# Simplex

Edmond Cheng

May 3, 2024

# Contents

# Chapter 1

# Pre-Implementation

## 1.1 Defining The Problem

Currently, there is a lack of text editors specifically for software development available for MacOS that fits its updated design style. The goal simplex is to be a modern alternative to the available code editors for MacOS.

The Simplex application aims to be an intuitive graphical text editor with some features of an IDE. It will be able to directly alter source files and compile or run them with a simple user input. An example is the source file editor, Visual Studio Code which allows users to edit the contents of a source file and then perform a build process with the press of a GUI button. To make improve the user experience the application will also include a preview feature that allows the user to see the end results of a build without leaving the application, reducing the hassle of switching between applications when developing software. All in all, the Simplex application will provide a GUI for editing source code, providing useful quality-of-life features, such as a view that shows source code execution results.

### 1.1.1 Alternative Technologies

An alternative to the Simplex application would be Visual Studio Code (VSC). It is a source code editor available on MacOS that is lightweight and has a lot of community support. Many of its features are planned for Simplex, including code editing and execution capabilities. Like VSC, Simplex will provide a space that allows users to write code based on a source file that they select. Another similarity is that it will also provide a method to save, compile and run the code. that they have written. Simplex and VSC will be functionally as the goal of Simplex is to also be a source code editor.
However, it is not programmed in Swift, being mainly made with TypeScript. This means that MacOS updates have the potential to remove VSC's capabilities whilst Swift, being supported by Apple, has a lower chance of deprecation. As Simplex will be programmed in Swift, it has a lower chance of being deprecated soon. VSC is an available alternative to Simplex, with both performing similar functions, though it is not fully optimised for MacOS.

## 1.2 Social and Ethical Issues

### 1.2.1 Intellectual Property

Intellectual property refers to the ownership of a work that was produced with mental labour. For example, a piece of software independently made would be the intellectual property of the software developer that made it.

For this project, there are two main considerations to be made about intellectual property, the ownership of code adapted from outside sources, such as Stack Overflow and ownership of the application being developed, being the Simplex IDE. It is important that intellectual proerty is considered to not only protect the rights to this application but also to guard against any problems related to intellectual property that could occur from using code from external sources.

#### 1.2.1.1 External Sources

Code will definitely be adapted from external sources and as such, the licensing of any code used must be clearly identified as it will be the intellectual property of another software developer. A source that will be used is Stack Overflow, which has all content published there under the CC BY-SA license according to their Terms of Service. This means that all content, which will most likely be code snippets, used from this website can be freely used under the condition that appropriate credit is given and any derivative versions of the code must be distributed with a similar [1] (or the same) license to the CC BY-SA license. However, it is ambiguous what constitutes a derivative work, especially if the code snippet is used in a much larger project, only elements dependent on that snippet need to be distributed under a similar license. The result of this is that, with regard to Stack Overflow content, the Simplex application can be distributed under any license, as long as content dependent on adapted code snippets has a similar license to the CC BY-SA license.
For other sources, such as a YouTube tutorial, the license for code should be properly identified, to properly attribute and use the code. Generally, code snippets can fall under Scènes à faire or Merger Doctrine, which say that given an idea or a combination of them, works produced based on the idea/s must be very similar and thus not fall under copyright.
Therefore, code that has been adapted from external sources can be used in this project as Stack Overflow content falls under the CC BY-SA license and code snippets from other sources are small enough to fit the Merger Doctrine, provided that the conditions of licenses are fulfilled.

#### 1.2.1.2 Application License

The Simplex application will be the fruit of mental labour and, as a result, become intellectual property. To properly protect the usage of this project, a suitable license must be chosen. For now, the license that will cover most of the project is the MIT license, with some CC BY-SA licensed parts (see Section 1.2.1.1). This license allows users to download and use this application free of charge, share the code free of charge and even commercialise it, with no warranty provided and waive liability. This means

[1] Similar licenses include the GPL and the Free Art License.

that the application can be optimistically open-source shareware with the added option of future versions being commercialised without breaking the license.

### 1.2.2 Privacy

Privacy issues relate to the protection of an individual's personal information.

In consideration of privacy issues, Simplex will not collect any personal information overtly, through prompts and data fields, or covertly, like through cookies. However, Simplex will have access to the user's file system in order to read and write them. To protect the user's privacy in this area, Simplex will only access files that the user specifies and will do so locally, with no interaction with the internet. This ensures that the user's data stays only on their device. To protect the user's privacy, Simplex will not require any personal information, and its interactions with the local file system of a device will stay local.

### 1.2.3 Language

In this context, there are two main types of languages, human languages and programming languages. The main issue relating to both types of languages is the support the Simplex application will have for them.

#### 1.2.3.1 Human Languages

Human languages are languages that are spoken, written and read by humans, for the purpose of communication.

As there are many human languages, Simplex should aim to support as many of them as possible because it is the responsibility of a software developer to be inclusive. However, it is not feasible to provide full support for all languages, so the development of Simplex will focus on support for English first. If time permits, localisation (adjusting aspects of the software to fit a given geographical region) features will be considered for implmentation to fulfill the responsibilities of a developer. Therefore, the main scope of the Simplex application will provide support for English, with localisation features being considered if the time allows for the scope to be extended.

#### 1.2.3.2 Programming Languages

Programming languages are a system of words and syntax that provide an interface for humans to program computers.

There are many programming languages available, from low-level languages like Assembly to high-level languages like Scratch. As the Simplex programming language is designed to input and run computer code that is written in some of these languages, it is appropriate to consider how design choices for this application are inclusive to different programmers. To be inclusive, Simplex will offer some support for Python and C, and provide a way for users to implement their own support for other languages. This allows Simplex to support some programming languages while not being exclusive to others.

### 1.2.4 Open Sourcing Zed

*The Creators of the Atom Code Editor Open-Source Zed, Their New Rust-Based High-Performance Editor* by Bruno Couriol on February 25, 2024

Zed is a new code editor for MacOS released in 2023 that is being open-sourced this year. It is being distributed under the strong copyleft licence GNU General Purpose License (GNU GPL). This allows users to freely use, modify and share the software while making the source code completely available. As a consequence, developers interested in Zed will be able to verify its safety and make contributions to improve the code editor. Another benefit is that the code in this application will be made fully available to the public, allowing tools used in Zed to benefit other applications. An example of this is Zed's GPUI which allows it to use the GPU to accelerate the rendering of GUI which significantly improves application performance. The open-sourcing of Zed will increase interest in the application as well as serve as contribution to the software development community.

## 1.3 Designing

Figures 1.4, 1.7, 1.6 and 1.8 can be found here or using this link: https://lucid.app/lucidchart/d7a18249-07a1-4dc4-a6ae-5872c5ff3270/edit?viewport_loc=-2851%2C-1398%2C4528%2C2189%2C0_0&invitationId=inv_ae026a95-d62d-49ed-8afb-8cf0dea00248

### 1.3.1 CASE Tools

Computer Assisted Software Engineering (CASE) tools are pieces of software that make it easier for a software developer to build a product, during both the design and implementation phases.

Some CASE tools that will be used are:

- an IDE: Xcode
- a diagram editor: Lucidchart
- a spreadsheet program: Microsoft Excel

### 1.3.2 Time Management

#### 1.3.2.1 Proposed Gantt Chart

| | Term 4 | | | | | Holidays | | |
|---|---|---|---|---|---|---|---|---|
| **Proposed Gantt Chart** | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | December | January | February |
| Define the problem | | | | | | | | |
| Understand the problem | | | | | | | | |
| Present the idea | | | | | | | | |
| Implement the solution | | | | | | | | |
| Test and Evaluate the solution (Beta\|Alpha) | | | | | | | | |
| Logbook | | | | | | | | |

| | Term 1 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 | Week 10 |
| Define the problem | | | | | | | | | | |
| Understand the problem | | | | | | | | | | |
| Implement the solution | | | | | | | | | | |
| Test and Evaluate the solution (Beta\|Alpha) | | | | | | | | | | |
| | | | | | | | | | | |
| Logbook | | | | | | | | | | |

| | Term 2 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 | Week 10 |
| Define the problem | | | | | | | | | | |
| Understand the problem | | | | | | | | | | |
| Implement the solution | | | | | | | | | | |
| Test and Evaluate the solution (Beta\|Alpha) | | | | | | | | | | |
| | | | | | | | | | | |
| Logbook | | | | | | | | | | |

| | Term 3 | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Week 1 | Week 2 | Week 3 | Week 4 | Week 5 | Week 6 | Week 7 | Week 8 | Week 9 | Week 10 |
| Define the problem | | | | | | | | | | |
| Understand the problem | | | | | | | | | | |
| Implement the solution | | | | | | | | | | |
| Test and Evaluate the solution (Beta\|Alpha) | | | | | | | | | | |

Figure 1.1:

#### 1.3.2.2 Actual Progress Gantt Chart

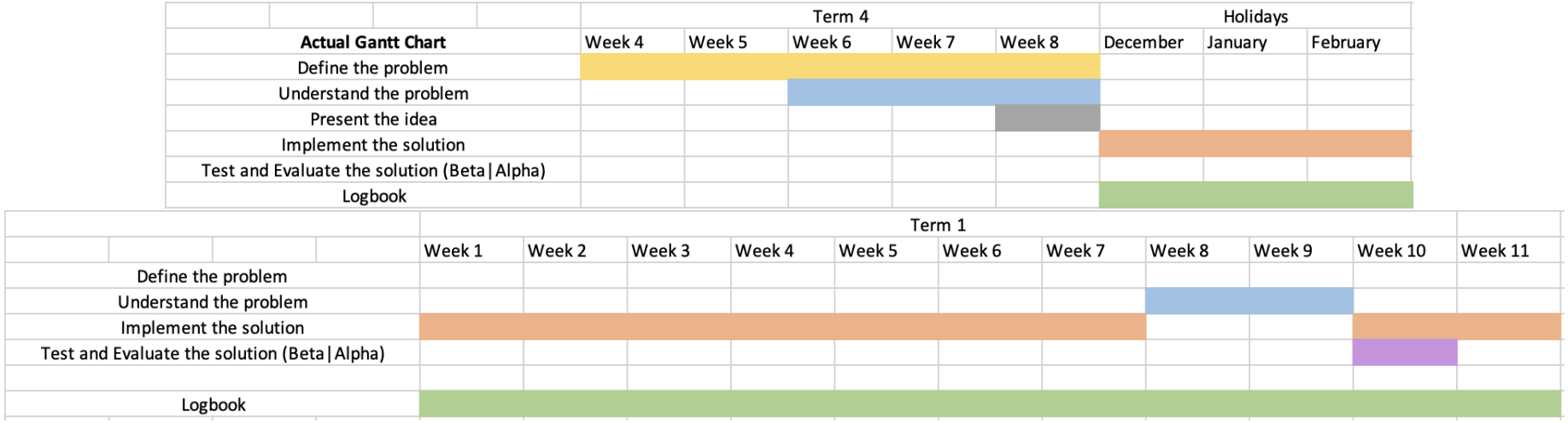



Figure 1.2: This Gantt chart shows how I actually spent my time

### 1.3.3 Top-level System Design

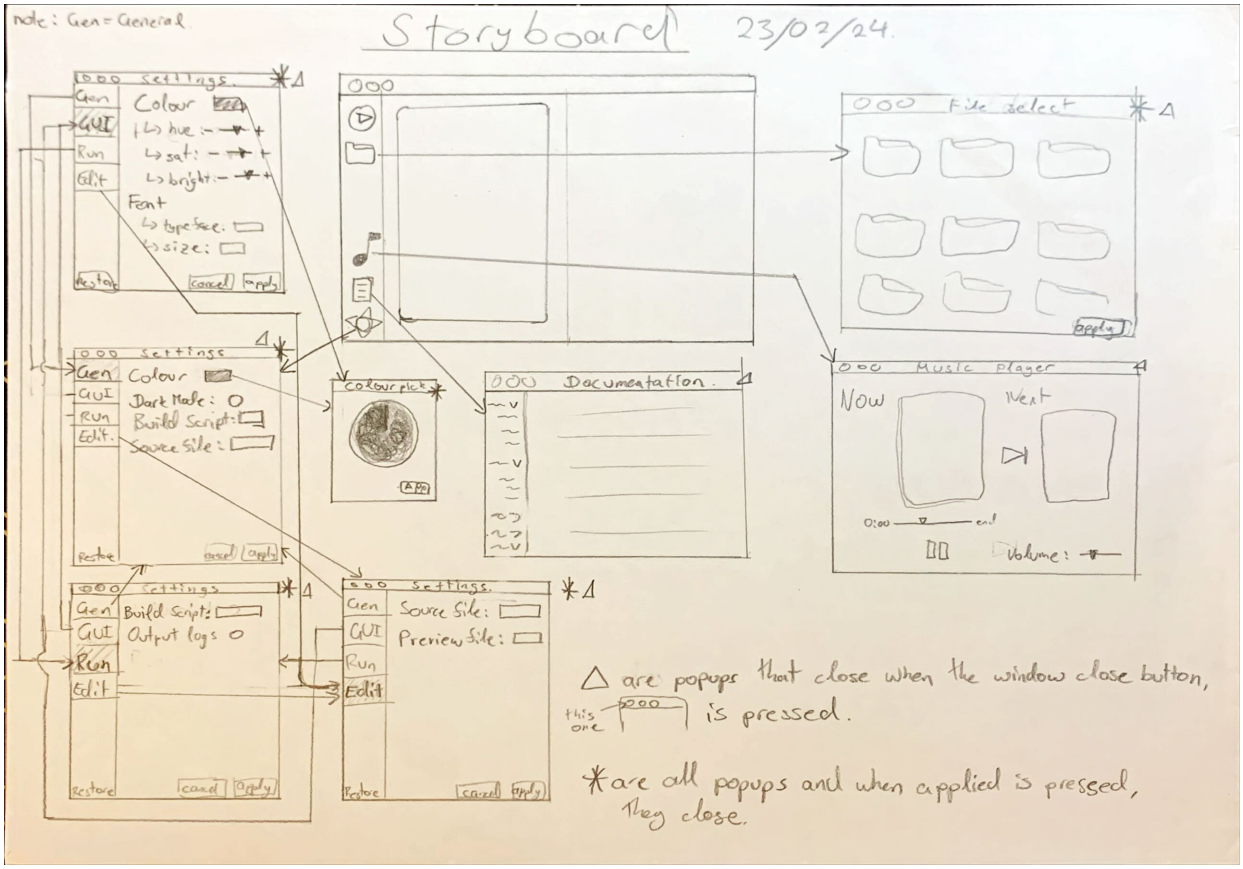



Figure 1.3: An initial storyboard for the full Simplex application

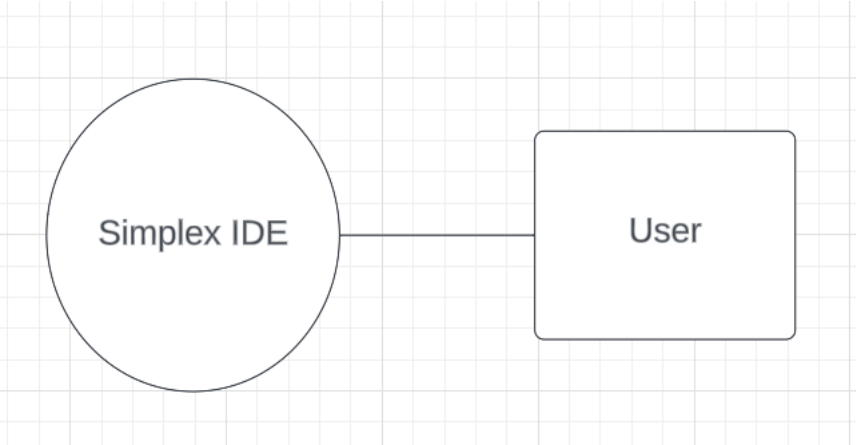



Figure 1.4: A context diagram for the Simplex application

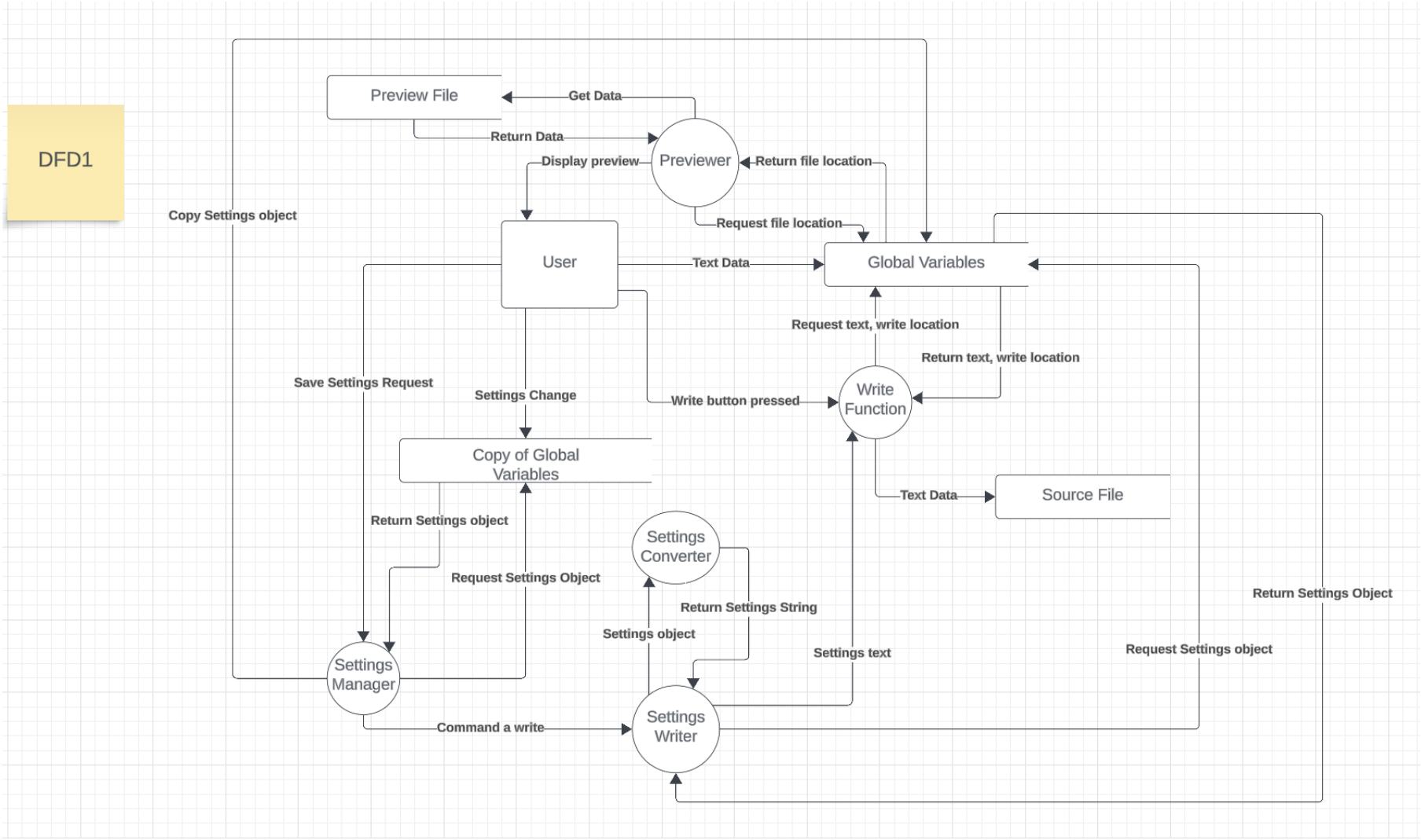



Figure 1.5: A data flow diagram for the Simplex application

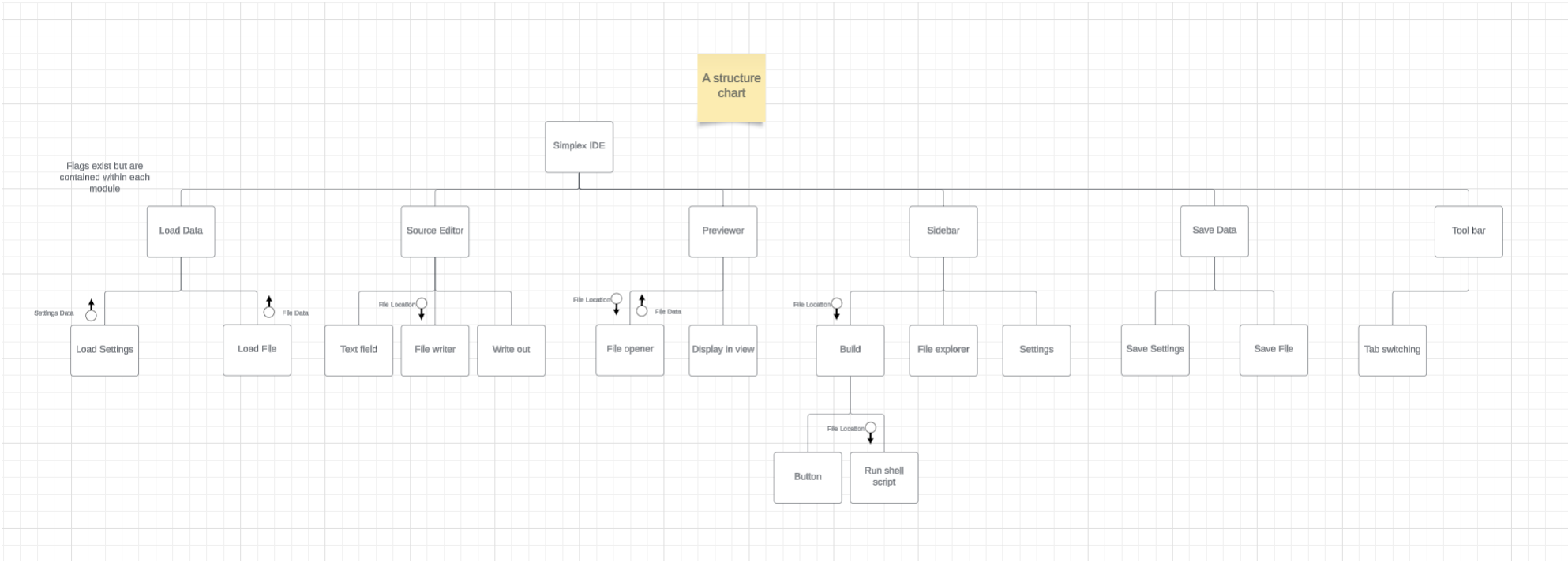



Figure 1.6: A structure chart of my app

| Data Item | Data Type | Format | Bytes for storage | size for display | Description | Example | Validation |
|---|---|---|---|---|---|---|---|
| GUIVariables | Observable Object | - | - | - | A class that becomes an object which stores global variables related to GUI | - | - |
| settingsFile | String | FILE.txt | 50 | 50 | The name of stored settings for the GUI object | GUI_Settings.txt | Is a readable plaintext file plaintext file |
| iconSize | CGFLoat | - | 8 | 8 | A number for the size of icons | 50 | |
| backgroundHue | Double | - | 8 | 8 | A value for the hue of the background | 50 | $0 < backgroundHue$ $backgroundHue < 1$ $backgroundHue < 1$ |
| backgroundSat | Double | - | 8 | 8 | A value for the saturation of the background | 50 | $0 < backgroundSat$ $backgroundSat < 1$ |
| backgroundBright | Double | - | 8 | 8 | A value for the brightness of the background | 50 | $0 < backgroundBright$ $backgroundBright < 1$ |

Table 1.1: A data dictionary for some GUI variables

#### 1.3.3.1 Data Strucutres

The main data that will be used are classes, variables, arrays, dictionaries and files. Classes are required as SwiftUI is object-oriented and requires other objects to interact with. By using classes with the attribute `ObservableObject`, global variables, that can interact with views are possible. The use of variables allows for the application to adapt to user needs in multiple places. For example, a variable for background colour can be changed by users through some GUI elements, which will then affect the background and text editor background as their colours are linked to the variable. Arrays are used for the backend of Simplex to store multiple related things in a central place. An example of this is storing the names of all the sidebar icons in the application. Lists allow key-value relationships which is helpful when loading in setting data. In the intialisation of the application, all setting data can be loaded into a dictionary and then separated by applying the data to the variable that the key represents. Finally, files are used to store data outside of the application's runtime. This is mainly used in storing the settings of the user so that the variable data can be recorded when the application is closed and loaded when it is opened.

### 1.3.4 Algorithm Design

#### 1.3.4.1 Text Cleaning function

The Text Cleaning function will remove any non-ASCII characters, as many programming languages can only handle certain parts of Unicode.

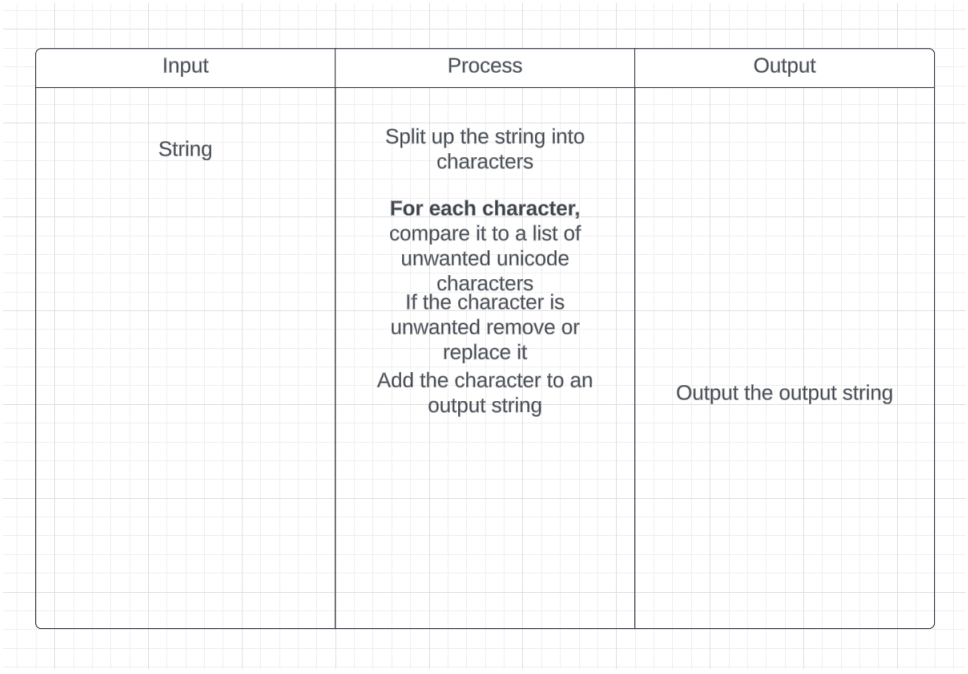



Figure 1.7: An IPO chart of the text cleaning function

#### 1.3.4.2 Settings Interpreter

As the plan is for user settings to be stored in a plaintext file, the contents of the settings file must be read and converted into structures that the application can use. I will call

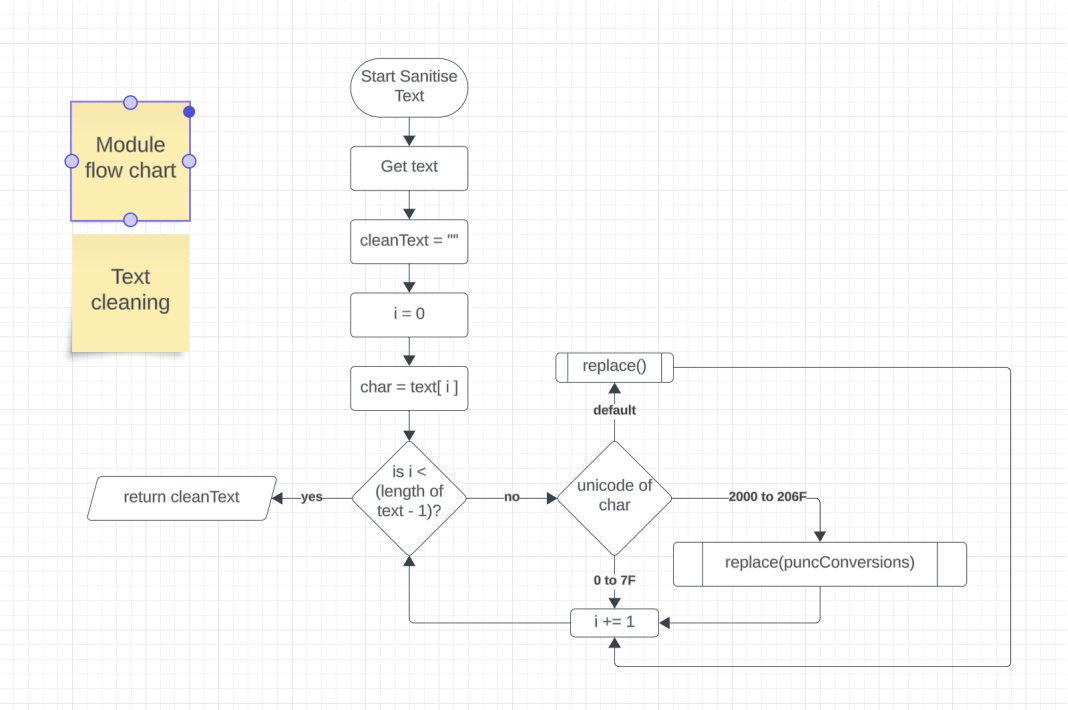



Figure 1.8: A flow chart of the text cleaning module

this function the "Settings Interpreter". The following is the pseudocode for this function.

```
1  BEGIN interpretSettings
2      LET settingsDict = [:[]]
3      GET fileContents
4      LET lines = split fileContents based on "\n" outputting as an array
5      FOR i = 0 TO length of lines STEP 1
6          LET str = lines[i] without whitespace
7          LET sep1 = split str based on ":", outputting as an array
8          LET sep2 = split sep1[1] based on ",", outputting as an array
9          settingsDict[sep1] = sep2
10     NEXT i
11     OUTPUT settingsDict
12 END interpretSettings
```

Note: in line 2, the `[:[]]` where the value in the key-value pair is an array.

# Chapter 2

# Implementation Logbook

**Note:** this is an organised version of my logbook, if the future me has handed in a physical version as well, use this to understand what my scrawl means.

## 2.1 08/12/2023

Did some research for my app as I don't know Swift or SwiftUI right now. I discovered that I can use objects `TextEditor()` and `TextField()`.

## 2.2 09/12/23

Did some more research, this time it was focused on using shell scripts for my app. Here are my notes:

- In-built Process class exists in Swift to run terminal commands.
- e.g. `Process.run(URL, arguements: )`
- connect it to a button
- for preinstalled languages in my app, the URL will start with /packages or something similar.

## 2.3 16/01/24

Discovered the SF symbols which are inbuilt icons for the MacOS. They can be called with `systemImage: "icon"` An example usage:

```
    Image(systemName: "globe")
```

The available icons can be browsed using the "SF Symbols" app.

## 2.4 08/01/24

Coded a prototype GUI that just helps me see where I want things to go, with no real functionality and being very static/hardcoded.

## 2.5 09/12/23

Did some planning of the operation of the sidebar. First, I decided to use the SF icons as the icons for the buttons. I also found the `Spacer()` view which can separate my buttons. I also considered how I could dynamically program the sidebar. What I came

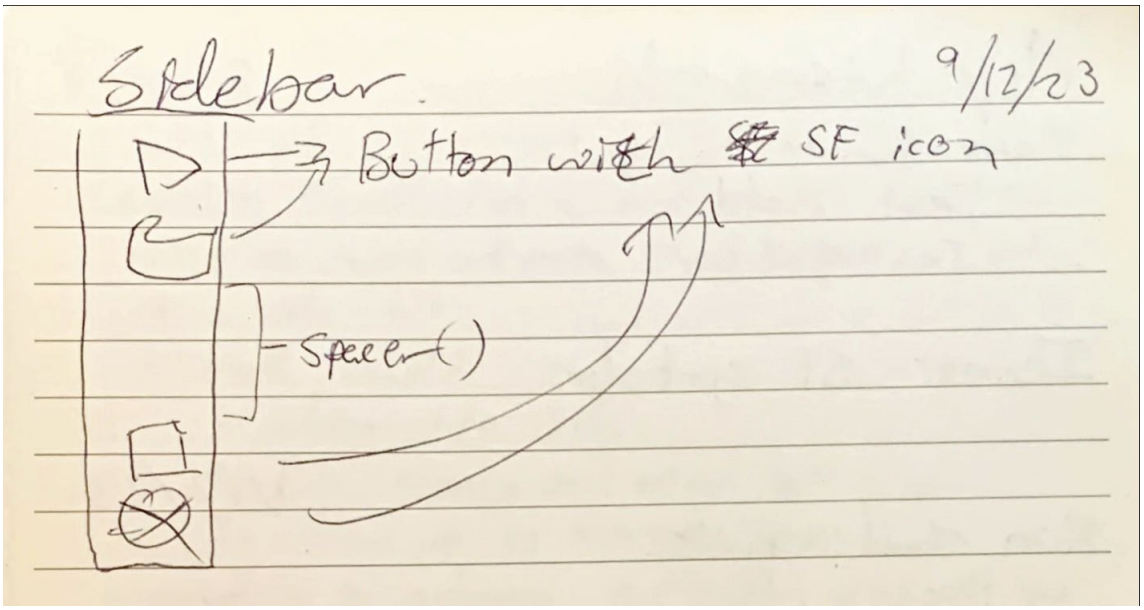



Figure 2.1: A rough sketch of the envisioned sidebar

up with is to make an enumerated array of functions and views which are then used to generate buttons but somehow attach the function to the button views.

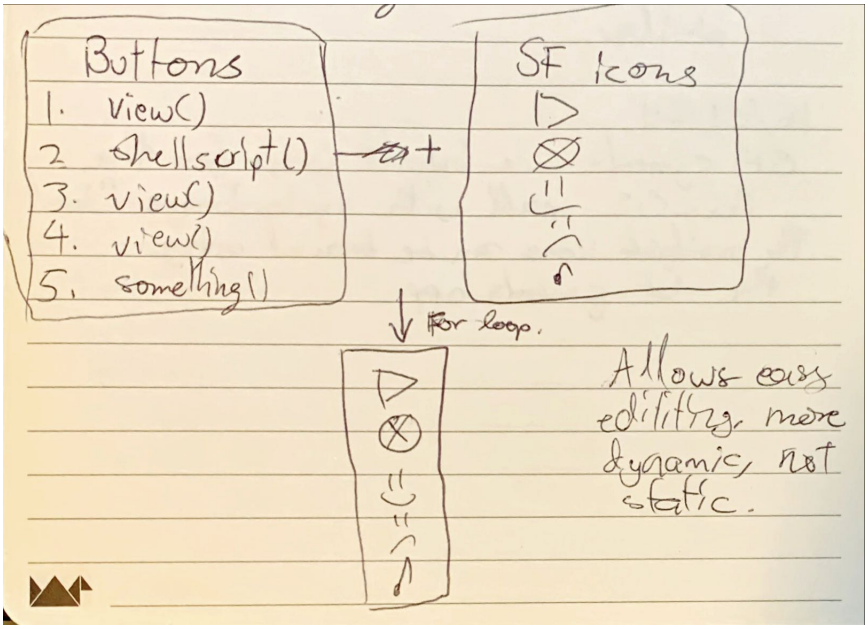



Figure 2.2: A visualisation of how the sidebar should be generated

An advantage of this is that it can support programmatic changes in the future.
Along a similar train of thought, I also planned how I could allow users to customise the GUI. As all top-level variables in Swift are global by default, I could use a single swift file

containing all variables that are used in the appearance of the app.

**Things that should be customised:**

- Background colours
- Icons
  - size
  - shape
- Fonts
  - typeface
  - size
- window size (this is inbuilt) and view sizes

### 2.5.1 Sharing Variables

Something else that I thought about was how to share variables across SwiftUI views. This is something that should be considered as the views have a different "runtime" than normal functional code. So to create global variables that can be accessed by views, I have to:

1. create an object with all variables that I want to share
2. Assign it to an environment by attaching `.environmentObject()` to a view
3. Tell the view to see an object and watch for changes in the variables using the `@EnvironmentObject` wrapper.

**An example of this being implemented in my app:**
In a file for shared variables:

```
import SwiftUI
/*A class is defined that contains all shared variables
with the ObservableObject attribute
that allows views to access an object made with the class. */
class GUIVariables: ObservableObject{
//The @Published wrapper allows views to access the wrapped variable.
    @Published var fullText: String = "This is some editable text..."
}
```

In a file for the preview view:

```
import SwiftUI

struct Previewer: View {
    //This tells the view to look for the object and gives it a name.
    @EnvironmentObject var guiVars: GUIVariables
```

```
    var body: some View {
        Text(guiVars.fullText)
    }
}
```

Some other things that I did were add a background to the app, prepare some prerequisites for window management and test a code snippet for running shell scripts.

## 2.6 05/01/24

Worked on the sidebar. Experimented with using a dictionary to store views and their associated functions but dictionaries in Swift are not ordered, so the icons will not display in order, and a dictionary is probably the wrong data type for this anyway.
I decided that it is better to not use a dictionary and instead, have two separate arrays and look through them together to generate the sidebar. This makes it easier to iteratively add views.
As a stub-like measure for now, I statically added the buttons instead of dynamically generating the sidebar.

## 2.7 08/01/24

Worked on making the dynamically generated sidebar.
This is the plan:

- Array with all icon names
- Array with all actions
- A function with a multiway selection to interpret which actions should be used
- `ForEach()` used to create icons based on the icon array then attaches the function to them on click and pass the corresponding number for the function.

**Something I learnt:** Swift and SwiftUI do not mix. This is because Swift is procedural while SwiftUI is more declarative and reactive, meaning that it only processes if something is changed.

## 2.8 09/01/24

Made a write function for writing text from the TextEditor view to a specified file.

## 2.9 10/01/24

Discovered that functions can't access `ObservableObject` so I coded the function to have parameters for the view to pass in the required variables. Also found out that by default apps have an app sandbox which they access files from.

## 2.10 16/01/24

To get around the functions, which are outside the view environment, not being able to access environment objects, added parameters for the function which will take in the required variable values which the view will pass in. Then, the function will output processed values back into the environment, where the values will be written into the right variables.

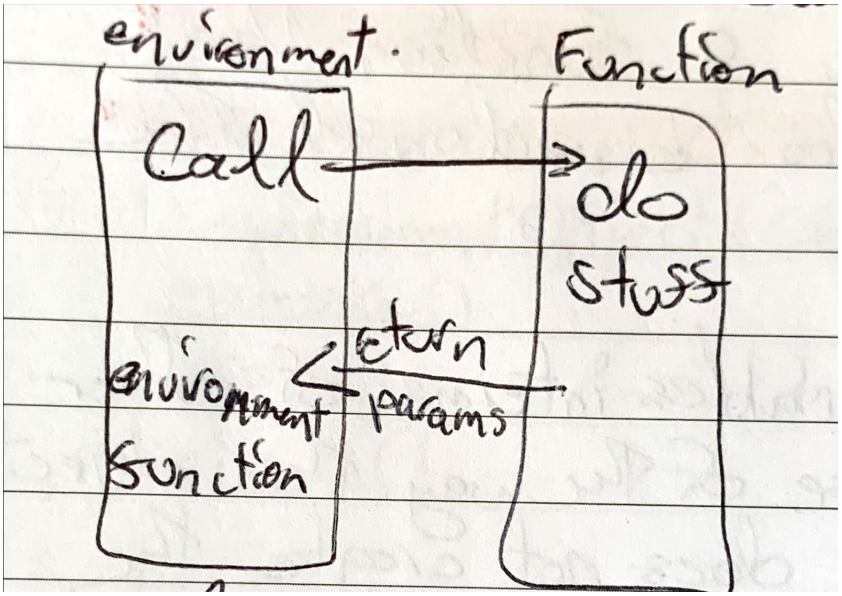



Figure 2.3: A quick sketch of how I want to pass variables

An example use case would be the dynamically generated buttons for the sidebar. By creating an external function, I can easily reuse code and modularise but functions can't perform some GUI actions, so the required parameters for the SwiftUI command can be returned from the function and the command performed in the View environment.

I also discovered that within environment objects, the `@Published` variables cannot be defined by each other. e.g.

```
class MyClass: ObservableObject{
    @Published var variable1: Int = 1
    @Published var variable2: Int = variable2
}
```

This results in an error about the variables not actually being available at the part of the runtime.
The main problem this causes is that I wanted to do something like this:

```
class GUIVariables: ObservableObject{
    @Published var iconSize: CGFloat = 50



    @Published var backgroundHue: Double = 0
    @Published var backgroundSaturation: Double = 0.2
    @Published var backgroundBrightness: Double = 0.9
```

```
    @Publisehd var backgroundColour: Color = Color(hue: backgroundHue,
                                            saturation: backgroundSaturation,
                                            brightness: backgroundBrightness)
}
```

As this causes an error, I have to find a different way to return a color value based on the various components of colour.

To get around this I created a function that will output a variable of type

```
func getBackgroundColour(brightnessMultiplier: Double) -> Color{
    //make temporary variables to do extra manipulation
    var usedBackgroundHue = backgroundHue
    var usedBackgroundSat = backgroundSaturation
    var usedBackgroundBright = backgroundBrightness*brightnessMultiplier

    return Color(hue: usedBackgroundHue, saturation: backgroundSaturation,
            brightness: backgroundBrightness*brightnessMultiplier)
}
```

I also found that the objects attached to views of the same layer are not actually the same object.
i.e.

```
View1()
    .object1() //an object of class object1()
View2()
    .object1() //a different object of the same class
```

## 2.11 17/01/24

Worked on fixing the problem of different objects from the previous day. The solution was to create the objects before attaching them to views. An implementation of this in my code is in my main file:

```
1  @main
2  struct SimplexApp: App {
3      @Environment(\.openWindow) var openWindow
4      @StateObject var guiObject = GUIVariables()
5      @StateObject var sidebarObject = SidebarVariables()
6      @StateObject var editorObject = EditorVariables()
7      //initialisation code
8      init(){
9          print("Initialising")
10     }
11     var body: some Scene {
12         WindowGroup(id: "MainWindow"){
13             MainView()
14                 .environmentObject(guiObject)
```

```
                .environmentObject(sidebarObject)
                .environmentObject(editorObject)
        }
        Window("Settings", id: "settings"){
            SettingsView()
            ZStack{
                Background()
                SettingsView()
            }
            .environmentObject(guiObject)
            .environmentObject(sidebarObject)
            .environmentObject(editorObject)
        }
    }
}
```

Note: lines 4-6 create the objects. Lines 14-16 and 24-26 attach the already created objects to the hierachy.

## 2.12 18/01/24

Added a function to read text from a file. Added some keyboard shortcuts that open different windows.

## 2.13 23/01/24

Realised that the output of the text editor can contain Unicode characters that are not supported by compilers. So I wrote a function that removes or replaces characters that are not in ASCII. To test the function I will use the following data:

- 
- 1
- Hello World
- \u{201D} Note: this represents a Unicode character.
- \u{3032} Note: this represents a Unicode character.

These test empty strings, numbers, ASCII strings, replaceable Unicode Characters, unreplaceable Unicode characters.
I tested the function that I wrote within a playground:

```
    import Cocoa

/*
A dictionary that stores the necessary converions to the ASCII unicode set.
Could be made acessible to the user later.
```

```
Form [Unicode character, ASCII character] but in the form of strings
 */
let randomConversions: [String: String] = ["\u{24}": "dog"]

let puncConversions: [String: String] = [
    "\u{2018}": "'",
    "\u{2019}": "'",
    "\u{201A}": ",",
    "\u{201B}": "'",
    "\u{201C}": #"""#,
    "\u{201D}": #"""#,
    //201E to 2023...
    "\u{2024}": ".",
    "\u{2025}": "..",
    "\u{2026}": "...",
    "\u{2027}": "."
]

func sanitiseText(text: String) -> String{
    var newText: String = ""
    for char in text.unicodeScalars{

        //Define areas of unicode to check
        //Ascii bounds
        let zero: UnicodeScalar = "0"
        let asciiBound: Unicode.Scalar = "\u{7F}"
        //punctuation bounds
        let puncLower: Unicode.Scalar = "\u{2000}"
        let puncUpper: Unicode.Scalar = "\u{206F}"

        switch char.value{
        case zero.value...asciiBound.value:
            //do nothing
            newText += "\(char)"

        case puncLower.value...puncUpper.value:
            //replace the offending character
            let newChar = puncConversions["\(char)", default: "\0"]
            newText += newChar

        default:
            //replace the offending character
            let newChar = randomConversions["\(char)", default: String(char)]
            newText += newChar
        }
    }
    return newText
}
```

```
//Some tests
var testText = " "
print("Original: \(testText)")
print("New: \(sanitiseText(text: testText))")

var testText = "1"
print("Original: \(testText)")
print("New: \(sanitiseText(text: testText))")

var testText = "Hello World"
print("Original: \(testText)")
print("New: \(sanitiseText(text: testText))")

var testText = "\u{201D}"
print("Original: \(testText)")
print("New: \(sanitiseText(text: testText))")

var testText = "\u{3032}"
print("Original: \(testText)")
print("New: \(sanitiseText(text: testText))")
```

## 2.14 26/01/24

Create some functions to save and load settings in between running the app. Did not fully finish functionality for saving settings.

## 2.15 28/01/24

Completed the functionality of saving settings through the GUI.

## 2.16 16/02/24

After taking a break from development, I have decided to work on developing the way Simplex executes code. To do this, I made a few shell scripts which I tested separately from the application, to learn how to write them and make sure they work properly.

## 2.17 01/03/24

Tried to run the shell scripts but discovered that my application could not see them in the file system. This was because my application was sandboxed, meaning that it had very limited access to the local file system. To fix this I will have to unsandbox my application and change the way I reference file paths.

## 2.18 02/03/24

Finished changing variables that affect file paths in my code. Also made a new system of file management so now application files are stored in sensible places.

## 2.19 11/03/24

Worked on preparing the previewer feature of my app by creating a previewer class that contains the variables requried. This not only involved creating the class but also declaring an object of that class, attaching the object to the required windows and executing functions to enable the user to change the previewer settings.

### 2.19.1 An Error

Whilst doing this, I ran into the following error:

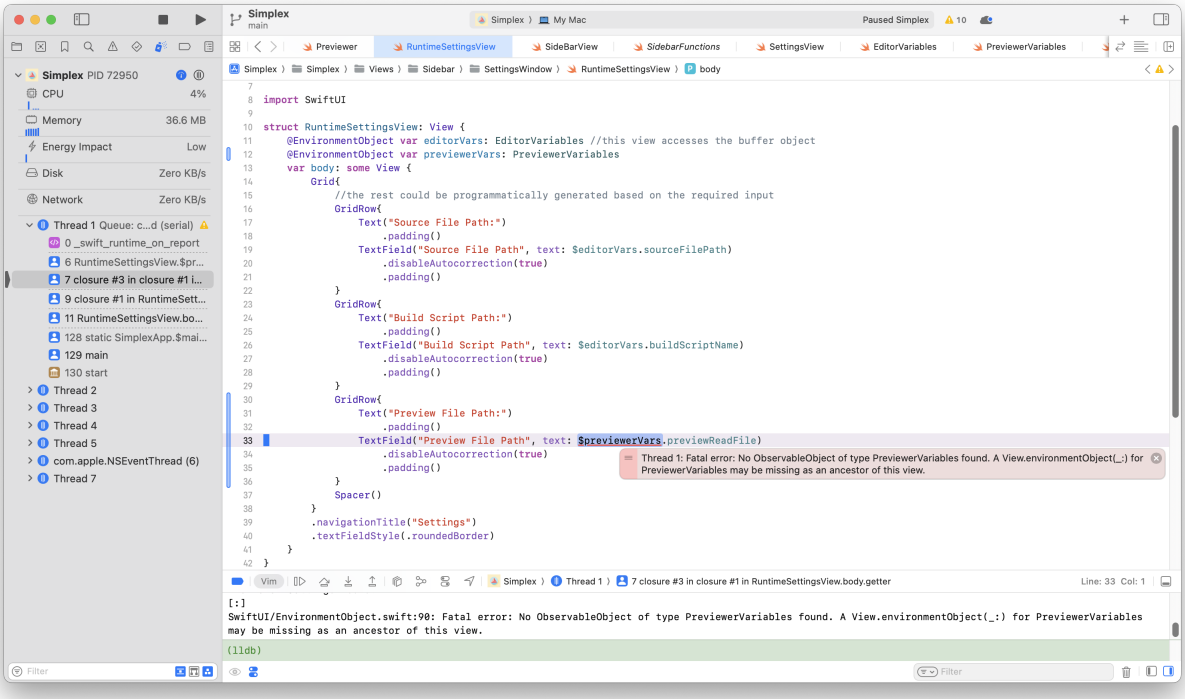



Figure 2.4: The error I encountered whilst implementing a previewer object

The error message reads

```
Thread 1: Fatal error: No ObservableObject of type PreviewerVariables
found. A View.environmentObject(_:) for
Previewer Variables may be missing as an ancestor of this view.
```

Although I thought I had attached the previewer object correctly to create the proper hierachy, I forgot to attach the global object to the settings window as seen below.

```
//the main scene
var body: some Scene {

//This is the main window where the user will spend most of their time
    WindowGroup(id: "MainWindow"){
        MainView()
            .environmentObject(guiObject)
            .environmentObject(sidebarObject)
            .environmentObject(editorObject)
            .environmentObject(previewerObject)
    //the above commands attach the objects to allow views lower
    //in the hierarchy to "see" them
    }

    //this window contains all the GUI for editing the app settings
    Window("Settings", id: "settings"){
        ZStack{
            Background()
//                SettingsView()
            NoDeprecateSettingsView()
        }
        .environmentObject(guiObject)
        .environmentObject(sidebarObject)
        .environmentObject(editorObject)
    }
    //keyboard shortcut of "cmd"+"," opens the window
    .keyboardShortcut(",")
```

Notice that `.environmentObject(previewerObject)` is not added to the end of `Window("Settings", id: "settings")`.

## 2.20 10/04/24

Worked on implementing some of the feedback I got from Israel. I worked on adding the frontend color picker view and some extra backend to synchronize it with my existing methods. See Section 3.3.2 for more detail.

## 2.21 03/05/24

Did some work on adding shortcuts for the sidebar. An interesting issue that occured is that the function `.keyboardShortcut()` required an input type of `KeyEquivalent` which only characters can be type casted into. Then, characters of strings in Swift cannot be easily extracted, requirirng the following solution to get variable keyboard shortcuts. `.keyboardShortcut(KeyEquivalent(Character(UnicodeScalar(string)!)))`

# Chapter 3

# Evaluations

## 3.1 Test Data

First, each module of the application will be separately tested with their own set of test data. For an example of testing the modules, see Section 2.13.
To test the whole application, several bits of data will have to be input into the various fields available. Currently, there are 4 unique text fields in my application. For each of these, a null string, ASCII string, numbers and Unicode string with characters outside ASCII ranges, will be used. Additionally, 3 of these text fields are inputs for file paths, so a file path that exists and one that doesn't will also be used for each of them. The remaining text field will be the code editor, so there are no additional tests. For each of these fields, I will use the following test data:

- "" Note: this is an empty string
- Hello World
- 1
- U+3032 Note: U+3032 refers to the Unicode character.

The file path test data is quite variable so it is quite hard to consistently use the same one.

Other interface elements like sliders and buttons, output a set range of data. For sliders, a range between 0 and 1, and for buttons a boolean value of true or false. To test these, the modules that interact with them should be tested within the range of data that they output, for example, the slider will be tested with 0, a number between 0 and 1, and a number outside of that range.

Successes and failures using this test data will provide a quantitative evaluation of the Simplex application.

## 3.2 Self-Evaluations

So far, the Simplex application performs the basic operations that a required. It is able to read, write and execute code, and provides a simple GUI for the user to interact with. However, it suffers from serious performance issues, has some unintuitive GUI elements and does not have the full features. I think I have used my time effectively in developing

this application as it has some functionality. The things I will be focusing on next are improving the GUI and improving performance. If there is still time left, I will add more features including a documentation interface, music player and custom file explorer. However, these will only be implemented if there is enough time, to prevent scope creep.

## 3.3 Israel's Evaluations

- Colour picker for background colour instead of just a slider.
- Clearly show what kind of file is open.

e.g. show a python symbol when a python file is open.

### 3.3.1 Issues to consider if there is more time

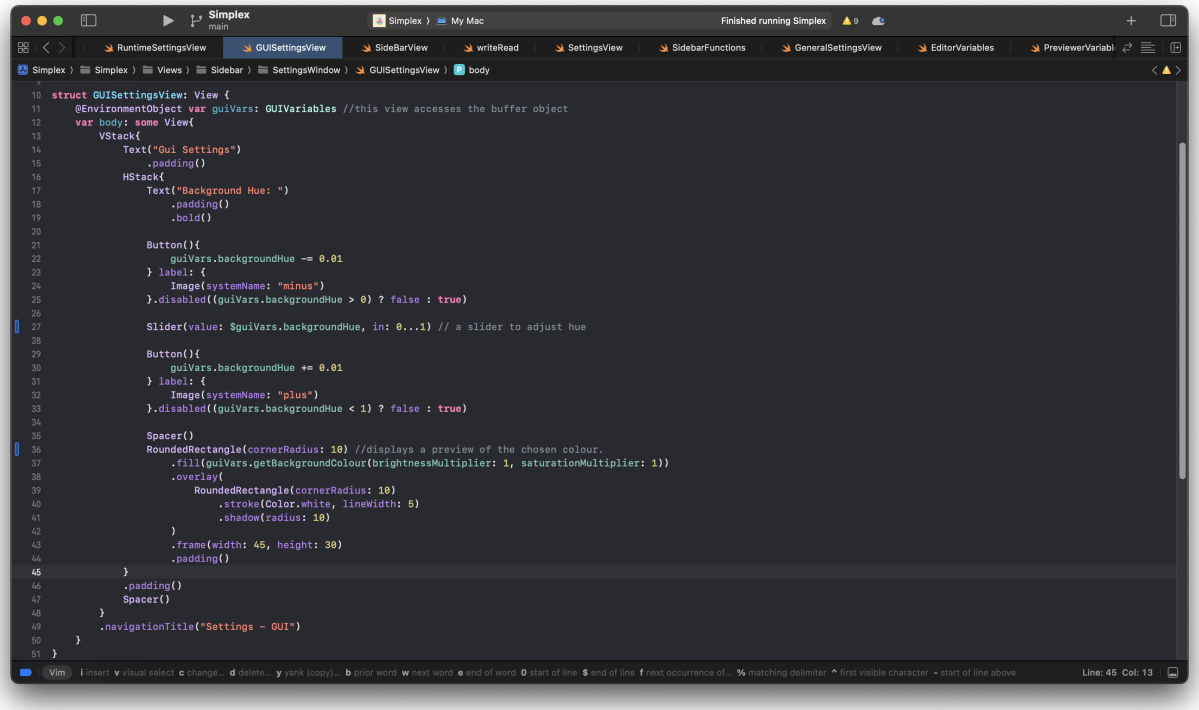
- automatic indenting
- syntax highlighting
- customisation for these features

### 3.3.2 Evidence of Implementation

The next page shows my implementation of a color picker feature instead of just the slider.

### 3.3.2.1 Before:

Before the implementation of the color picker...



```
struct GUISettingsView: View {
    @EnvironmentObject var guiVars: GUIVariables //this view accesses the buffer object
    var body: some View{
        VStack{
            Text("Gui Settings")
                .padding()
            HStack{
                Text("Background Hue: ")
                    .padding()
                    .bold()

                Button(){
                    guiVars.backgroundHue -= 0.01
                } label: {
                    Image(systemName: "minus")
                }.disabled((guiVars.backgroundHue > 0) ? false : true)

                Slider(value: $guiVars.backgroundHue, in: 0...1) // a slider to adjust hue

                Button(){
                    guiVars.backgroundHue += 0.01
                } label: {
                    Image(systemName: "plus")
                }.disabled((guiVars.backgroundHue < 1) ? false : true)

                Spacer()
                RoundedRectangle(cornerRadius: 10) //displays a preview of the chosen colour.
                    .fill(guiVars.getBackgroundColour(brightnessMultiplier: 1, saturationMultiplier: 1))
                    .overlay(
                        RoundedRectangle(cornerRadius: 10)
                            .stroke(Color.white, lineWidth: 5)
                            .shadow(radius: 10)
                    )
                    .frame(width: 45, height: 30)
                    .padding()
            }
            .padding()
            Spacer()
        }
        .navigationTitle("Settings - GUI")
    }
}
```

Figure 3.1: The code before implementation

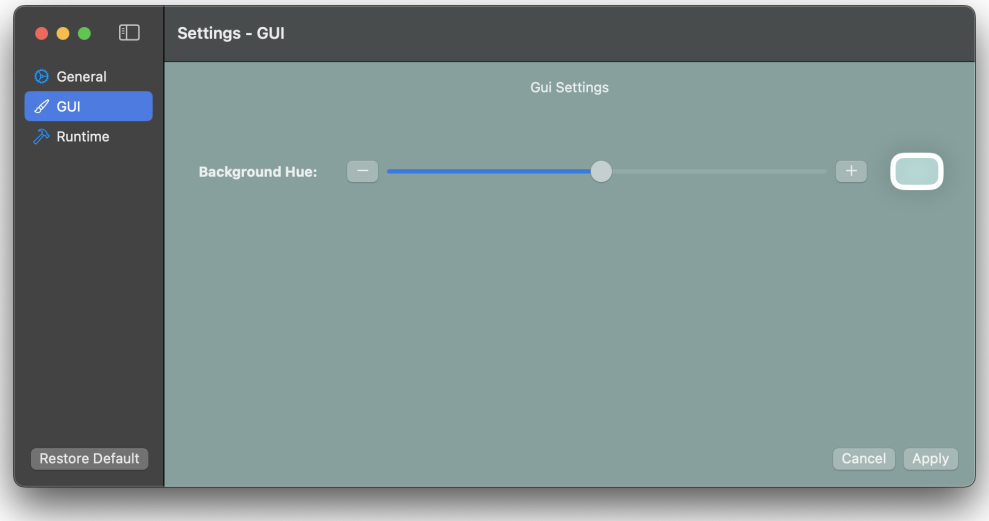



Figure 3.2: The GUI before implementation

### 3.3.2.2 After:

After the implementation of the color picker...

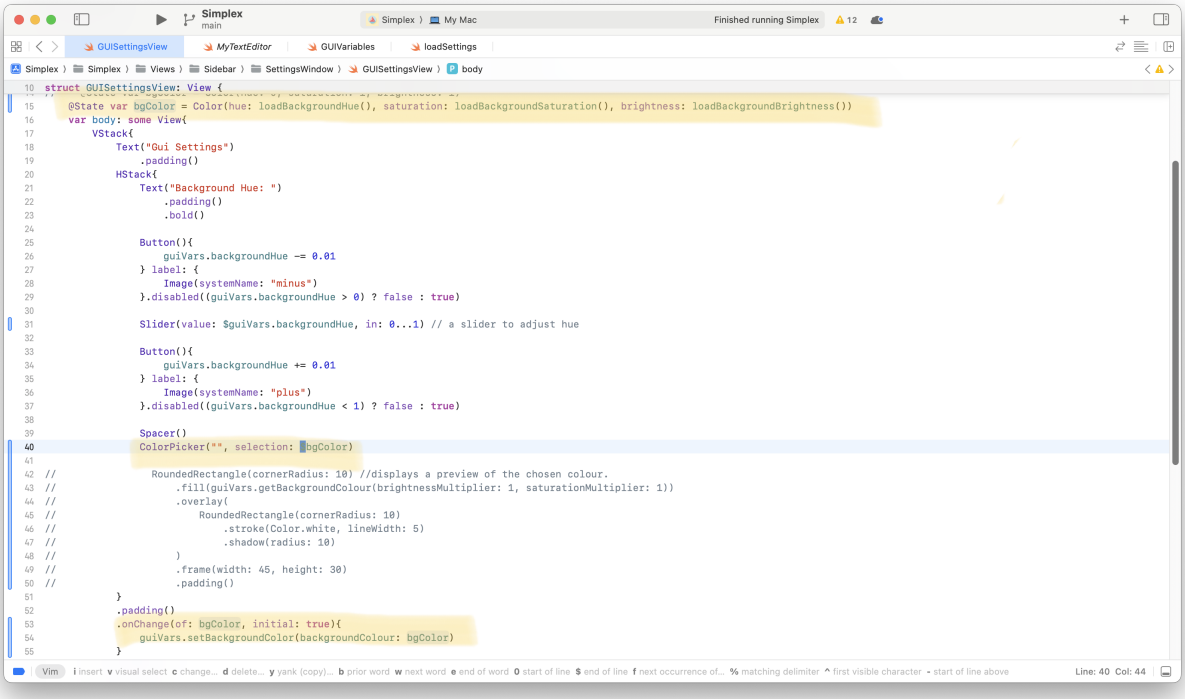



Figure 3.3: The frontend code after implementation

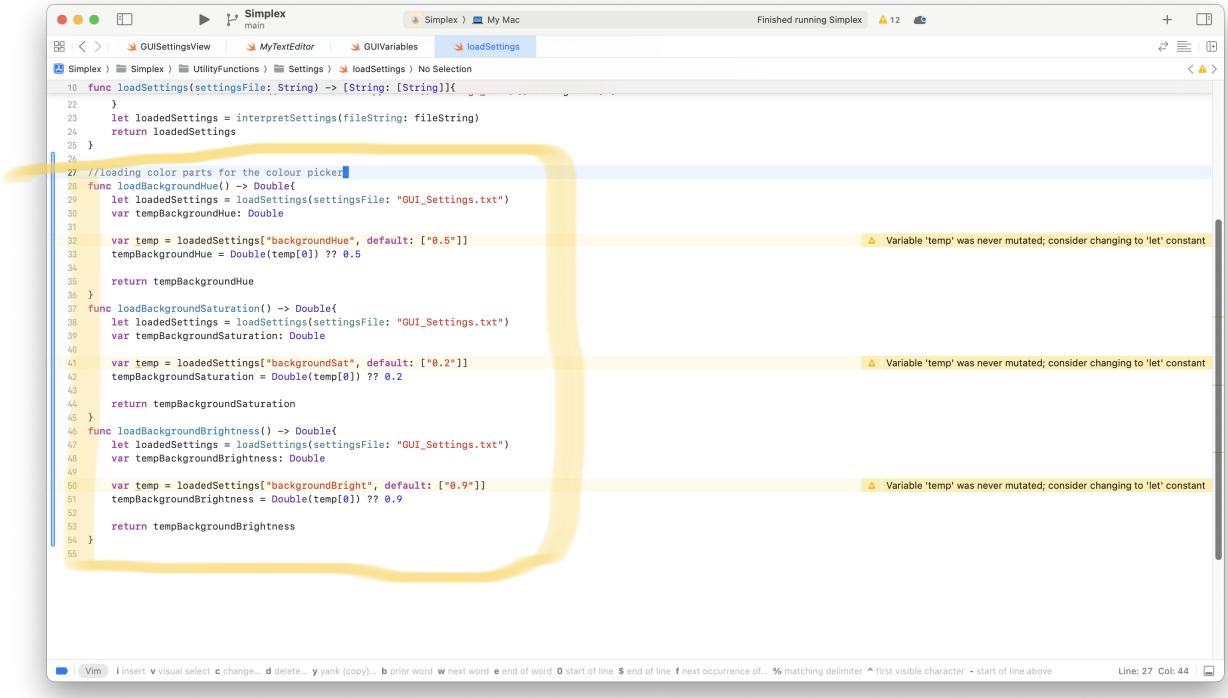



Figure 3.4: The backend code before implementation

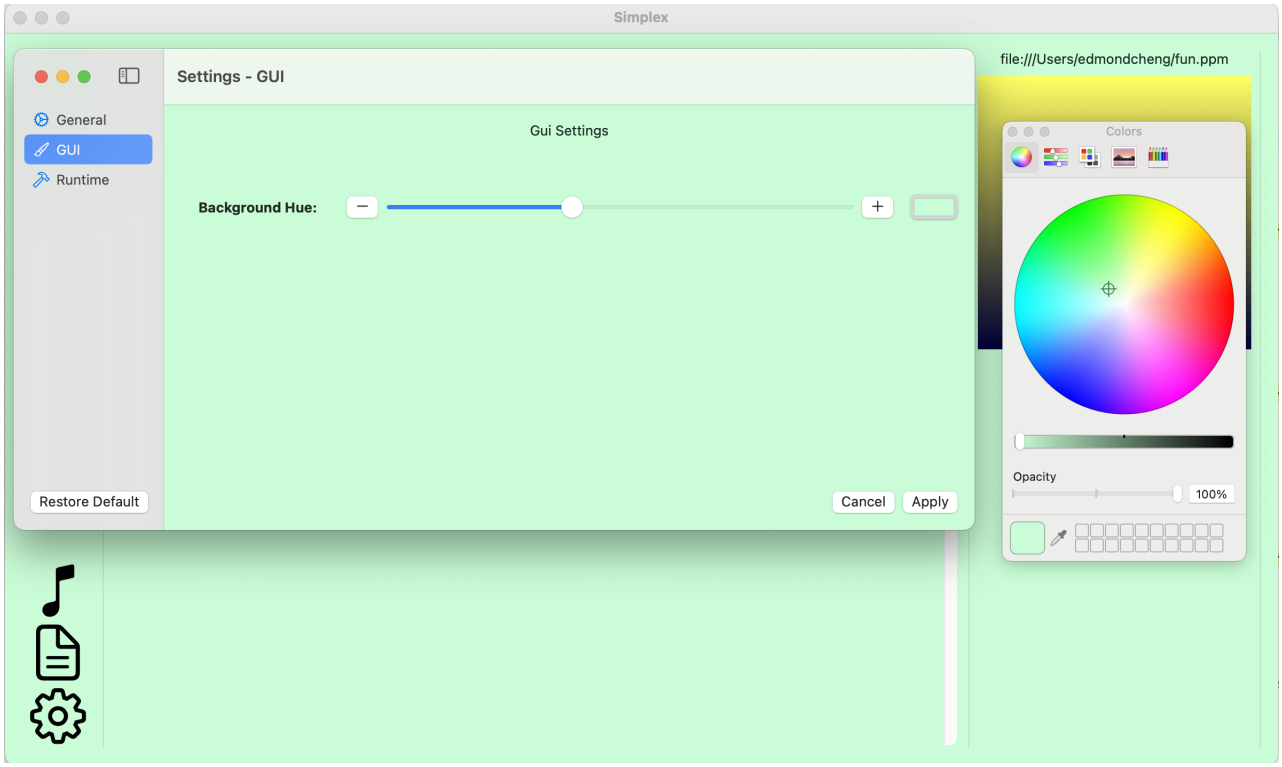



Figure 3.5: The GUI after implementation

# Bibliography


Apple Inc. (2024). *Swiftui — apple developer documentation*. Retrieved December 8, 2023, from https://developer.apple.com/documentation/swiftui

Couriol, B. (2024, February 25). *The creators of the atom code editor open-source zed, their new rust-based high-performance editor*. Retrieved April 7, 2024, from https://www.infoq.com/news/2024/02/zed-code-editor-open-sourced/

Creative Commons. (2024). *Cc by-sa 4.0 deed*. Retrieved March 20, 2024, from https://creativecommons.org/licenses/by-sa/4.0/deed.en

Davis, S. (2012). *Software design and development: The hsc course (second edition)* (J. Fendall, Ed.). Parramatta Education Centre.

Hosseini, M. (2020, July 20). *How to get rgb components from color in swiftui*. Retrieved April 9, 2024, from https://stackoverflow.com/questions/56586055/how-to-get-rgb-components-from-color-in-swiftui

Hudson, P. (n.d.). *How to use @environmentobject to share data between views*. Retrieved December 9, 2023, from https://www.hackingwithswift.com/quick-start/swiftui/how-to-use-environmentobject-to-share-data-between-views

Laku, Z. I. (2024). *How to add options in a bash script [2 methods]*. Retrieved March 1, 2024, from https://linuxsimply.com/bash-scripting-tutorial/functions/bash-script-options

Moeykens, M. (2023, March 30). *Swiftui environmentobject: How to share data between views*. Retrieved December 9, 2023, from https://medium.com/@mark.moeykens/swiftui-environmentobject-how-to-share-data-between-views-13f354011081

Sarun. (2021, April 19). *How to set a screen's background color in swiftui*. Retrieved December 9, 2023, from https://sarunw.com/posts/how-to-set-screen-background-color-in-swiftui/

Siegel, D. (2018). *When is copyright infringement committed?* Retrieved March 20, 2024, from https://law.stackexchange.com/questions/24517/when-is-copyright-infringement-committed

Stack Overflow. (2024). *Public network terms of service*. Retrieved March 20, 2024, from https://stackoverflow.com/legal/terms-of-service/public#licensing

user3064009. (2018, April 26). *How do i run a terminal command in a swift script? (e.g. xcodebuild)*. Retrieved December 9, 2023, from https://stackoverflow.com/questions/26971240/how-do-i-run-a-terminal-command-in-a-swift-script-e-g-xcodebuild

Wikipedia. (2023). *Idea–expression distinction*. Retrieved March 20, 2024, from https://en.wikipedia.org/wiki/Idea%E2%80%93expression_distinction#Merger