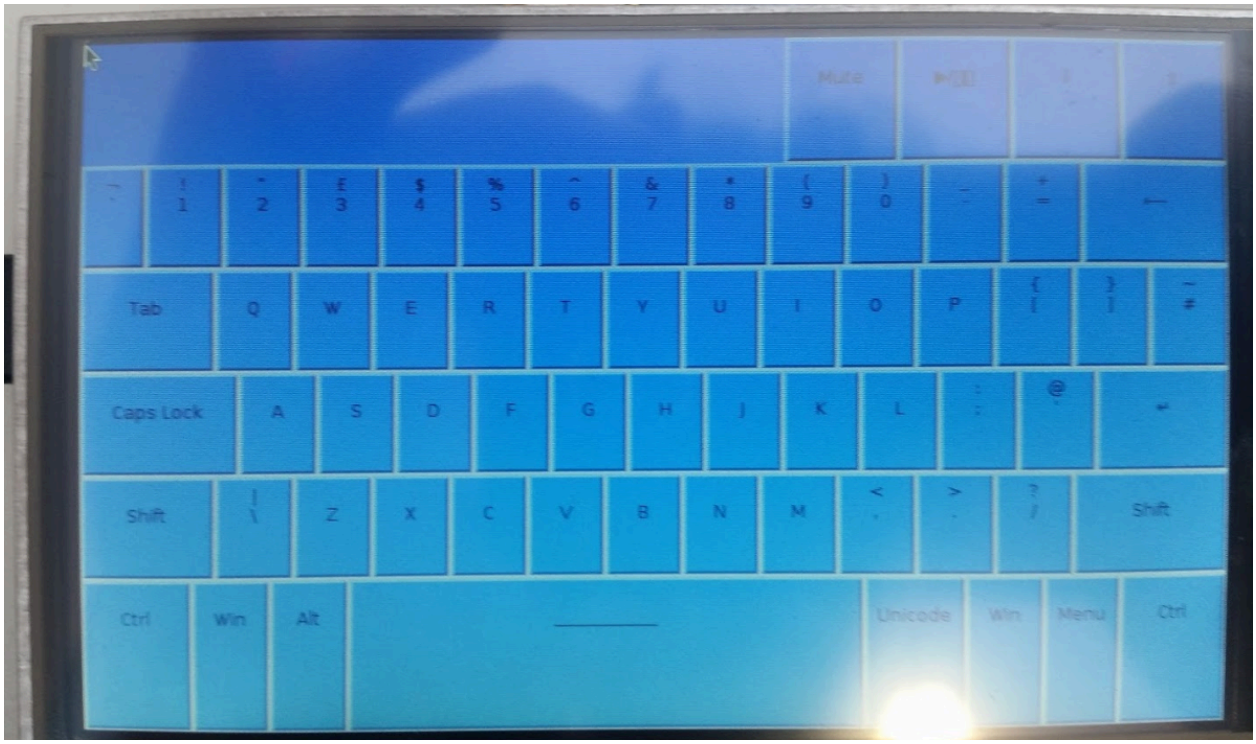


**Figure 4.7.3.2 - A photo of the improved GUI**

The final issue identified during testing was that the “Unicode” key began from the second item in uniList. This was not a major issue but could be easily fixed by modifying the procedure uniChange() to send the value before incrementing uniPos, as shown in figure 4.7.3.3

```
71
72  def uniChange(uniVal): # changes the value of uniPos and passes the character at corresponding itex of uniList into send()
73      global uniList, uniPos
74      send(uniList[uniPos])
75      if uniPos < len(uniList)-1:
76          uniPos += 1
77      else:
78          uniPos = 0
79
```

**Figure 4.7.3.3 - A screenshot of the modified version of the procedure uniChange()**

### 4.7.4 - Evaluation of success criteria

Criteria I had aimed to complete:

| Criteria | Evaluation | Explanation |
|---|---|---|
| Create the software required for a basic ASCII keyboard | Fail | The keyboard does not send ASCII values at this time, as it uses the placeholder procedure send() which currently just outputs the value to the console as a string |
| Create a GUI for a basic QWERTY keyboard | Pass | The GUI that contains all keys needed for a basic QWERTY keyboard |
| Create a button capable of inputting unicode characters | Pass | The “Unicode” key is able to input a sequence of unicode values from a list |

Criteria I also attempted to satisfy where possible:

| Criteria | Evaluation | Explanation |
|---|---|---|
| Implement functioning media keys to the keyboard | Fail | The keyboard does have some basic media keys but at this time these keys only use the procedure send() to print a string corresponding to the media key |
| Create keys capable of being used as macros | Fail | The keyboard does have a frame dedicated to macro keys, but it is empty at this time as macro keys have not been implemented |
| Implement a system to save macros to memory | Fail | To implement his feature I would first need to incorporate macro keys into my GUI |

# 5 - Cycle 3: Customisation

## 5.1 - Cycle 3 Outline

### 5.1.1 intentions for Cycle 3

In this cycle I will focus on the customisation options of the keyboard. This will involve modifying the keyboard software to allow for a greater number of customisation options such as colour and layout and implement a system to save multiple profiles to memory. I will also create a separate software application that will allow the user to change the values of these options. If possible, I will also implement additional features, such as the ability to simulate peripherals or the presence of an SDK.

### 5.1.2 Success Criteria

The criteria I aim to complete in this cycle are as follows:

- Modify my keyboard software to allow for a greater number of customisation options.
- Implement a system to save these customisation options to memory.
- Create a simple command line software application to change the values of the options stored in memory.

Where possible I will also attempt to satisfy the following criteria:

- Modify my keyboard software to allow it to simulate a number of additional peripherals
- Create and implement a GUI for the customisation software
- Create a SDK for my system to allow others to more easily create software that is compatible with mine
- Implement a system to allow my keyboard software to be run automatically when the device is turned on

## 5.2 - Selecting customisation options

Before I am able to continue with this cycle, I must first decide on which customisation options I intend to implement and what values will be available to select. Below is the list of customisation options I have decided to include:

| Option | Explanation | Available values |
|---|---|---|
| Layout | This will be the keyboard layout displayed by the software | The user will be able to choose between a number of predefined layouts, such as Dvorak and Colemak |
| Background | This will be the colour given to the background of the keyboard and the background of the keys when not pressed | The user will be able to enter a hexadecimal value specifying the proportion of red, green and blue, with each colour having a value ranging from 0 to 255 |
| Foreground | This will be the colour given to the text of the keys when not pressed | The user will be able to enter a hexadecimal value specifying the proportion of red, green and blue, with each colour having a value ranging from 0 to 255 |
| Active background | This will be the colour given to the background of the keyboard and the background of the keys when pressed | The user will be able to enter a hexadecimal value specifying the proportion of red, green and blue, with each colour having a value ranging from 0 to 255 |
| Active foreground | This will be the colour given to the text of the keys when pressed | The user will be able to enter a hexadecimal value specifying the proportion of red, green and blue, with each colour having a value ranging from 0 to 255 |
| Unicode value | This will be the value given to the unicode key | The user will be able to enter any single character. |

## 5.3 - Design of solution

Once I had decided on the customisation options I was going to implement, I had to break down how I would go about implementing them. This involved designing three aspects of the solution, These were a system to save multiple profiles to memory, an algorithm for the keyboard

software to access the value of the options within these profiles and an algorithm for the customisation software to create  and modify profiles.

### 5.3.1 System to save multiple profiles to memory

The first aspect of my solution that needed to be addressed was designing the file structure that I would use to save multiple profiles to memory.

The system I designed involves a file named “Profiles”, which I would use to store the names of all profiles. The first line of “Profiles” would contain the name of the profile currently in use. Each additional line would contain the name of one profile. This means that from line 2 onwards the name of every profile, including the one currently in use, will appear once, as shown in figure 5.3.1.1.

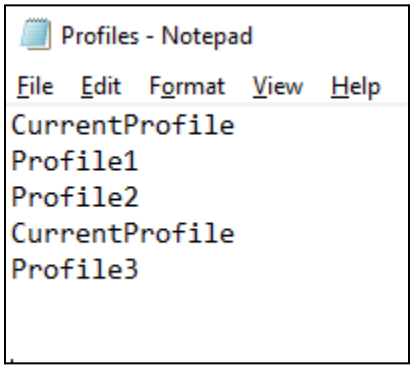


**Figure 5.3.1.1 - An example of the file "Profiles"**

Each profile will exist as a file, with a name corresponding to it’s entry in “Profiles”. Each of these files will contain the value of each option on a separate line. Line 1 will contain the layout, line will contain the background, line 3 will contain the foreground, line 4 will contain the active background, line 5 will contain the active foreground and line 6 will contain the unicode value as shown in figure 5.3.1.2.

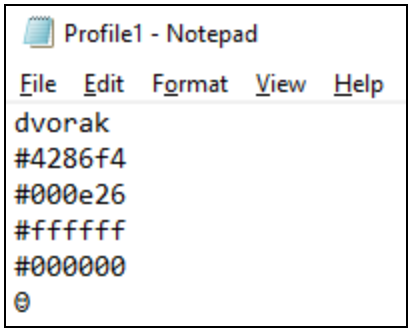


**Figure 5.3.1.2 - An example of a file representing the profile "Profile1"**

### 5.3.2 Algorithm to access profiles from memory

Once I had designed a system to store multiple profiles in memory, I was able to to break down how I would modify my keyboard software to allow me to implement these options and access them from file. I did this by creating a pseudocode algorithm to be added to the main section of the keyboard software. This algorithm is shown in figure 5.3.2.1

```
Profiles = open("Profiles")
CurrentProfileName = Profiles.readLine(1)
CurrentProfile = open(CurrentProfileName)
Values = CurrentProfile.readLines()
if Values[0] == "dvorak":
      dvorakBoard(Values)
else if Values[0] == "colemak":
      colemakBoard(Values)
else:
      qwertyBoard(Values)
end if
```

**Figure 5.3.2.1 - The main section of a pseudocode algorithm to access profiles from memory**

The first line of the algorithm opens the file "Profiles", creating an object.

```
Profiles = open("Profiles")
```

The next section of the algorithm reads the first line of “Profiles” and stores this line using the variable “CurentProfileName”. The algorithm then opens the file with this name, creating the object “CurrentProfile”.

```
CurrentProfileName = Profiles.readLine(1)
CurrentProfile = open(CurrentProfileName)
```

The next section of the algorithm creates an array of strings containing each line of the file currently represented by the object “CurrentProfile” and stores then using the variable “Values”.

```
Values = CurrentProfile.readLines()
```

The next section of the algorithm checks whether the string at index 0 of Values is the string “dvorak”, if it is, the algorithm calls the procedure dvorakBoard(), with the parameter “values”, this is a procedure similar to the procedure mainBoard() from cycle 2, except the locations of the keys conform to the dvorak layout.

```
if Values[0] == "dvorak":
      dvorakBoard(Values)
```

The next section of the algorithm checks whether the string at index 0 of Values is the string “colemak”, if it is, the algorithm calls the procedure colemakBoard(), with the parameter “values”. colemakBoard() is a procedure similar to the procedure mainBoard() from cycle 2, except the locations of the keys conform to the colemak layout.

```
else if Values[0] == "colemak":
      colemakBoard(Values)
```

The next section of the algorithm calls the procedure qwertyBoard with the parameter “Values” if neither of the previous conditions are met, this is a procedure similar to the procedure mainBoard() from cycle 2, where the locations of the keys conform to the qwerty layout.

```
else:
      qwertyBoard(Values)
end if
```

The variable “Values” can then be passed into the procedure createButton() which can be easily modified to use indexes of “Values” as parameters in the tkinter function “Button()” to apply the settings to tkinter button objects.

### 5.3.3 Algorithm for the customisation software

Once I had designed an algorithm to load the current profile from memory, I was left with the task of designing an algorithm for the customisation software. This software needed to be able to create, delete and modify profiles, as well as change the current profile.

Below is a pseudocode algorithm for this software.

```
import os
function hexChecker(value):
      validChars = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "a", "b", "c", "d", "e", "f"]
      if length(value) != 7:
            return false
      else if value[0] != "#":
            return false
      end if
      for i from i=1 to length(value):
            if value[i] NOT in validChars:
                  return false
            end if
      end for
      return true
end function
```

**Figure 5.3.3.1 - The first section of my algorithm for the customisation software**

This section of my algorithm contains the hexChecker() function. This function validates that a string passed to it conforms to the hex colour formatting. The function returns true is the string is valid and false if not.

```
procedure newProfile(profileNames, keyboardLayouts):
      name = input("Please enter a name for the new profile: )
      while name in profileNames:
            name = input("That name is already taken, please choose another)
      end while
      print("Please enter the keyboard layout you would like to use, the options are:")
```

```
for item in keyboardLayouts:
      print(item)
end for
layout = input("")
while layout NOT in keyboardLayouts:
      print("That is not a valid layout, please select one of the following:")
      for item in keyboardLayouts:
            print(item)
      end for
      layout = input("")
end while
background = input("Please enter a hex colour value for the background, eg #3d5f96:")
while hexCheck(background) == false:
      print(Sorry, that is not a valid value)
      background = input("Please enter a hex colour value for the background, eg #3d5f96:")
end while
foreground = input("Please enter a hex colour value for the foreground, eg #3d5f96:")
while hexCheck(foreground) == false:
      print(Sorry, that is not a valid value)
      foreground = input("Please enter a hex colour value for the foreground, eg #3d5f96:")
end while
activeBackground = input("Please enter a hex colour value for the active background, eg: #3d5f96:")
while hexCheck(activeBackground) == false:
      print(Sorry, that is not a valid value)
      activeBackground = input("Please enter a hex colour value for the active background, eg #3d5f96:")
end while
activeForeground = input("Please enter a hex colour value for the active foreground, eg: #3d5f96:")
while hexCheck(activeForeground) == false:
      print(Sorry, that is not a valid value)
      activeForeground = input("Please enter a hex colour value for the active foreground, eg #3d5f96:")
end while
uniVal = input("Please enter a unicode character for the unicode key)
while length(uniVal) != 1:
      uniVal = input("That is not a valid character, please try again")
end while
file = createFile(name)
file.write(layout)
file.write(background)
file.write(foreground)
file.write(activeBackground)
file.write(activeForeground)
file.write(uniVal)
```

```
      file.close
      Profiles = open(“Profiles”)
      Profiles.write(name)
      Profiles.close
end procedure
```

**Figure 5.3.3.2 - The second section of my algorithm for the customisation software**

This section of my algorithm contains the procedure newProfile(). This procedure allows the user to create a new profile. It does this by asking the user for a name, layout, colours and unicode value and validates each. Once a valid input is given for each, a new file is created, the corresponding data is added and the new file is added to the file “Profiles”

```
procedure editProfile(profileNames, keyboardLayouts):
      print("Please enter one of the following files to edit:)
      for item in profileNames:
            print(item)
      end for
      selectedName = input("")
      while selectedName NOT in profileNames:
            print("That is not a valid profile, please select one of the following")
            for item in profileNames:
                  print(item)
            end for
      end while
      file = open(selectedName)
      values = file.readlines()
      print("The file you have selected has the following values:")
      print("layout: ", values[0])
      print("background: ", values[1])
      print("foreground: ", values[2])
      print("activeBackground: ", values[3])
      print("activeForeground: ", values[4])
      print("unicodeValue: ", values[5])
      option = input("Please enter one of these values to change)
      while option NOT in [“layout”, “background”, “foreground”, “activeBackground”, “activeForeground”, “unicodeValue”]:
            print(That is not a valid option, please enter one of the following:)
            print("layout")
            print("background")
            print("foreground")
```

```
            print("activeBackground")
            print("activeForeground")
            print("unicodeValue")
            option = input("")
      end while
      if option == "layout":
            newVal = input("Please select a new layout")
            while newVal NOT in keyboardLayouts:
                  print("That is not a valid layout, please select from: ")
                  for item in keyboardLayouts:
                        print(item)
                  end for
                  newVal = input("")
            end while
            values[0] = newVal
      else if option  == “background”:
            newVal = input("Please enter a new hex colour value for the background, eg #3d5f96:")
            while hexCheck(newVal) == false:
                  print(Sorry, that is not a valid value)
                  background = input("Please enter a new hex colour value   for the background, eg #3d5f96:")
            end while
            values[1] = newVal
      else if option  == “foreground”:
            newVal = input("Please enter a new hex colour value for the foreground, eg #3d5f96:")
            while hexCheck(newVal) == false:
                  print(Sorry, that is not a valid value)
                  newVal = input("Please enter a new hex colour value for the foreground, eg #3d5f96:")
            end while
            values[2] = newVal
      else if option  == “activeBackground”:
            newVal = input("Please enter a new hex colour value for the active background, eg #3d5f96:")
            while hexCheck(newVal) == false:
                  print(Sorry, that is not a valid value)
                  newVal = input("Please enter a new hex colour value for the active background, eg #3d5f96:")
            end while
            values[3] = newVal
      else if option  == “activeForeground”:
            newVal = input("Please enter a new hex colour value for the active foreground, eg #3d5f96:")
            while hexCheck(newVal) == false:
                  print(Sorry, that is not a valid value)
                  newVal = input("Please enter a new hex colour value for the active foreground, eg #3d5f96:")
            end while
```

```
            values[4] = newVal
      else:
            newVal = input("Please enter a new unicode character for the unicode key)
            while length(newVal) != 1:
                  newVal= input("That is not a valid character, please try again")
            end while
            Values[5] = newVal
      end if
      file.clear()
      for item in Values:
            file.write(item)
      end for
      file.close()
end procedure
```

**Figure 5.3.3.3 - The third section of my algorithm for the customisation software**

This section of my algorithm contains the editProfile() procedure. This procedure allows a user to change one of the values of a profile. It does this by asking the user for a profile, validating that it exists, and then storing its contents in the array Values. The user is asked for a new value, this is validated and the corresponding value is changed in Values. The contents of the file containing the profile is cleared and each index of Values is written to the file. This creates the new file.

```
procedure deleteProfile(profileNames, currentProfile):
      print("Please select one of the following files to delete: )
      for item in profileNames:
            print(item)
      end for
      selectedProfile = input("")
```

```
      while selectedProfile NOT in profileNames:
            selectedProfile = input("That profile does not exist, please try again")
      end while
      Profiles = open("Profiles")
      Profiles.clear()
      Profiles.write(currentProfile)
      for item in profileNames:
            if item != selectedProfile:
                  Profiles.write("item")
            end if
      end for
      os.remove(selectedProfile)
end procedure
```

**Figure 5.3.3.4 - The fourth section of my algorithm for the customisation software**

This section of my algorithm contains the procedure deleteProfile(). This procedure allows the user to delete a profile. It does this by asking them for a profile, validating that is exists, removing it from the file “Profiles” and using the os library to remove it from memory.

```
procedure changeProfile(profileNames, currentProfile):
      print("The available profile are:")
      for item in profileNames:
            print(item)
      end for
      newProfile = input("Please choose one of the above values")
      while newProfile NOT in profileNames:
```

```
            print("That is not a valid profile")
            newProfile = input("Please choose a profile")
      end while
      Profiles = open("Profiles")
      Profiles.clear
      Profiles.write(currentProfile)
      for item in profileNames:
            Profiles.write(item)
      end for
      Profiles.close()
end procedure
```

**Figure 5.3.3.5 - The fifth section of my algorithm for the customisation software**

This section of my algorithm contains the changeProfile() procedure. This procedure allows the user to change the profile that is currently selected. This is achieved by asking the user for a profile, validating that it exists, clearing the contents of the file “Profiles” and writing the selected profile to the file followed by each of the profiles stored in the list profileNames.

```
Repeat = true
print("Welcome to the keyboard customisation software")
print("Please select one of the following options:")
Profiles = open("Profiles")
profiles = Profiles.readLines()
currentProfile = profiles[0]
Profiles.close()
profileNames = []
keyboardLayouts = [qwerty, dvorak, colemak]
for item in profiles:
      if item NOT in profileNames:
            profileNames.append(item)
      end if
end for
while Repeat:
      print("1: Create a new profile")
      print("2: Edit an existing profile")
```

```
      print("3: Delete an existing profile ")
      print("4: Change current profile")
      OptionInput = input("")
      if OptionInput == "1":
            newProfile(profileNames, keyboardLayouts)
      else if OptionInput == "2":
            editProfile(profileNames, keyboardLayouts)
      else if OptionInput == "3":
            deleteProfile(profileNames)
      else if OptionInput == "4":
            changeProfile(profileNames, currentProfile)
      else:
            print("That is not a valid option")
      end if
      RepeatInput = input("If you would like choose another option, please type \"REPEAT\"")
      if RepeatInput != "REPEAT":
            Repeat = False
      end if
end while
```

**Figure 5.3.3.6 - The sixth section of my algorithm for the customisation software**

This section of my algorithm contains the main section of the algorithm. This is opens the file “Profiles” and asks the user which of the previous subroutines they would like to use. After validating this, the subroutine is called and the user is asked if they would like to make another change, the program repeats every time the user enters REPEAT.

## 5.4 - Data Structures

| Data structures within the algorithm to access profiles from memory | | |
|---|---|---|
| Name | Data Structure | Contents |
| Profiles | Object | The text file “Profiles” |
| CurrentProfileName | String | The first line of the file “Profiles” containing the name of the currently |

| | | |
|---|---|---|
| | | selected profile |
| CurrentProfile | Object | The text file representing the profile that is currently selected |
| Values | List | The contents of the file representing the profile that is currently selected, with each value existing as a separate data item of type string |
| Data structures within the hexChecker() function of the algorithm for the customisation software | | |
| Name | Data Structure | Contents |
| value | String | The value that is being validated |
| validChars | List | The list of characters that are acceptable in the numerical portion of the hex code |
| i | integer | The index within string of the character currently being valiated |
| Data structures within the newProfile() procedure of the algorithm for the customisation software | | |
| Name | Data Structure | Contents |
| profileNames | List | The names of each of the available profiles, each appearing only once as a string. |
| keyboardLayouts | List | The list of acceptable keyboard layouts, with each value existing as a separate data item of type string. The list is currently planned to contain the values “dvorak”, “colemak” and “qwerty” but layouts may be added or removed from this list as the project progresses |
| name | String | The most recent value entered by the user for the name of the new profile |
| item | String | The string within keyboardLayout that needs to be printed |
| layout | String | The most recent value entered by the user for the keyboard layout of the new profile |
| background | String | The most recent hex code value entered by the user for the background colour |
| foreground | String | The most recent hex code value entered by the user for the foreground colour |
| activeBackground | String | The most recent hex code value entered by the user for the active background colour |
| activeForeground | String | The most recent hex code value entered by the user for the active foreground colour |
| uniVal | String | The most recent string entered by the user to be used as the character for the unicode key |

| | | |
|---|---|---|
| file | Object | The file representing the new profile |
| Profiles | Object | The text file “Profiles” |
| Data structures within the editProfile() procedure of the algorithm for the customisation software | | |
| Name | Data Structure | Contents |
| profileNames | List | The names of each of the available profiles, each appearing only once as a string. |
| keyboardLayouts | List | The list of acceptable keyboard layouts, each existing as strings. The list is currently planned to contain the values “dvorak”, “colemak” and “qwerty” but layouts may be added or removed from this list as the project progresses |
| item | String | The next string in the list to be printed or written to file |
| selectedName | String | The name of the profile the user would like to edit |
| file | Object | The file containing the profile the user has chosen to edit |
| values | List | The values stored in the file containing the profile being edited, with each value existing as a separate data item of type string |
| option | String | The most recent string that the user has entered to indicate which value they would like to change. |
| newVal | String | The most recent string entered by the user to be used as the new value in the profile. |
| Data structures within the deleteProfile() procedure of the algorithm for the customisation software | | |
| Name | Data Structure | Contents |
| profileNames | List | The names of each of the available profiles, each appearing only once as a string. |
| currentProfile | String | The name of the profile that is currently being used by the keyboard |
| item | String | The next item in the list profileNames |
| selectedProfile | String | The most recent string entered by the user to indicate the profile they would like to delete |
| Profiles | Object | The text file “Profiles” |
| os | Object | The attributes and methods used by the os library |

| Data structures within the changeProfile() procedure of the algorithm for the customisation software | | |
|---|---|---|
| Name | Data Structure | Contents |
| profileNames | List | The names of each of the available profiles, each appearing only once as a string. |
| currentProfile | String | The name of the profile that is currently being used by the keyboard |
| item | String | The next string in the list to be printed or written to file |
| newProfile | String | The most recent string entered by the user to indicate the profile they would like to use. |
| Profiles | Object | The text file “Profiles” |
| Data structures within the main section of the algorithm for the customisation software | | |
| Name | Data Structure | Contents |
| Repeat | Boolean | Whether or not the user would like to make another change |
| Profiles | Object | The text file “Profiles” |
| profiles | List | The values stored in the file “Profiles”, with each line in the file existing as a separate data item of type string |
| currentProfile | String | The first item of the list “profiles”, this will be the first line of the file “Profile” containing the name of the profile that is currently selected |
| profileNames | List | Initially empty but will be filled with the name of each available profile, until all available profiles appear in the list as a single data item of type string |
| keyboardLayouts | List | The list of acceptable keyboard layouts, with each value existing as a separate data item of type string. The list is currently planned to contain the values “dvorak”, “colemak” and “qwerty” but layouts may be added or removed from this list as the project progresses |
| item | String | The next item in the list profiles to be added to the list profileNames if it is not already in the list |
| OptionInput | String | The string entered by the user to indicate the type of change they would like to make |
| RepeatInput | String | The string input by the user to indicate whether they would like to make another change |

## 5.5 - Testing and implementation of keyboard software

Due to the complexity of this cycle, I plan to complete it in two sections. I will begin by testing and implementing the improved keyboard software and file structure. Only after this is complete will I focus on testing and implementing the customisation software.

### 5.5.1 Test Plan for keyboard software

Below are the tests I plan to use to evaluate my implementation of the keyboard software.

| Test ID | Test | Reasoning | Expected Outcome |
|---|---|---|---|
| 3.1.1 | With the values of the current profile as follows, run the keyboard program: “qwerty”, “#4286f4”, “#000000”, “#6d0f0f”, “#ffffff”, “☺” | To test that the program can load the qwerty layout correctly | The program should load the qwerty layout with a blue background and black text |
| 3.1.2 | With the same values as test 3.1.1, press the “A” key | To test that the qwerty layout can correctly apply the values of active background and active background, and to test that the qwerty layout can input send data | The “A” key should turn red, with white text when pressed, should send the value “a” and should return to its previous colouration when another key is pressed |
| 3.1.3 | With the same values as test 3.1.1, press the “Unicode” key | To test that the unicode key works correctly in the qwerty layout and to test that the qwerty layout can change the value of the “Unicode” key based on the data from file | The “Unicode” key should turn red, with white text when pressed, should send the value “☺” when released and should return to its previous colouration when another key is pressed |
| 3.1.4 | With the values of the current profile as follows, run the keyboard program: “dvorak”, | To test that the program can load the dvorak layout correctly | The program should load the dvorak layout with a purple background and black text |

| | | | |
|---|---|---|---|
| | “#4e42f4”, “#000000”, “#033d0b”, “#ffffff”, “฿” | | |
| 3.1.5 | With the same values as test 3.1.4, press the “B” key | To test that the dvorak layout can correctly apply the values of active background and active background, and to test that the dvorak layout can input send data | The “B” key should turn green, with white text when pressed, should send the value “b” and should return to its previous colouration when another key is pressed |
| 3.1.6 | With the same values as test 3.1.4, press the “Unicode” key | To test that the unicode key works correctly in the dvorak layout and to test that the dvorak layout can change the value of the “Unicode” key based on the data from file | The “Unicode” key should turn green, with white text when pressed, should send the value “฿” when released and should return to its previous colouration when another key is pressed |
| 3.1.7 | With the values of the current profile as follows, run the keyboard program: “colemak”, “#ed97b7”, “#000000”, “#a05006”, “#ffffff”, “θ” | To test that the program can load the colemak layout correctly | The program should load the colemak layout with a pink background and black text |
| 3.1.8 | With the same values as test 3.1.7, press the “C” key | To test that the colemak layout can correctly apply the values of active background and active background, and to test that the colemak layout can input send data | The “C” key should turn brown, with white text when pressed, should send the value “c” and should return to its previous colouration when another key is pressed |
| 3.1.9 | With the same values as test 3.1.7, press the “Unicode” key | To test that the unicode key works correctly in the colemak layout and | The “Unicode” key should turn brown, with white text when pressed, should send the value “θ” when released and |

| | | | |
|---|---|---|---|
| | | to test that the colemak layout can change the value of the “Unicode” key based on the data from file | should return to its previous colouration when another key is pressed |

### 5.5.2 Prototype testing of the keyboard software

| Test ID | Test | Expected Outcome | Actual Outcome | Evidence |
|---|---|---|---|---|
| 3.1.1 | With the values of the current profile as follows, run the keyboard program: “qwerty”, “#4286f4”, “#000000”, “#6d0f0f”, “#ffffff”, “☺” | The program should load the qwerty layout with a blue background and black text | Pass - The program correctly loaded the qwerty layout with a blue background and black text | Video Evidence 3.1.mp4 |
| 3.1.2 | With the same values as test 3.1.1, press the “A” key | The “A” key should turn red, with white text when pressed and should send the value “a” | Pass - The "A" key turned red with white text when pressed and output "a" to console | Video Evidence 3.1.mp4 |
| 3.1.3 | With the same values as test 3.1.1, | The “Unicode” key should turn red, with white text when | Pass - The "Unicode" key turned red with white text when pressed and output "☺" | Video Evidence 3.1.mp4 |

| | | | | |
|---|---|---|---|---|
| | press the “Unicode” key | pressed and should send the value “☺” when released | to console | |
| 3.1.4 | With the values of the current profile as follows, run the keyboard program: “dvorak”, “#a641f4”, “#000000”, “#033d0b”, “#ffffff”, “฿” | The program should load the dvorak layout with a purple background and black text | Pass - The program correctly loaded the dvorak layout with a purple background and black text | Video Evidence 3.2.mp4 |
| 3.1.5 | With the same values as test 3.1.4, press the “B” key | The “B” key should turn green, with white text when pressed and should send the value “b” | Pass - The "B" key turned green with white text when pressed and output "b" to console | Video Evidence 3.2.mp4 |
| 3.1.6 | With the same values as test 3.1.4, press the “Unicode” key | The “Unicode” key should turn green, with white text when pressed and should send the value “฿” when released | Pass - The "Unicode" key turned green with white text when pressed and output "฿" to console | Video Evidence 3.2.mp4 |
| 3.1.7 | With the values of the current profile as follows, run the keyboard program: “colemak”, “#ed97b7”, “#000000”, “#a05006”, “#ffffff”, “θ” | The program should load the colemak layout with a pink background and black text | Pass - The program correctly loaded the colemak layout with a pink background and black text | Video Evidence 3.3.mp4 |
| 3.1.8 | With the same values | The “C” key should turn brown, with white text when | Pass - The "C" key turned brown with white text when | Video Evidence |

| | | | | |
|---|---|---|---|---|
| | as test 3.1.7, press the “C” key | pressed and should send the value “c” | pressed and output “c” to console | 3.3.mp4 |
| 3.1.9 | With the same values as test 3.1.7, press the “Unicode” key | The “Unicode” key should turn brown, with white text when pressed and should send the value “θ” when released | Pass - The “Unicode” key turned brown with white text when pressed and output “θ” to console | Video Evidence 3.3.mp4 |

### 5.5.3 Implementation of the keyboard software

Due to large quantity of code, I will only comment on sections of the keyboard software that have been changed from section 4.6

```
1   from tkinter import *
2
3   ### global variables: ###
4   caps = False
5   shift = False
6   ctrl = False
7   alt = False
8   ##########################
9
```

**Figure 5.5.3.1 - The first section of the changes to the keyboard software**

In this section, the only change made was the removal of the global variables bg, fg, uniList and uniPos. These variables are no longer needed, as the data they previously held is now stored in file.

```
12
13  def send(val): # sends the value to the other device···
15
16  def enter(val, capsVal, shiftVal, shiftLabel): # calculates which value to pass into send()···
34
35  def shiftChange(): # changes the value of shift and passes the corresponding value into send()···
43
44  def capsChange(): # changes the value of caps and passes the corresponding value into send()···
52
53  def ctrlChange(): # changes the value of ctrl and passes the corresponding value into send()···
61
62  def altChange(): # changes the value of alt and passes the corresponding value into send()···
70
```

**Figure 5.5.3.2 - A minimised version of the subroutines that remained from the previous cycle**

In this section, the subroutines have been minimised as they are unchanged from how they were in figures 4.6.3 to 4.6.8. The uniChange() procedure shown in figure 4.6.9 was removed from my program as the program now takes the unicode value from file instead of the next item in uniList.

The createButton() function was also changed, as shown in Photo Evidence 3.1.JPG. The parameter “values” is now used to pass data about the current profile into the function as a list. This data is then used to apply the appropriate colour schemes by using the corresponding indexes of “values” as the attributes bg, fg, activebackground and activeforeground of the tkinter button object. The parameter “uniVal” has also been removed, and the elif statement on line 85 has been changed to identify the “Unicode” key using the value of the variable “val”, rather than “uniVal”.  The button created in this case now calls the procedure send(), using the unicode character at index 5 of the “values” as a parameter.

```
91
92  def qwertyBoard(values, currentProfileName): # creates a tkinter window object containing a keyboard interface with a qwerty layout
93      height1 = 5
94      width1 = 6
95      width2 = 3
96      height6 = 7
97      home = Tk()
98      home.title("Keyboard: " + currentProfileName)
99      home.geometry("800x480")
100     home.resizable(False, False)
101     home.attributes("-fullscreen", True)
102     home["bg"]=values[1]
103     row1 = Frame(home, bg=values[1])
104     row1.pack(side=TOP, anchor=W)
105     row2 = Frame(home)
106     row2.pack(side=TOP, anchor=W)
107     row3 = Frame(home)
108     row3.pack(side=TOP, anchor=W)
109     row4 = Frame(home)
110     row4.pack(side=TOP, anchor=W)
111     row5 = Frame(home)
112     row5.pack(side=TOP, anchor=W)
113     row6 = Frame(home)
114     row6.pack(side=TOP, anchor=W)
```

**Figure 5.5.3.3 - The first section of the procedure qwertyBoard()**

This is the first section of the procedure qwertyBoard(), which loads the keyboard with a qwerty layout. The majority of the procedure is the same as the procedure mainBoard() that has since been removed. The procedure how passes in the parameters “values” and “currentProfileName”. The parameter of home.title() on line 98 has been changed to include the name of the profile currently selected into the name of the window. Line 102 has also been changed  to use the background colour stored in the list “values”, rather than using a global variable “bg”. The global variable “bg” was also replaced with an index of “values” on line 103

The remainder of the procedure remains largely unchanged from that of mainBoard(). The only changes are that the parameter “values” is passed into createButton() each time it is called and that the parameter “uniVal” has been removed from the definition of the “Unicodekey” object on line 187. This code is shown in Photo Evidence 3.2.1 to 3.2.3.

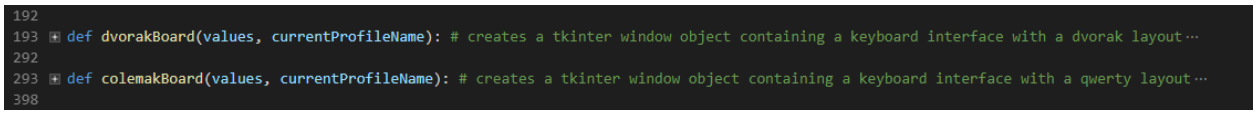


**Figure 5.5.3.4 - A minimised version of the procedures containing the remaining two keyboard layouts**

The remaining two keyboard layouts were implemented using similar procedures. The only change made to the procedure dvorakBoard(), shown in Photo Evidence 3.3.1 to 3.3.3, was the order in which the objects representing the keys were defined and the value of the parameter “row”. The only effect these changes had was the position of the keys. The procedure colemakBoard(), shown in Photo Evidence 3.4.1 to 3.4.3, was changed in a similar way to dvorakBoard(), with differing values of the parameter “row” and a different order in which objects were defined. In addition to these changes, the “CapsKey” button object on line 355 was replaced with a “LeftBackKey” button object. This object had similar attributes to the “BackKey” object except from the values of “row” and “width”. This change can be seen in Photo Evidence 3.4.2.

```
399
400    Profiles = open("Saves/Profiles.txt", "r")
401    currentProfileName = Profiles.readline()
402    currentProfileName = currentProfileName.replace("\n", "")
403    currentProfile = open("Saves/"+ currentProfileName +".txt", "rb")
404    values = currentProfile.readlines()
405
```

**Figure 5.5.3.5 - The section of the keyboard software that accesses data from file**

This section of the program accesses the necessary data from file. It does this by opening the file “Profiles.txt” located in the “Saves” folder, representing the file as the object “Profiles”. The first line of this file is then read and stored, as a string, under the variable “currentProfileName”. The newline character is then replaced with an empty string, thus removing it from the string. This name is then used to open the corresponding text file in the “Saves” folder and this file is represented by the object “currentProfile”. The data from this file is then read, creating a list of strings, which is assigned to the variable “values”, each encoded using the UTF-8 standard.

```
405
406    for i in range (0, len(values)):
407        newVal = values[i]
408        newVal = newVal.decode("utf-8")
409        newVal = newVal.replace("\n", "")
410        newVal = newVal.replace("\r","")
411        values[i] = newVal
412
```

**Figure 5.5.3.6 - The section of the keyboard software that formats the data read from file**

This section of the program that formats and decodes the contents of the list “values”. It does this by iterating for each index of the list. For each index, the program takes the string stored at that position in the list and assigns it to the variable “newVal”. The program then decodes this using the UTF-8 standard and replaces the characters “\n” and “\r” with empty strings, thus removing them from “newVal”. The item in the list is then replaced with the newly formatted string in newVal.

```
412
413     if "dvorak" in values[0]:
414         dvorakBoard(values, currentProfileName)
415     elif "colemak" in values[0]:
416         colemakBoard(values, currentProfileName)
417     else:
418         qwertyBoard(values, currentProfileName)
419
```

**Figure 5.5.3.7 - The section of the keyboard software that calculates the correct layout of the keyboard**

This section branches calculates the correct keyboard layout and calls the corresponding procedure. It does this by checking the first item in values. If “dvorak” is contained within the string, the program calls “dvorakBoard()”. If this is not the case but “colemak” is contained within the string, the program calls colemakBoard(). If neither value is contained within the string, the layout must be qwerty so the program calls “qwertyBoard()”.

## 5.6 - Evaluation

### 5.6.1 Reduction in scale of cycle 3

Due to time constraints, I was unable to implement my customisation software. I was able to begin outlining the tests I would use evaluate the software, based on the algorithm in section 5.3.3 but was unable to complete it. I intend to complete the testing and implementation of a customisation software at a later date. The tests that made up the partially completed testing table are shown below.

| Test ID | Test | Reasoning | Expected Outcome |
|---|---|---|---|
| 3.2.1 | Run the customisation software and enter the the value “test” | To test that the program recognises that this is not a valid input | The program should print the string “That is not a valid option” |
| 3.2.2 | After test 3.2.1, enter the value “REPEAT” | To test that the program can repeat | The program should ask the user to select an option |

| | | | |
|---|---|---|---|
| 3.2.3 | Enter the value “1” | To test that the program can call the procedure newProfile() | The program should respond by asking the user to input a name for the new profile |

### 5.6.2 Evaluation of success criteria

Criteria I had aimed to complete:

| Criteria | Evaluation | Explanation |
|---|---|---|
| Modify my keyboard software to allow for a greater number of customisation options. | Pass | I have modified my keyboard software so that it now supports a range of customisation options, as outlined in section 5.2, which can be changed by altering the text files in the “Saves” folder |
| Implement a system to save these customisation options to memory. | Pass | I have implemented a system to store these options as a series of text files. |
| Create a simple command line software application to change the values of the options stored in memory. | Fail | I have designed an algorithm for a software application that would fulfil this criteria but was unable to implement it due to time constraints, as stated in section 5.6.1 |

Criteria I also attempted to satisfy where possible:

| Criteria | Evaluation | Explanation |
|---|---|---|
| Modify my keyboard software to allow it to simulate a number of additional peripherals | Fail | Due to time constraints, it was not practical for me to implement additional peripherals |
| Create and implement a GUI for the customisation software | Fail | Due to time constraints, I was unable to implement the customisation software. I was therefore unable to create a GUI for it. |
| Create a SDK for my system to allow others to more easily create software that is compatible with mine | Fail | Due to time constraints, It was not practical for me to create an SDK. This is because ensuring the device was functional was more important than allowing other applications to interface with it. The file structure used would also make it possible, albeit difficult, for another developer to implement some degree of integration with my keyboard |
| Implement a system to allow my keyboard software to be run automatically when the device is turned on | Fail | Due to time constraints, this is a feature I was unable to implement. The user must instead execute the code saved on the device. |

# 6 - Overall Evaluation

## 6.1 - Evaluation of success criteria

In this section I will evaluate whether my solution meets the success criteria set out in section 1.4.2

### 6.1.1 Criteria relating to the functionality of the keyboard

Essential:

| Feature | Evaluation | Evidence | Explanation | Response |
|---|---|---|---|---|
| Unicode Support | Partially met | Tests 3.1.3, 3.1.4 and 3.1.7 verify that the keyboard has support for unicode characters | The keyboard software does support all unicode characters, but only supports a single value at a time and this character can only be changed by editing a text file. The keyboard is also currently unable to transfer unicode data to other devices. | In the future, I plan to create additional layouts containing a greater number of unicode values. I also plan to use VNC software to allow the keyboard to transfer unicode data to other devices |
| Touch | Met | Test 1.1.3 verify that | The keyboard has a | This criteria has |

| | | | | |
|---|---|---|---|---|
| screen interface | | the keyboard has a working touchscreen, figure 4.7.3.2 shows that the device can output graphical data and test 1.1.4 verifies that the device can detect touch | touch screen, can output graphical data to it and can detect when and where the user has touched it. | been met. |

Additional:

| Feature | Evaluation | Evidence | Explanation | Response |
|---|---|---|---|---|
| API support | Not met | This criteria has not been met | The software managing the keyboard does not currently have an API | In the future, I plan to develop an API for both the software managing the keyboard and the customisation software |
| SDK | Not met | This criteria has not been met | I have not yet developed an SDK for any of the software on my device | In the future, I plan to create SDK software for my device |
| MIDI inputs | Not met | This criteria has not been met | My device cannot currently transfer MIDI data to other devices. This software managing the keyboard does not currently have any way for the user to enter MIDI data | In the future, I plan to look into the use of VNC software as a way to transfer MIDI data to other devices. I also plan to improve my keyboard software to allow it to simulate a MIDI keyboard, allowing the user to input MIDI data. |
| Macros | Not Met | This criteria has not been met | My keyboard software does not currently allow the user to use macros. | In the future, I plan to modify the keyboard software to incorporate macro keys on the top row of the display. I also plan to use a file system to save macros into text files, similar to the system currently in play |

| | | | | |
|---|---|---|---|---|
| | | | | to save profiles |

### 6.1.2 Criteria relating to the usability of the keyboard

Essential:

| Feature | Evaluation | Evidence | Explanation | Response |
|---|---|---|---|---|
| GUI | Partially met | Video Evidence 3.1, 3.2 and 3.3 all show that the device can load a functioning GUI. | The program is able to load a functioning GUI, with all necessary keys but the user is required to open and execute the python file “keyboard.py”. This is possible to do using only the touchscreen but this process would likely be unclear to a new user. The buttons in the GUI also pass their values to the placeholder procedure “send()” which currently only outputs the character to console. | In the future, I plan to convert “keyboard.py” to an executable and modify the operating system of the pi to run this executable whenever the device is powered on. I also plan to modify the procedure “send()” to send the character to the other device using VNC software. |

Additional:

| Feature | Evaluation | Evidence | Explanation | Response |
|---|---|---|---|---|
| Customisable keyboard layouts | Partially met | Tests 3.1.1, 3.1.4 and 3.1.7 verify that the keyboard is able to load a variety of layouts and colour schemes | The coloration of the keyboard interface can be greatly customised and the keyboard can support multiple predefined layouts. The keyboard does not allow the user to create their own layouts and any changes they do make require the software to be restarted to take effect | In the future, I plan to improve the implementation of my keyboard software to support any possible layout, rather than a select number of predefined layouts. I also plan to implement a refresh button, which checks for any changes to the settings and reloads the keyboard window if necessary. |
| Wrist rest | Not met | This criteria has not been met | The device does not currently have a case, making it difficult to attach a wrist rest. The touchscreen is also much smaller than initially planned, meaning a wrist rest would have little effect on the usability of the device. | In the future, I plan to use 3D printing to create a case for the device. This would allow me to incorporate a wrist rest into the case. |
| Customisation software | Not met | This criteria has not been met | The device does currently have any software dedicated to the customisation of the keyboard interface. | In the future, I plan to improve the design of the customisation software designed in section 5.3.3 to support a wider range of features, such as a |

| | | | | |
|---|---|---|---|---|
| | | | | GUI, customisable layouts and MIDI keys. I will then implement the software with these additional features |

## 6.2 - Evaluation of the maintainability and limitations

### 6.2.1 Maintainability of my keyboard software

The vast majority of the code within the software has been encapsulated using subroutines. This reduces the likelihood of errors and aids the process of diagnosing any errors that do occur. One example of this is the function createButton() which creates a tkinter button object using the parameters passed into the function. This eliminates the need to directly use the tkinter Button() function and allows the size of large numbers of buttons to be defined at once.

The definition of each subroutine has been commented with an explanation of the subroutine. This improves the readability of my program, although the contents of each subroutine do not contain any comments which somewhat limits these improvements. The maintainability of the software is further limited by the use of global variables, which was necessary to correctly implement keys such as “Caps Lock” but may cause unforeseen errors, particularly when switching layouts or when implementing the refresh button mentioned on page 93.

The names of variables and subroutines relate to their purpose helping to further improve the legibility of the program.  All button objects follow a similar format, with the name of the button followed by “Key” and all layout procedures follow a similar format, with the name of the layout followed by “board”. This allows them to be easily identified, without the name becoming impractically long.

Within the keyboard software, the definitions of all buttons are grouped by row, with each row separated by an empty line. This improves maintainability by making it easier to identify the definition of specific keys.

The use of procedures to represent each keyboard layout also negatively impacts the maintainability of the program, as large amounts of the code is repeated. This implementation also means the entire function needs to be repeated in order for new layouts to be added.

### 6.2.2 Limitations of my solution

The most significant limitation of my solution by far is the inability to send data to other devices. Implementing a system to transfer data to other devices would be the most important task moving forward as it would allow my device to function as initially intended.

Another limitation of my program is the lack of a customisation software as the current system requires the user to connect an external keyboard and edit a text file. Moving forward, I would like to implement customisation software that uses a GUI to remove the need for the user to connect a keyboard.

The hardware that runs my keyboard is another limitation. The lack of an outer case means the device is extremely fragile, the lack of a battery requires the device to be connected to a power outlet while in use and the size if the screen makes the keyboard difficult to use. At a later date, I would like to remedy these issues by 3D printing a case, allowing better protection. A battery, large screen and wrist rest could all be incorporated into this case.

My solution is further limited by the lack of additional simulated peripherals. By modifying the keyboard software, the device could be used for a wider range of purposes, as initially envisioned in section 1.4.1.

# 7 - Final Code

```
from tkinter import *


### global variables: ###
caps = False
shift = False
ctrl = False
alt = False
#########################




def send(val): # sends the value to the other device
    print(val)


def enter(val, capsVal, shiftVal, shiftLabel): # calculates which value to pass into send()
    global shift, caps
    if shift:
        if shiftVal!="":
            send(shiftVal)
        elif shiftLabel!="":
            send(shiftLabel)
        elif capsVal != "":
            send(capsVal)
        else:
            send(val)
```

```
    elif caps:
        if capsVal != "":
            send(capsVal)
        else:
            send(val)
    else:
        send(val)


def shiftChange(): # changes the value of shift and passes the corresponding value into send()
    global shift
    if shift:
        shift=False
        send("ShiftOff")
    else:
        shift=True
        send("ShiftOn")


def capsChange(): # changes the value of caps and passes the corresponding value into send()
    global caps
    if caps:
        caps=False
        send("CapsOff")
    else:
        caps=True
        send("CapsOn")


def ctrlChange(): # changes the value of ctrl and passes the corresponding value into send()
    global ctrl
    if ctrl:
        ctrl=False
        send("CtrlOff")
    else:
```

```
        ctrl=True
        send("CtrlOn")

def altChange(): # changes the value of alt and passes the corresponding value into send()
    global alt
    if alt:
        alt=False
        send("AltOff")
    else:
        alt=True
        send("AltOn")

def createButton(row, values, val, capsVal="", shiftVal="", shiftLabel="", label="", changeVal="", width=3, height=4, border=2): #
creates, packs and returns a tkinter button object
    if label == "":
        if capsVal!="":
            label = capsVal
        else:
            label = val
    if changeVal == "caps":
        button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border, bg=values[1], fg=values[2],
activebackground = values[3], activeforeground = values[4], anchor=N, command=lambda: capsChange())
    elif changeVal == "shift":
        button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border, bg=values[1], fg=values[2],
activebackground = values[3], activeforeground = values[4], anchor=N, command=lambda: shiftChange())
    elif changeVal == "ctrl":
        button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border, bg=values[1], fg=values[2],
activebackground = values[3], activeforeground = values[4], anchor=N, command=lambda: ctrlChange())
    elif changeVal == "alt":
        button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border, bg=values[1], fg=values[2],
activebackground = values[3], activeforeground = values[4], anchor=N, command=lambda: altChange())
    elif val == "Unicode":
        button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border, bg=values[1], fg=values[2],
activebackground = values[3], activeforeground = values[4], anchor=N, command=lambda: send(values[5]))
```

```
    else:
        button = Button(row, text=shiftLabel+"\n"+label, width=width, height=height, borderwidth=border, bg=values[1], fg=values[2], activebackground = values[3], activeforeground = values[4], anchor=N, command=lambda: enter(val, capsVal, shiftVal, shiftLabel))
    button.pack(side=LEFT)
    return button


def qwertyBoard(values, currentProfileName): # creates a tkinter window object containing a keyboard interface with a qwerty layout
    height1 = 5
    width1 = 6
    width2 = 3
    height6 = 7
    home = Tk()
    home.title("Keyboard: " + currentProfileName)
    home.geometry("800x480")
    home.resizable(False, False)
    home.attributes("-fullscreen", True)
    home["bg"]=values[1]
    row1 = Frame(home, bg=values[1])
    row1.pack(side=TOP, anchor=W)
    row2 = Frame(home)
    row2.pack(side=TOP, anchor=W)
    row3 = Frame(home)
    row3.pack(side=TOP, anchor=W)
    row4 = Frame(home)
    row4.pack(side=TOP, anchor=W)
    row5 = Frame(home)
    row5.pack(side=TOP, anchor=W)
    row6 = Frame(home)
    row6.pack(side=TOP, anchor=W)

    macroFrame = Frame(row1)
    macroFrame.pack(fill=NONE, padx=244, pady=5, side=LEFT)
```

```
MuteKey = createButton(row1, values, "Mute", label="Mute", height=height1, width=width1)

PlayPauseKey = createButton(row1, values, "PlayPause", label="▶/⏯", height=height1, width=width1)

VolDownKey = createButton(row1, values, "VolDown", label="⇩", height=height1, width=width1)

VolUpKey = createButton(row1, values, "VolUp", label="⇧", height=height1, width=width1)


AccentKey = createButton(row2, values, "`", shiftLabel="¬", width=width2-1)

OneKey = createButton(row2, values, "1", shiftLabel="!", width=width2)

TwoKey = createButton(row2, values, "2", shiftLabel="\"", width=width2)

ThreeKey = createButton(row2, values, "3", shiftLabel="£", width=width2)

FourKey = createButton(row2, values, "4", shiftLabel="$", width=width2)

FiveKey = createButton(row2, values, "5", shiftLabel="%", width=width2)

SixKey = createButton(row2, values, "6", shiftLabel="^", width=width2)

SevenKey = createButton(row2, values, "7", shiftLabel="&", width=width2)

EightKey = createButton(row2, values, "8", shiftLabel="*", width=width2)

NineKey = createButton(row2, values, "9", shiftLabel="(", width=width2)

TenKey = createButton(row2, values, "0", shiftLabel=")", width=width2)

MinusKey = createButton(row2, values, "-", shiftLabel="_", width=width2)

EqualKey = createButton(row2, values, "=", shiftLabel="+", width=width2)

BackKey = createButton(row2, values, "Back", label="←", shiftLabel="", width=20)


TabKey = createButton(row3, values, "Tab", width=8)

QKey = createButton(row3, values, "q", capsVal="Q")

WKey = createButton(row3, values, "w", capsVal="W")

EKey = createButton(row3, values,"e", capsVal="E")

RKey = createButton(row3, values,"r", capsVal="R")

TKey = createButton(row3, values,"t", capsVal="T")

YKey = createButton(row3, values, "y", capsVal="Y")

UKey = createButton(row3, values, "u", capsVal="U")

IKey = createButton(row3, values, "i", capsVal="I")

OKey = createButton(row3, values, "o", capsVal="O")

PKey = createButton(row3, values, "p", capsVal="P")

SqBrOKey = createButton(row3, values, "[", shiftLabel="{")
```

```
SqBrCKey = createButton(row3, values, "]", shiftLabel="}")

HashKey = createButton(row3, values, "#", shiftLabel="~", width=10)


CapsKey = createButton(row4, values, "Caps Lock", width=10, changeVal="caps")

AKey = createButton(row4, values, "a", capsVal="A")

SKey = createButton(row4, values, "s", capsVal="S")

DKey = createButton(row4, values, "d", capsVal="D")

FKey = createButton(row4, values, "f", capsVal="F")

GKey = createButton(row4, values, "g", capsVal="G")

HKey = createButton(row4, values, "h", capsVal="H")

JKey = createButton(row4, values, "j", capsVal="J")

KKey = createButton(row4, values, "k", capsVal="K")

LKey = createButton(row4, values, "l", capsVal="L")

SemiColonKey = createButton(row4, values, ";", shiftLabel=":")

ApostropheKey = createButton(row4, values, "'", shiftLabel="@")

EnterKey = createButton(row4, values, "\n", "↵", width=14)


ShiftLKey = createButton(row5, values, "Shift", width=8, changeVal="shift")

BSlashKey = createButton(row5, values, "\\", shiftLabel="|")

ZKey = createButton(row5, values, "z", capsVal="Z")

XKey = createButton(row5, values, "x", capsVal="X")

CKey = createButton(row5, values, "c", capsVal="C")

VKey = createButton(row5, values, "v", capsVal="V")

BKey = createButton(row5, values, "b", capsVal="B")

NKey = createButton(row5, values, "n", capsVal="N")

MKey = createButton(row5, values, "m", capsVal="M")

CommaKey = createButton(row5, values, ",", shiftLabel="<")

StopKey = createButton(row5, values, ".", shiftLabel=">")

FSlashKey = createButton(row5, values, "/", shiftLabel="?")

shiftRKey = createButton(row5, values, "Shift", width=20, changeVal="shift")


CtrlLKey = createButton(row6, values, "Ctrl", width = 6, height=height6, changeVal="ctrl")
```

```
    WinLKey = createButton(row6, values, "Win", height=height6)
    AltLKey = createButton(row6, values, "Alt", height=height6, changeVal="alt")
    SpaceKey = createButton(row6, values, " ", label="_____", width=41, height=height6)
    UnicodeKey = createButton(row6, values, "Unicode", width=5, height=height6)
    WinRKey = createButton(row6, values, "Win", height=height6)
    MenuKey = createButton(row6, values, "Menu", height=height6)
    CtrlRKey = createButton(row6, values, "Ctrl", width=6, height=height6, changeVal="ctrl")

    home.mainloop()

def dvorakBoard(values, currentProfileName): # creates a tkinter window object containing a keyboard interface with a dvorak layout
    height1 = 5
    width1 = 6
    width2 = 3
    height6 = 7
    home = Tk()
    home.title("Keyboard: " + currentProfileName)
    home.geometry("800x480")
    home.resizable(False, False)
    home.attributes("-fullscreen", True)
    home["bg"]=values[1]
    row1 = Frame(home, bg=values[1])
    row1.pack(side=TOP, anchor=W)
    row2 = Frame(home)
    row2.pack(side=TOP, anchor=W)
    row3 = Frame(home)
    row3.pack(side=TOP, anchor=W)
    row4 = Frame(home)
    row4.pack(side=TOP, anchor=W)
    row5 = Frame(home)
    row5.pack(side=TOP, anchor=W)
```

```
row6 = Frame(home)

row6.pack(side=TOP, anchor=W)


macroFrame = Frame(row1)

macroFrame.pack(fill=NONE, padx=244, pady=5, side=LEFT)

MuteKey = createButton(row1, values, "Mute", label="Mute", height=height1, width=width1)

PlayPauseKey = createButton(row1, values, "PlayPause", label="▶/⏯", height=height1, width=width1)

VolDownKey = createButton(row1, values, "VolDown", label="⇩", height=height1, width=width1)

VolUpKey = createButton(row1, values, "VolUp", label="⇧", height=height1, width=width1)


AccentKey = createButton(row2, values, "`", shiftLabel="¬", width=width2-1)

OneKey = createButton(row2, values, "1", shiftLabel="!", width=width2)

TwoKey = createButton(row2, values, "2", shiftLabel="\"", width=width2)

ThreeKey = createButton(row2, values, "3", shiftLabel="£", width=width2)

FourKey = createButton(row2, values, "4", shiftLabel="$", width=width2)

FiveKey = createButton(row2, values, "5", shiftLabel="%", width=width2)

SixKey = createButton(row2, values, "6", shiftLabel="^", width=width2)

SevenKey = createButton(row2, values, "7", shiftLabel="&", width=width2)

EightKey = createButton(row2, values, "8", shiftLabel="*", width=width2)

NineKey = createButton(row2, values, "9", shiftLabel="(", width=width2)

TenKey = createButton(row2, values, "0", shiftLabel=")", width=width2)

MinusKey = createButton(row2, values, "-", shiftLabel="_", width=width2)

EqualKey = createButton(row2, values, "=", shiftLabel="+", width=width2)

BackKey = createButton(row2, values, "Back", label="⟵", shiftLabel="", width=20)


TabKey = createButton(row3, values, "Tab", width=8)

CommaKey = createButton(row3, values, ",", shiftLabel="<")

StopKey = createButton(row3, values, ".", shiftLabel=">")

PKey = createButton(row3, values, "p", capsVal="P")

YKey = createButton(row3, values, "y", capsVal="Y")

FKey = createButton(row3, values, "f", capsVal="F")

GKey = createButton(row3, values, "g", capsVal="G")
```

```
CKey = createButton(row3, values, "c", capsVal="C")

RKey = createButton(row3, values,"r", capsVal="R")

LKey = createButton(row3, values, "l", capsVal="L")

SqBrOKey = createButton(row3, values, "[", shiftLabel="{")

SqBrCKey = createButton(row3, values, "]", shiftLabel="}")

FSlashKey = createButton(row3, values, "/", shiftLabel="?")

HashKey = createButton(row3, values, "#", shiftLabel="~", width=10)


CapsKey = createButton(row4, values, "Caps Lock", width=10, changeVal="caps")

AKey = createButton(row4, values, "a", capsVal="A")

OKey = createButton(row4, values, "o", capsVal="O")

EKey = createButton(row4, values,"e", capsVal="E")

UKey = createButton(row4, values, "u", capsVal="U")

IKey = createButton(row4, values, "i", capsVal="I")

DKey = createButton(row4, values, "d", capsVal="D")

HKey = createButton(row4, values, "h", capsVal="H")

TKey = createButton(row4, values,"t", capsVal="T")

NKey = createButton(row4, values, "n", capsVal="N")

SKey = createButton(row4, values, "s", capsVal="S")

ApostropheKey = createButton(row4, values, "'", shiftLabel="@")

EnterKey = createButton(row4, values, "\n", "↵", width=14)


ShiftLKey = createButton(row5, values, "Shift", width=8, changeVal="shift")

BSlashKey = createButton(row5, values, "\\", shiftLabel="|")

SemiColonKey = createButton(row5, values, ";", shiftLabel=":")

QKey = createButton(row5, values, "q", capsVal="Q")

JKey = createButton(row5, values, "j", capsVal="J")

KKey = createButton(row5, values, "k", capsVal="K")

XKey = createButton(row5, values, "x", capsVal="X")

BKey = createButton(row5, values, "b", capsVal="B")

MKey = createButton(row5, values, "m", capsVal="M")

WKey = createButton(row5, values, "w", capsVal="W")
```

```
    VKey = createButton(row5, values, "v", capsVal="V")

    ZKey = createButton(row5, values, "z", capsVal="Z")

    shiftRKey = createButton(row5, values, "Shift", width=20, changeVal="shift")


    CtrlLKey = createButton(row6, values, "Ctrl", width = 6, height=height6, changeVal="ctrl")

    WinLKey = createButton(row6, values, "Win", height=height6)

    AltLKey = createButton(row6, values, "Alt", height=height6, changeVal="alt")

    SpaceKey = createButton(row6, values, " ", label="_____", width=41, height=height6)

    UnicodeKey = createButton(row6, values, "Unicode", width=5, height=height6)

    WinRKey = createButton(row6, values, "Win", height=height6)

    MenuKey = createButton(row6, values, "Menu", height=height6)

    CtrlRKey = createButton(row6, values, "Ctrl", width=6, height=height6, changeVal="ctrl")


    home.mainloop()


def colemakBoard(values, currentProfileName): # creates a tkinter window object containing a keyboard interface with a qwerty layout

    height1 = 5

    width1 = 6

    width2 = 3

    height6 = 7

    home = Tk()

    home.title("Keyboard: " + currentProfileName)

    home.geometry("800x480")

    home.resizable(False, False)

    home.attributes("-fullscreen", True)

    home["bg"]=values[1]

    row1 = Frame(home, bg=values[1])

    row1.pack(side=TOP, anchor=W)

    row2 = Frame(home)

    row2.pack(side=TOP, anchor=W)

    row3 = Frame(home)
```

```
row3.pack(side=TOP, anchor=W)

row4 = Frame(home)

row4.pack(side=TOP, anchor=W)

row5 = Frame(home)

row5.pack(side=TOP, anchor=W)

row6 = Frame(home)

row6.pack(side=TOP, anchor=W)


macroFrame = Frame(row1)

macroFrame.pack(fill=NONE, padx=244, pady=5, side=LEFT)

MuteKey = createButton(row1, values, "Mute", label="Mute", height=height1, width=width1)

PlayPauseKey = createButton(row1, values, "PlayPause", label="▶/⏯", height=height1, width=width1)

VolDownKey = createButton(row1, values, "VolDown", label="⇩", height=height1, width=width1)

VolUpKey = createButton(row1, values, "VolUp", label="⇧", height=height1, width=width1)


AccentKey = createButton(row2, values, "`", shiftLabel="¬", width=width2-1)

OneKey = createButton(row2, values, "1", shiftLabel="!", width=width2)

TwoKey = createButton(row2, values, "2", shiftLabel="\"", width=width2)

ThreeKey = createButton(row2, values, "3", shiftLabel="£", width=width2)

FourKey = createButton(row2, values, "4", shiftLabel="$", width=width2)

FiveKey = createButton(row2, values, "5", shiftLabel="%", width=width2)

SixKey = createButton(row2, values, "6", shiftLabel="^", width=width2)

SevenKey = createButton(row2, values, "7", shiftLabel="&", width=width2)

EightKey = createButton(row2, values, "8", shiftLabel="*", width=width2)

NineKey = createButton(row2, values, "9", shiftLabel="(", width=width2)

TenKey = createButton(row2, values, "0", shiftLabel=")", width=width2)

MinusKey = createButton(row2, values, "-", shiftLabel="_", width=width2)

EqualKey = createButton(row2, values, "=", shiftLabel="+", width=width2)

BackKey = createButton(row2, values, "Back", label="←", shiftLabel="", width=20)


TabKey = createButton(row3, values, "Tab", width=8)

QKey = createButton(row3, values, "q", capsVal="Q")
```

```
WKey = createButton(row3, values, "w", capsVal="W")

FKey = createButton(row3, values, "f", capsVal="F")

PKey = createButton(row3, values, "p", capsVal="P")

GKey = createButton(row3, values, "g", capsVal="G")

JKey = createButton(row3, values, "j", capsVal="J")

LKey = createButton(row3, values, "l", capsVal="L")

UKey = createButton(row3, values, "u", capsVal="U")

YKey = createButton(row3, values, "y", capsVal="Y")

SemiColonKey = createButton(row3, values, ";", shiftLabel=":")

SqBrOKey = createButton(row3, values, "[", shiftLabel="{")

SqBrCKey = createButton(row3, values, "]", shiftLabel="}")

HashKey = createButton(row3, values, "#", shiftLabel="~", width=10)



LeftBackKey = createButton(row4, values, "Back", label="←", shiftLabel="", width=10)

AKey = createButton(row4, values, "a", capsVal="A")

RKey = createButton(row4, values,"r", capsVal="R")

SKey = createButton(row4, values, "s", capsVal="S")

TKey = createButton(row4, values,"t", capsVal="T")

DKey = createButton(row4, values, "d", capsVal="D")

HKey = createButton(row4, values, "h", capsVal="H")

NKey = createButton(row4, values, "n", capsVal="N")

EKey = createButton(row4, values,"e", capsVal="E")

IKey = createButton(row4, values, "i", capsVal="I")

OKey = createButton(row4, values, "o", capsVal="O")

ApostropheKey = createButton(row4, values, "'", shiftLabel="@")

EnterKey = createButton(row4, values, "\n", "↵", width=14)


ShiftLKey = createButton(row5, values, "Shift", width=8, changeVal="shift")

BSlashKey = createButton(row5, values, "\\", shiftLabel="|")

ZKey = createButton(row5, values, "z", capsVal="Z")

XKey = createButton(row5, values, "x", capsVal="X")
```

```
    CKey = createButton(row5, values, "c", capsVal="C")
    VKey = createButton(row5, values, "v", capsVal="V")
    BKey = createButton(row5, values, "b", capsVal="B")
    KKey = createButton(row5, values, "k", capsVal="K")
    MKey = createButton(row5, values, "m", capsVal="M")
    CommaKey = createButton(row5, values, ",", shiftLabel="<")
    StopKey = createButton(row5, values, ".", shiftLabel=">")
    FSlashKey = createButton(row5, values, "/", shiftLabel="?")
    shiftRKey = createButton(row5, values, "Shift", width=20, changeVal="shift")



    CtrlLKey = createButton(row6, values, "Ctrl", width = 6, height=height6, changeVal="ctrl")
    WinLKey = createButton(row6, values, "Win", height=height6)
    AltLKey = createButton(row6, values, "Alt", height=height6, changeVal="alt")
    SpaceKey = createButton(row6, values, " ", label="_____", width=41, height=height6)
    UnicodeKey = createButton(row6, values, "Unicode", width=5, height=height6)
    WinRKey = createButton(row6, values, "Win", height=height6)
    MenuKey = createButton(row6, values, "Menu", height=height6)
    CtrlRKey = createButton(row6, values, "Ctrl", width=6, height=height6, changeVal="ctrl")



    home.mainloop()



Profiles = open("Saves/Profiles.txt", "r")
currentProfileName = Profiles.readline()
currentProfileName = currentProfileName.replace("\n", "")
currentProfile = open("Saves/"+ currentProfileName +".txt", "rb")
values = currentProfile.readlines()
```

```
for i in range (0, len(values)):
    newVal = values[i]
    newVal = newVal.decode("utf-8")
    newVal = newVal.replace("\n", "")
    newVal = newVal.replace("\r","")
    values[i] = newVal


if "dvorak" in values[0]:
    dvorakBoard(values, currentProfileName)
elif "colemak" in values[0]:
    colemakBoard(values, currentProfileName)
else:
    qwertyBoard(values, currentProfileName)
```