2022

UNIT 4-7 ASSIGNMENT 2

PROGRAMMING AND MOBILE APPS

OLIVER COLLINS-COPE

Contents

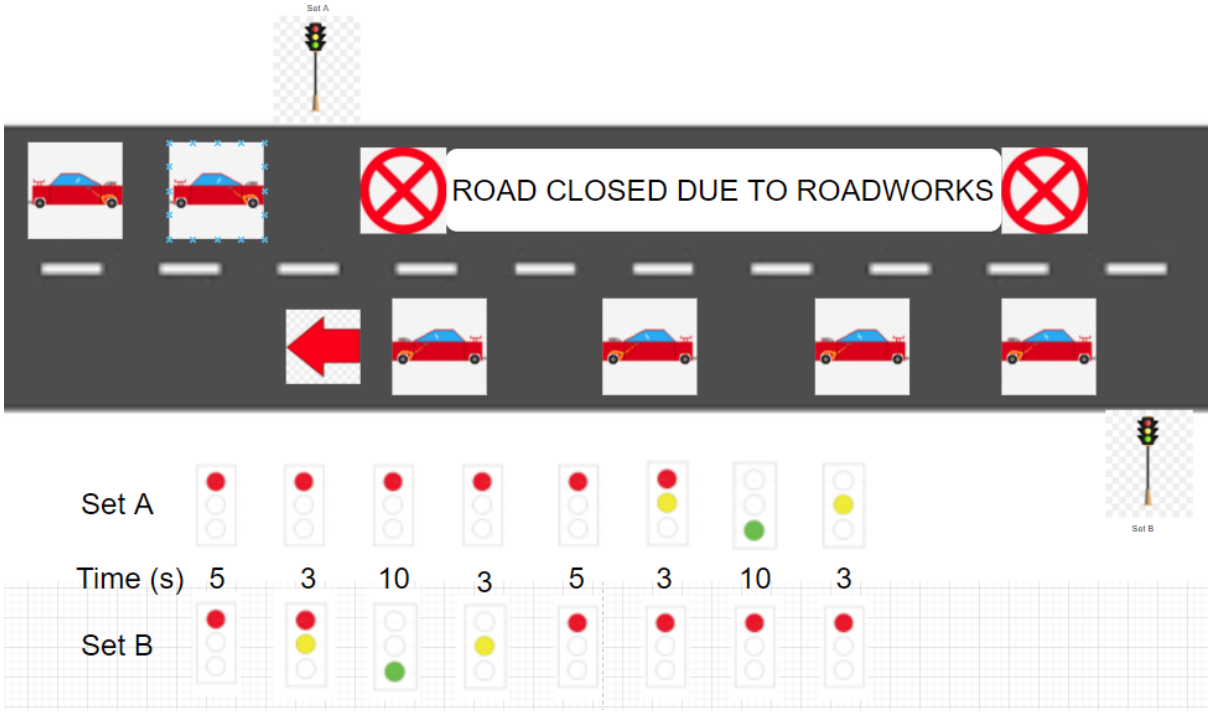
Introduction

Inside of this project, I will be developing a mobile traffic light application. I will make this after first developing my plan for the application, going into depth with the scope and designing it in clear detail. Following this, I will be testing my application and receiving feedback. It is crucial to consider all features when developing my application and all things that might need to be included.

Scope of Project – Design

The project is to create a program for temporary traffic lights that can be operated using a mobile device. It is being created due to the roadwork occurring and it will help to allow traffic to flow functionally without halting day to day life. It has been approved by the relevant organisation "Highway-RUTC Road Services" and the project will be completed through numerous different phases which will be elaborated on further, such as planning and implementation phases. The product produced will be a functioning program which will be able to change traffic lights using a mobile and will follow the predetermined eight set instruction sequence that lasts forty-four seconds. It will be delivered by the 20th of May and is estimated to cost a grand total of £0 through the use of student labour.

To be included in the program includes the correct operational sequences for both Set A and Set B of the traffic lights. Furthermore, there will be an option to override the operational sequence in order to set both traffic lights to red in case of an incident/accident. Following this, the user will be able to safely restart the traffic light operational sequence after overriding the program and it will continue to function smoothly. All of this will be achieved through a mobile app that will be developed around the program I will create.



Mobile requirements

There are four components to consider for the mobile application to consider when designing the mobile application. These are:

- Device capabilities
- Input required
- Output required
- User needs

Device capabilities

The device capabilities refer to things such as GPS, or an accelerometer. For this situation/program, it is important that the devices that use the app we create have touch screen capabilities in order for the user to interact with the buttons that will be present in the application, whether it is to activate the program or to activate the emergency stop.

Input required

Similar to the device capabilities that we require for this application and program, “Input required” refers to what input the program will seek from the user, such as voice input, or a timed event. For our program, we will be using a combination of touch screen and timed events.

We will be using touch screen in order to begin the program and activate the emergency stop using the mobile device on the application, and we will be using timed events to ensure that the operational sequences on the traffic lights eight different options runs smoothly and in time to prevent issues arriving from cars colliding.

Output required

The output required asks the program what the output of the input will be. As mentioned above, this will be the traffic lights working in the order they are established to be working in, going through the eight different sequences, along with the emergency stop option.

This is important as this output component will be what allows the program to operate functionally and dictate the response from what the user inputs.

User needs

Finally, there are also the user needs to consider, such as whether the user needs accessibility options, or any kind of location services. In this situation, some examples of possible user needs may be a password protected system, so only authorised users are able to affect the traffic lights, and accessibility options in-case the user operating the application might have different disabilities, such as blindness or epilepsy. We can achieve these different user needs by ensuring the program is able to be accessed by third party voice screen readers and applications such as “Voiceover” by Apple. We can cater to epilepsy users’ needs by ensuring that the application/program does not contain any potentially triggering content, such as flashing lights.

These are important to consider as making an application as accessible as possible to different users is crucial as a developer in order to allow for as many users as possible to be able to use the application and not be restricted by their disabilities. Furthermore, it is also good practice to ensure that disabled users are able to access and fully use the produced application just the same as abled users.

Pseudocode for Program

```
Input Driver Age
If Driver Age is greater than sixteen
        Then they are allowed to drive
Else
        They are not allowed to drive

Is there an accident?
Input Yes/No
If Yes
        Then Emergency Stop
Else
        Continue running traffic lights

Input Touch screen
If button program is pressed
        Traffic lights Set A Off/Red (5)
        Traffic lights Set B Off/Red (5)
        Traffic lights Set A Off/Red (3)
        Traffic lights Set B Half/Red&Yellow (3)
        Traffic lights Set A Off/Red (10)
        Traffic lights Set B On/Green (10)
```

ETC

```
Else
        Program does not run and traffic lights stay off


Input Age
If Age is greater than sixteen
        Return Yes
Else
        Return No

Module Module1
        Sub Main()
                Dim Age as Integer = Nothing
                Console.WriteLine("What is your age?")
                Age = Console.ReadLine()
                If Age >= "16" Then
                        Console.WriteLine("You are old enough to drive")
                Else
                        Console.WriteLine("You are not old enough to drive")
                End If

                Console.ReadLine()
        End Sub
End Module
```

Software solutions design

There are several things to include when discussing the software design and viable solutions. To begin with, I will discuss different problems that must be resolved before planning out the software and programming.

To resolve

These include:

- Intended users
- Summary of the program and solutions
- Constraints
- Benefits
- Interactivity
- Complexity

Intended users

The intended users of this application and software will be members of the company Highway RUTC Road Services, which are likely to be slightly older people who might not be

as technologically literate, and therefore our software should be as simple as possible to minimise chances of confusion.

Summary of the program and solutions

The program will include a diagram of the two sets of traffic lights, and these traffic lights will change colour depending on the timing of the predefined operational sequence. Due to the fact that the brief states that the actual timing of the traffic lights will be different, it will be important to include the option to easily be able to change the timing of each step of the correct operational sequence.

One way I will be able to achieve this is through setting up an internal timer with the code, and when the timer reaches a certain value, it will change the colour of the traffic lights and therefore run through the correct operational sequences.

Constraints

Some constraints that may be encountered along the way with the development of this program could be that setting up the internal timer and making sure it functions the way it is required will be much harder than initially predicted and therefore there may be bugs or I will have to change the way I decide what time to change the code.

Another constraint I may come across would be I am able to turn this into a mobile application and have it run for the user; However, I believe that I have resolved this issue already through deciding how I am going to make my software, which I will discuss in the section below.

Benefits

I will be using the game development software “Unity” in order to make my software application. I believe this is a benefit because it allows for builds in mobile which means I do not have to worry about making my program function on mobile. In addition to this, I will be able to make different menus for the user and present customisable options through the use of “scenes” and “game objects” which interact with each other.

Interactivity

The interactivity of my application will be limited to buttons that the user can press in order to achieve the desired results. I am limiting the interactivity of my application in order to reduce confusion and limit user error. This will result in less errors occurring and less bugs developing and therefore the application will continue to work for longer without the need for updates or hotfixes.

Complexity

I do not believe that my application will be particularly complex, and I will do all I can to ensure that the UI of my application and the navigation of my application will be simple in order to reduce complexity and confusion.

Purpose and requirements

In this section I will clearly outline the purpose of my application and the requirements of the user in further detail, while including how I will meet these requirements and my justification for doing in that way.

Purpose

The purpose of this software and mobile application will be to create a software solution that can operate a traffic light software system using predetermined operational sequences.

Requirements

The requirements of this software solution include following the correct operational sequence, which I will achieve through using a timer, and I will be doing it in this way as it will allow me to keep track of the timing within the code and allow it to be independent. To be able to safely override the operational sequence and restart it in the case of an emergency, which I will achieve by including a button which stops the code from executing and sets the lights to red. Finally, to develop a mobile app for this which can achieve this, which I will do using Unity because it has build options for mobile.

Features of the software design

The features of the software design will cover:

- Main tasks, inputs, and outputs
- Illustrations
- Algorithms and pseudocode
- Data Structures
- Data Storage
- Control structures
- Data validation
- Error handling and reporting

The requirements of this software solution include following the correct operational

Main tasks, inputs, outputs

The main tasks, inputs, and outputs of this program will include:

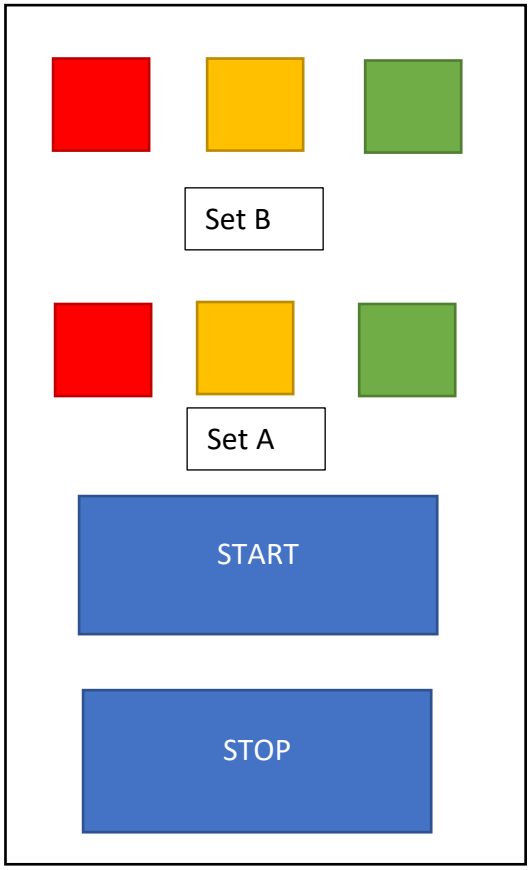
Main tasks – starting the lights and running the correct operational sequence for them.

Inputs – the user pressing the button on the screen in order to start the traffic lights and emergency stop them.

Output – to output the current setting that the traffic lights are on, and a noise when the emergency stop is played.

Illustrations

The layout of my application will look something like this.



Algorithm and pseudocode

The pseudocode for my application will be this, as I believe this is the best way to do this.

```
IF BUTTON START PRESSED Then
        Start coroutine sequence1
```

Or I will establish a timer and when that timer reaches past a certain value, it will run a specific function which sets the light to the right colours.

```
IF BUTTON START PRESSED Then
        timerLights += Time.deltatime
IF timerLights > int step1Time Then
        Sequence1();
```

IF timerLights > int step2Time Then

Sequence2();

Ect…

Data structure

I will be structuring my data to be easily manipulated by the user in Unity, with fields exposed to the inspector so the user can easily configure the sequence time. In addition to this, I will make as much of my data as I can private and not public, in order to minimise the risk of errors occurring when running my program.

Data storage

My data will be stored inside of my program in my scripts written in C#. This will allow my program to run optimally and minimise build size.

Control structures

My program will include many control structures as they will be what allows my program to function properly, and I will make sure to include comments on my code in order for people to understand the code.

Data validation

I will ensure that the data is valid through the data types used declared in my scripts. For example, the timer will be a float as time has a decimal value, however the specific times at which the lights change, the “step1Time” will be integers, as these will be prespecified seconds.

Error handling and reporting

To handle my errors and reporting them, I will include debug.logs in my testing of my program, and if it is necessary, I will throw an error box on the screen when something unexpected happens with my code in order to inform the user that it is not functioning correctly.

Choice of language

I will be using C# in order to make my program as this is the language used by Unity, and therefore this decision was already made when I made the choice to use Unity as my program. It was based off of C, the low-level programming language, and expanded on many of its original and powerful features. Today, C# is a static typing, strong typing, lexically scoped, imperative, declarative, functional, generic, object-oriented (class-based), and component-oriented programming disciplines.

List of predefined programs/code snippets

The only pre-defined program to mention will be Unity, as this is the software I will be using to develop my application, and potentially visual studio as I will be using this as my IDE to program in C# for my applications.

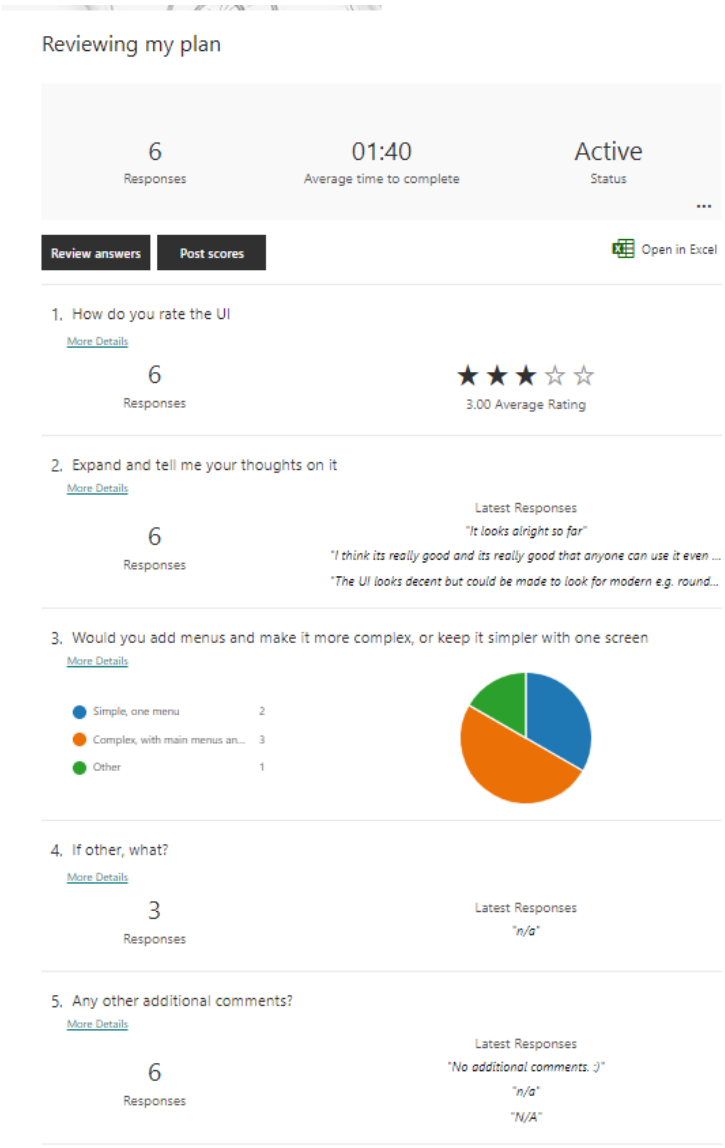
Premade assets

I may use premade traffic light images; however, it is very likely that I will use boxes that I change the colour of as this will save space and loading times for the application.

I will also likely use some sort of alarm noise that I will download from somewhere, provided I have permission to use it in my application.

Feedback from others

Here I will ask for feedback regarding what I have written so far, as well as acting on that and improving my assignment based on what I receive.



Judging from the reaction to my assignment, I will adjust my plan to include another main menu screen before it loads the screen I depicted above. Furthermore, I will add more modern displays in my final design, such as circles over squares in order to make it look more like actual traffic lights.

Test data

I will include test data once I have begun developing my program as I do not want to skip important steps such as deciding on my architecture of my program and things such as requirements for my application which are vital to development.

Technical constraints

Some technical constraints I may encounter could be limitations with getting the application on apple devices.

Another technical constraint I might run into could be limitations with the programming language I am using and the software, however I am not aware of any that I might meet and therefore I believe that I will mostly be free of technical constraints.

Designing a mobile app

I will briefly cover this section of my assignment, as a significant part of it is covered in the programming aspect with “Software solutions design” above.

User requirements

As discussed above, the user requires an application which can change the traffic lights by following a specific sequence and therefore changing when required, which can also stop at the press of a button and restart when another is pressed.

A proposed solution

A majority of this section was covered above; however, I will include it for complete clarity and to ensure everything was covered to the best of my ability.

- Program tasks
- Target platforms
- Screen layouts and navigation
- Algorithm and pseudocode
- Control structures
- Data validation
- Integration of device capabilities

Program tasks

My programs tasks will be running the application and the lights in the correct sequence as I had previously mentioned. Furthermore, stopping when a button is pressed and being able to restart without issues.

Target platforms

The target platforms for my application will be iOS devices as well as Android devices, however I will include build options for windows so users can test out the application as if it was a mobile emulator before they use it on mobile.

Screen layouts and navigation

The screen layout and navigation of my application has been discussed above as well, I will include a main menu which users will load into, and then once they press a button to proceed to the application, I will add the circles for my traffic lights, and my buttons which start and stop the sequence respectively.

Algorithms and pseudocode, control structures, data validation

All of these have been discussed above and I will not discuss them here, please see pages 8 and 9 for this.

Integration of device capabilities

My application will not include and integration of device capabilities as I do not believe that a traffic light system requires this, and if I added device capabilities then my application will be limited to specific devices that I have added these capabilities for, and I want to ensure that my application can be used universally.

Alternative solutions

Some other solutions that could be included might be making the app to be developed landscape rather than portrait, however I do not think this is appropriate for my application. Another example of an alternative solution would be to develop my program in something like windows forms, however I am not certain how I would port it to mobile, so I believe that the way I have chosen is the best for me.

Details of resources used

The only resources I will be using (as discussed above) will be the audio of an alarm noise for my emergency stop button in order to help alert the user.

See above for more detail.

Test and review schedule

As mentioned above, I have no current test and review schedule however once I begin developing my program then I will add my tests as I produce my program.

Mobile constraints

Some mobile constraints that I may run into could be device limitations with battery and such. Another mobile constraint I might run into is if my mobile device is too outdated, I will not be able to download and run the application, and therefore I have to develop for older devices as well.

Furthermore, the phone may not be able to handle the load my application will put on it. However, considering the program and application size, I have concluded that if this is an issue, then it is the fault of the device being outdated and will not be able to run other applications either.

Legal and ethical considerations

The legal and ethical considerations for me to consider would be any data gathering that I would like my application to do, which I will not include, and any copyright issues. In order to avoid these copyright issues, I will make sure I have the rights to whatever sound I use for the alarm so I can proceed with peace of mind.
I may also have to consider any ethical issues like ensuring the application I design is suitable for users with accessibility issues like colour blind issues, and I will resolve this by adding in text for my application that ensures the users understand which lights are turned on and which are off.

Application Development

Content preparation and software solutions development

Software solutions development

This will include:

- The development environment to produce code
- The development and the refinement of my program using C# (the suitable programming language)
- Libraries, standard code, and user generated subroutines for efficacy

Development environment

I will be using Visual Studio 2022 Community as my IDE for developing my program, as it is the IDE I am most familiar with, and is the standard Unity IDE, other than monodevelop, however I have no experience with this, and Visual Studio is a reliable IDE.

Refinement of my program

In order to further enhance and refine program, linked with user generated subroutines, majority of my code will be made within functions that are aptly named so it is easier to read and understand without having to look at all of my code. I have done my programming in C# as that is the programming language for Unity.

Libraries, code, efficiency

As mentioned above I am making my own subroutines, also known as functions, however as they do not return values then some could argue they are not functions, which has all of my code for each of the steps of the sequence, which can each be individually modified without difficulty and without struggling to locate the correct part of the code, as it is named under each function.

I will be using the Unity Libraries that come with Unity in order to do this and the standard code alongside it.

Content preparation

This will cover:

- Selection and application of appropriate processing and editing for the specific devices
- Optimisation
- Formats
- File formats
- Compression

Selection and application of appropriate processing and editing for the specific devices

The goal with my application is to make it run universally across all applications without having the need to modify it for different devices, and therefore I will avoid doing things for specific devices.

Optimisation

The only file that the application has which the user might have to load would be the audio file, which is compressed and set to decompress on load, and therefore file size should not be a problem with my application.

Formats

I will only be developing for portrait and therefore landscape will not be an option for my application, so I will be only allowing portrait.

File formats

As previously mentioned, the only file needed to load will be an audio file, in mp3, and therefore will be usable by all devices.

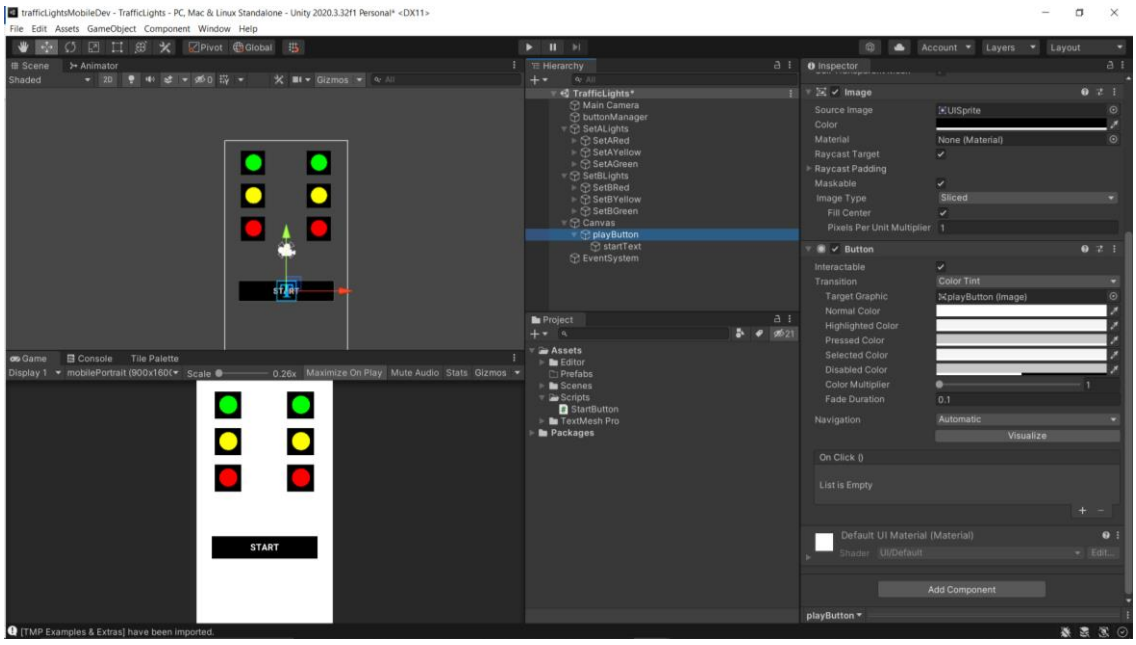
Compression

Also mentioned above, the file for the audio of the alert noise will be compressed in order to reduce the space it takes up and the size of my application.

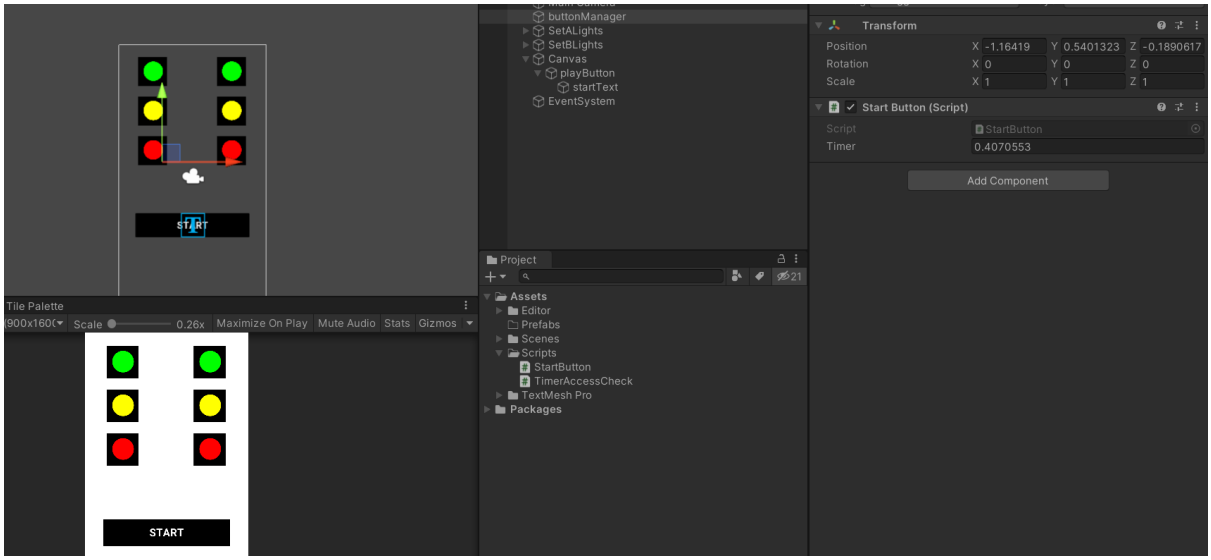
Developing the application

This section will document the process of which I make the traffic light application in Unity, and I will add updates along with annotations explaining what I am doing and how it is being done.

I have begun making my application in Unity, and so far, it looks like this, without any code elements



Here I have added in a script that starts the button and I was testing to see if it worked properly.



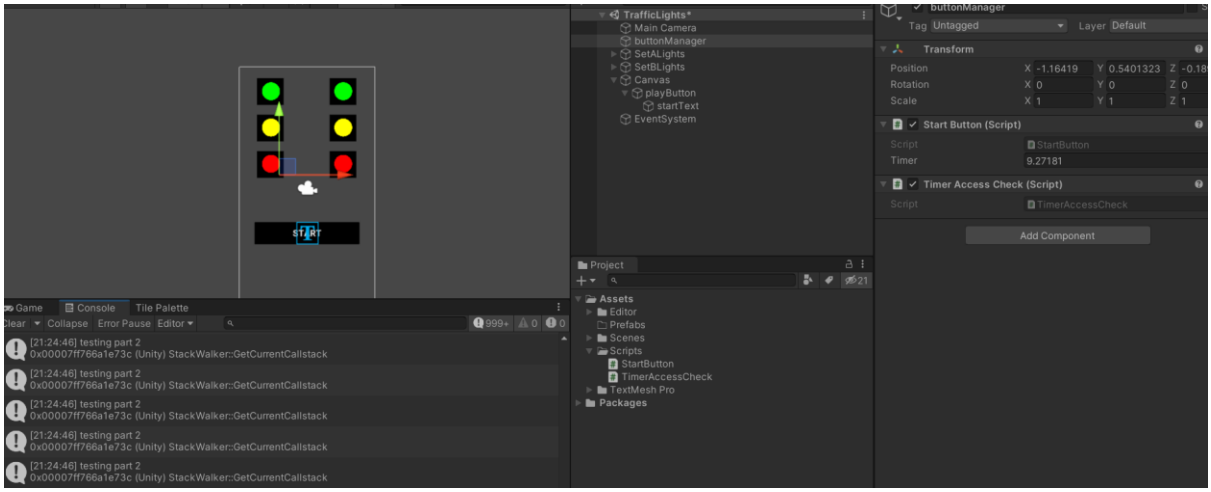
My code for my button so far

```
Unity Script | 0 references
public class StartButton : MonoBehaviour
{
    #region Public Variables
    public float timer; //Sets up a private float called "timer"
    #endregion
    #region Inspector Variables
    #endregion
    #region Private Variables
    #endregion
    #region Components
    #endregion


    Unity Message | 0 references
    private void Update()
    {
        timer += Time.deltaTime; //Updates timer variable every frame equal to deltaTime
                                 //DeltaTime is a standardiser that ensures the timer
                                 //runs at the same speed everytime
    }

    Unity Message | 0 references
    private void OnGUI()
    {
        if (timer < 2)
        {
            Debug.Log("timer worked");
        }
    }

}
```

I am now working on checking if the timer is accessible from other scripts so I can attach them to the traffic lights.

```
TimerAccessCheck.cs  StartButton.cs*
Assembly-CSharp | TimerAccessCheck | OnGUI()

1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
   Unity Script | 0 references
5  public class TimerAccessCheck : MonoBehaviour
6  {
7      StartButton timerCheck;
8
       Unity Message | 0 references
9      private void Start() //runs on start
10     {
11         timerCheck = GetComponent<StartButton>(); //sets timerCheck to be "GetComponent<StartButton>()
12     }
13
14
       Unity Message | 0 references
15     private void OnGUI() //GUI elements
16     {
17         if (timerCheck.timer > 5) //checks if the timer for the other script is over 5
18         {
19             Debug.Log("testing part 2"); //sends a debug log after 5s to see if its working
20         }
21     }
22 }
23
```

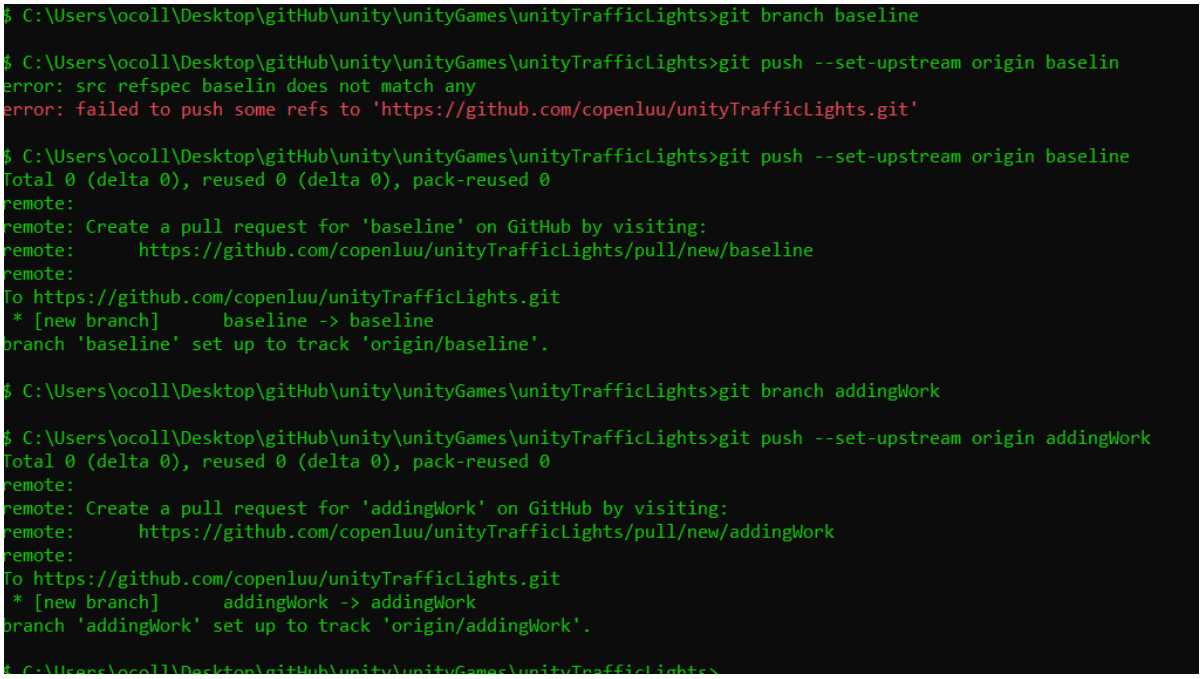
It worked! Though I think that I will have to change the final code to be a bit more reliable.



I have managed to develop a system where the traffic light changes based on time values decided in the inspector by the user and this can be used later on with all the traffic lights, albeit with a lot of repetition, to make all the lights change colour accordingly.

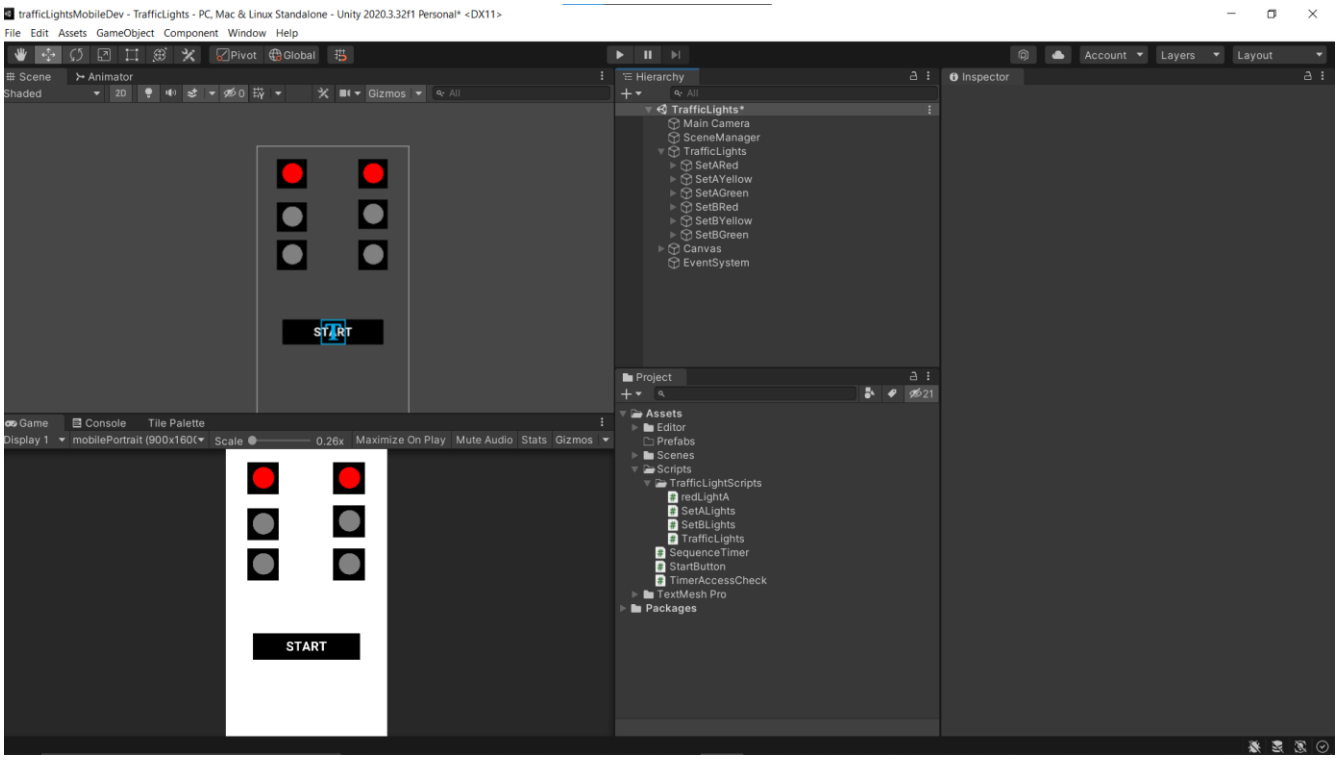
```
public class redLightA : MonoBehaviour
{
    #region Public Variables
    #endregion //these "regions" are all collapsible features that I always include to help organise my code
    Inspector Variables
    #region Private Variables
    private SpriteRenderer self;
    #endregion
    #region Components
    private StartButton timerCheck; //
    private SequenceTimer timerChanges; // these two reference the scripts and set them to have shortcuts
    #endregion


    Unity Message | 0 references
    private void Start()
    {
        self = gameObject.GetComponent<SpriteRenderer>(); //sets up shortcut for self to be getting the gameObjects Sprite renderer. This allows me to change the colour
        timerCheck = GameObject.FindGameObjectWithTag("SceneManager").GetComponent<StartButton>(); //Gets the script so it can access the timer to check
        timerChanges = GameObject.FindGameObjectWithTag("SceneManager").GetComponent<SequenceTimer>(); //Gets the script so it can access the timer sequence steps
    }

    Unity Message | 0 references
    private void Update()
    {
        if (timerCheck.timer > timerChanges.step1TL ) //if the timer value is greater than the "step 1 traffic light" then it runs the code. Same thing as writing
                                                      //GameObject.FindGameObjectWithTag("SceneManager").GetComponent<StartButton>().timer > GameObject.FindGameObjectWithTag("SceneManager").GetComponent<SequenceTimer>().step1TL
        {
            self.color = Color.blue; //changes the colour to blue. used as test

        }

        if (timerCheck.timer > timerChanges.step2TL ) //same thing except checks for the second step
        {
            self.color = Color.red; //changes the colour back to red. again, test

        }
    }
}
```

I also set up an external repo as a way to track my changes and manage my work.



I have now begun to develop my application in more depth and began organising it properly.

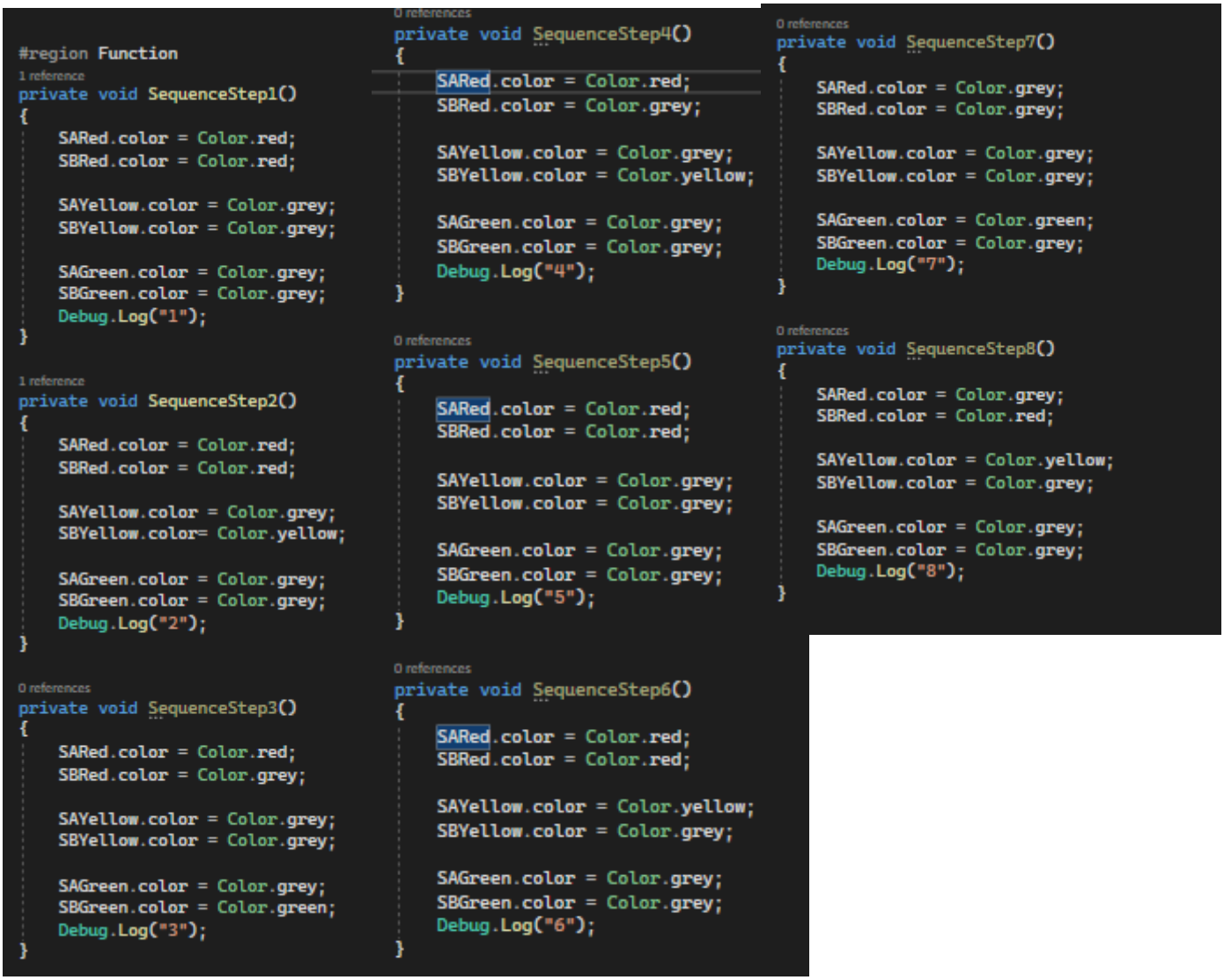
I decided that having all of my lights under one GameObject would be a cleaner way to make it, and to also have one script for all of my lights, rather than Set A and Set B, as this would lead to me making the same changes twice and having to write everything twice with minor adjustments, rather than just doing it all at once in one script. I learnt this when making a previous game where I had two separate player scripts and I had to microadjust everything twice when I wanted to make a change to the players or implement new features.



#

```
public class TrafficLights : MonoBehaviour
{
    #region Public Variables
    #endregion
    #region Inspector Variables
    #endregion
    #region Private Variables
    private SpriteRenderer SetARed;
    private SpriteRenderer SetBRed;
    private SpriteRenderer SetAYellow;
    private SpriteRenderer SetBYellow;
    private SpriteRenderer SetAGreen;
    private SpriteRenderer SetBGreen;
    #endregion
    #region Components
    private StartButton timerCheck;
    private SequenceTimer timerChanges;
    #endregion

    Unity Message | 0 references
    private void Start()
    {
        SetARed = GameObject.FindGameObjectWithTag("SARed").GetComponent<SpriteRenderer>(); //
        SetBRed = GameObject.FindGameObjectWithTag("SBRed").GetComponent<SpriteRenderer>(); //
        SetAYellow = GameObject.FindGameObjectWithTag("SAYellow").GetComponent<SpriteRenderer>(); //
        SetBYellow = GameObject.FindGameObjectWithTag("SBYellow").GetComponent<SpriteRenderer>();//
        SetAGreen = GameObject.FindGameObjectWithTag("SAGreen").GetComponent<SpriteRenderer>(); //
        SetBGreen = GameObject.FindGameObjectWithTag("SBGreen").GetComponent<SpriteRenderer>(); //I moved the script from the colour to the object its inside, so I am re-establishing the colour changing option
        timerCheck = GameObject.FindGameObjectWithTag("SceneManager").GetComponent<StartButton>(); //Gets the script so it can access the timer to check
        timerChanges = GameObject.FindGameObjectWithTag("SceneManager").GetComponent<SequenceTimer>(); //Gets the script so it can access the timer sequence steps
    }

    Unity Message | 0 references
    private void Update()
    {

    }
```

Next I began to make the functions which have the colours the lights are supposed to be when it changes to a certain step in the sequence.



```
#region Function
1 reference
private void SequenceStep1()
{
    SARed.color = Color.red;
    SBRed.color = Color.red;

    SAYellow.color = Color.grey;
    SBYellow.color = Color.grey;

    SAGreen.color = Color.grey;
    SBGreen.color = Color.grey;
    Debug.Log("1");
}

1 reference
private void SequenceStep2()
{
    SARed.color = Color.red;
    SBRed.color = Color.red;

    SAYellow.color = Color.grey;
    SBYellow.color= Color.yellow;

    SAGreen.color = Color.grey;
    SBGreen.color = Color.grey;
    Debug.Log("2");
}

0 references
private void SequenceStep3()
{
    SARed.color = Color.red;
    SBRed.color = Color.grey;

    SAYellow.color = Color.grey;
    SBYellow.color = Color.grey;

    SAGreen.color = Color.grey;
    SBGreen.color = Color.green;
    Debug.Log("3");
}
```

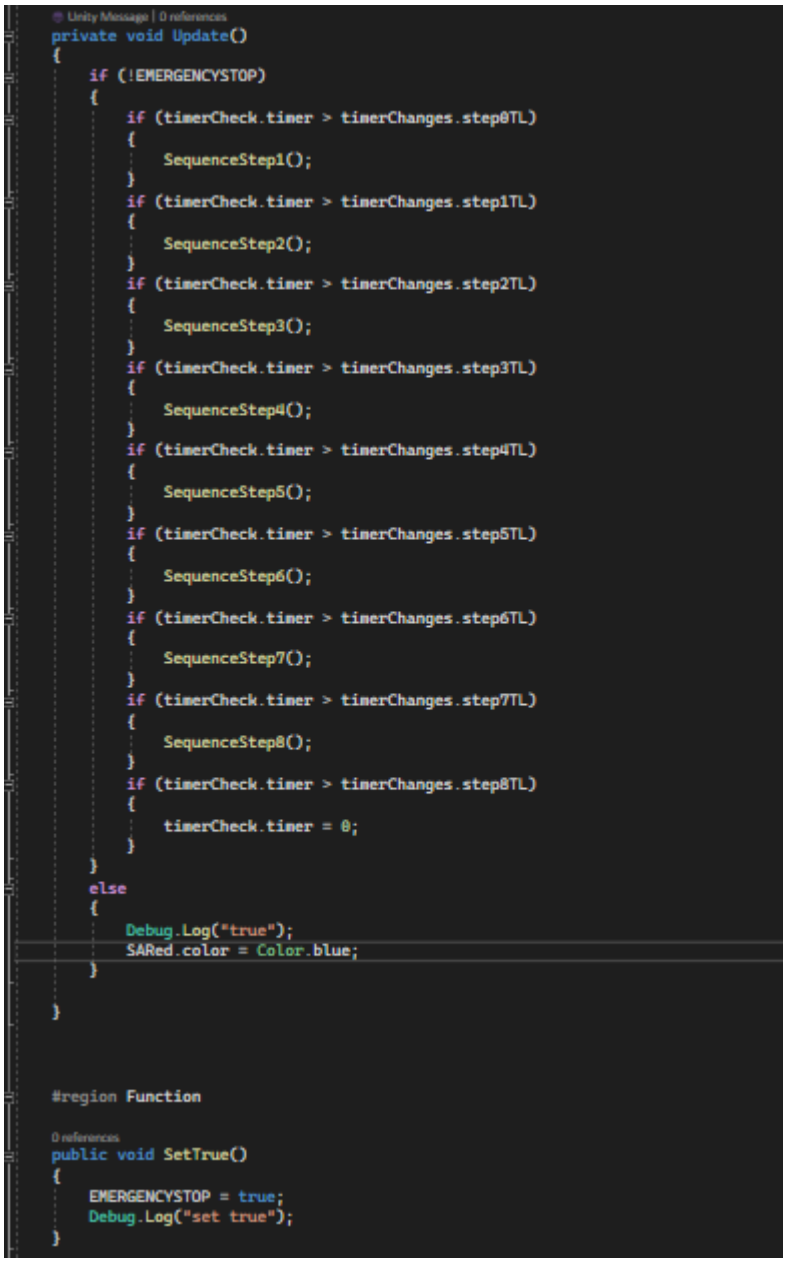
```
0 references
private void SequenceStep4()
{
    SARed.color = Color.red;
    SBRed.color = Color.grey;

    SAYellow.color = Color.grey;
    SBYellow.color = Color.yellow;

    SAGreen.color = Color.grey;
    SBGreen.color = Color.grey;
    Debug.Log("4");
}

0 references
private void SequenceStep5()
{
    SARed.color = Color.red;
    SBRed.color = Color.red;

    SAYellow.color = Color.grey;
    SBYellow.color = Color.grey;

    SAGreen.color = Color.grey;
    SBGreen.color = Color.grey;
    Debug.Log("5");
}

0 references
private void SequenceStep6()
{
    SARed.color = Color.red;
    SBRed.color = Color.red;

    SAYellow.color = Color.yellow;
    SBYellow.color = Color.grey;

    SAGreen.color = Color.grey;
    SBGreen.color = Color.grey;
    Debug.Log("6");
}
```

```
0 references
private void SequenceStep7()
{
    SARed.color = Color.grey;
    SBRed.color = Color.grey;

    SAYellow.color = Color.grey;
    SBYellow.color = Color.grey;

    SAGreen.color = Color.green;
    SBGreen.color = Color.grey;
    Debug.Log("7");
}

0 references
private void SequenceStep8()
{
    SARed.color = Color.grey;
    SBRed.color = Color.red;

    SAYellow.color = Color.yellow;
    SBYellow.color = Color.grey;

    SAGreen.color = Color.grey;
    SBGreen.color = Color.grey;
    Debug.Log("8");
}
```

This changes the colours of the lights according to the sequence required, and I will call the functions when they are needed.

My lights now change when needed, although I had to slightly alter the code as it was running the sequence one slower than it should have been, so now it starts from 0.
I have now begun implementing my emergency stop feature and I am doing so by making the main part of my traffic lights program only function if it fulfils an if statement, and then

The stop button makes it fail that if statement.

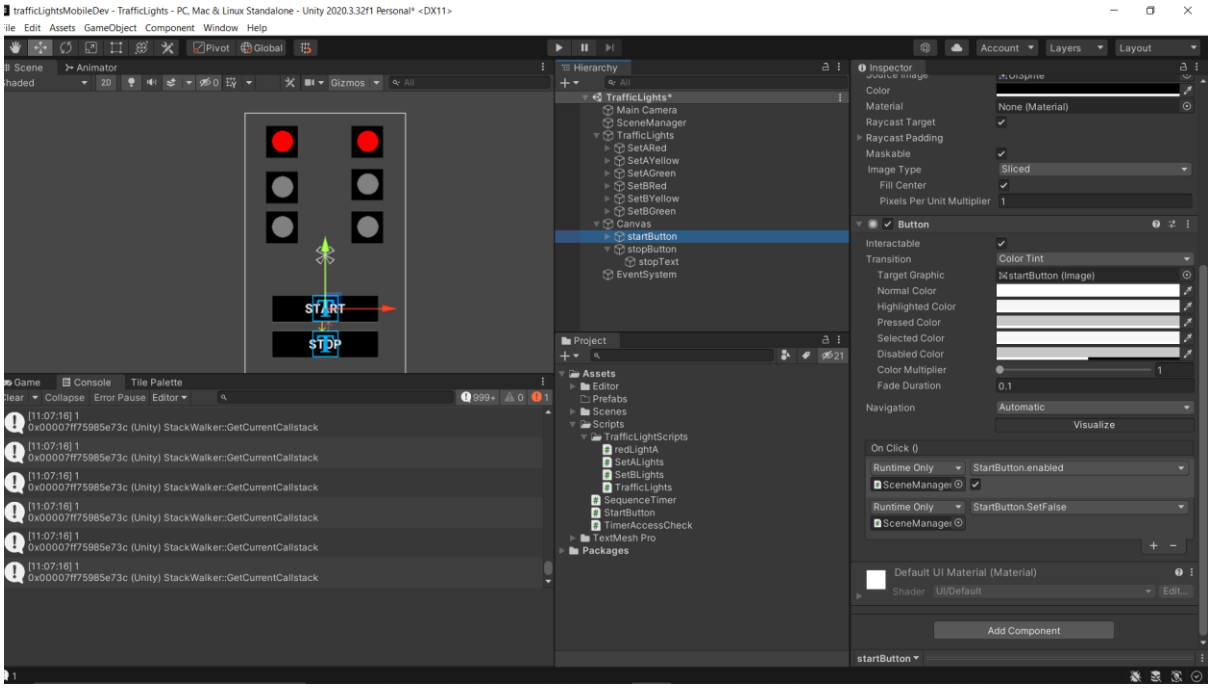


```
Unity Message | 0 references
private void Update()
{
    if (!EMERGENCYSTOP)
    {
        if (timerCheck.timer > timerChanges.step0TL)
        {
            SequenceStep1();
        }
        if (timerCheck.timer > timerChanges.step1TL)
        {
            SequenceStep2();
        }
        if (timerCheck.timer > timerChanges.step2TL)
        {
            SequenceStep3();
        }
        if (timerCheck.timer > timerChanges.step3TL)
        {
            SequenceStep4();
        }
        if (timerCheck.timer > timerChanges.step4TL)
        {
            SequenceStep5();
        }
        if (timerCheck.timer > timerChanges.step5TL)
        {
            SequenceStep6();
        }
        if (timerCheck.timer > timerChanges.step6TL)
        {
            SequenceStep7();
        }
        if (timerCheck.timer > timerChanges.step7TL)
        {
            SequenceStep8();
        }
        if (timerCheck.timer > timerChanges.step8TL)
        {
            timerCheck.timer = 0;
        }
    }
    else
    {
        Debug.Log("true");
        SARed.color = Color.blue;
    }

}



#region Function

0 references
public void SetTrue()
{
    EMERGENCYSTOP = true;
    Debug.Log("set true");
}
```

The part here for “SARED.color = Color.blue” was just to test if my if statement would work, in the actual program it will not return anything for the “else”.

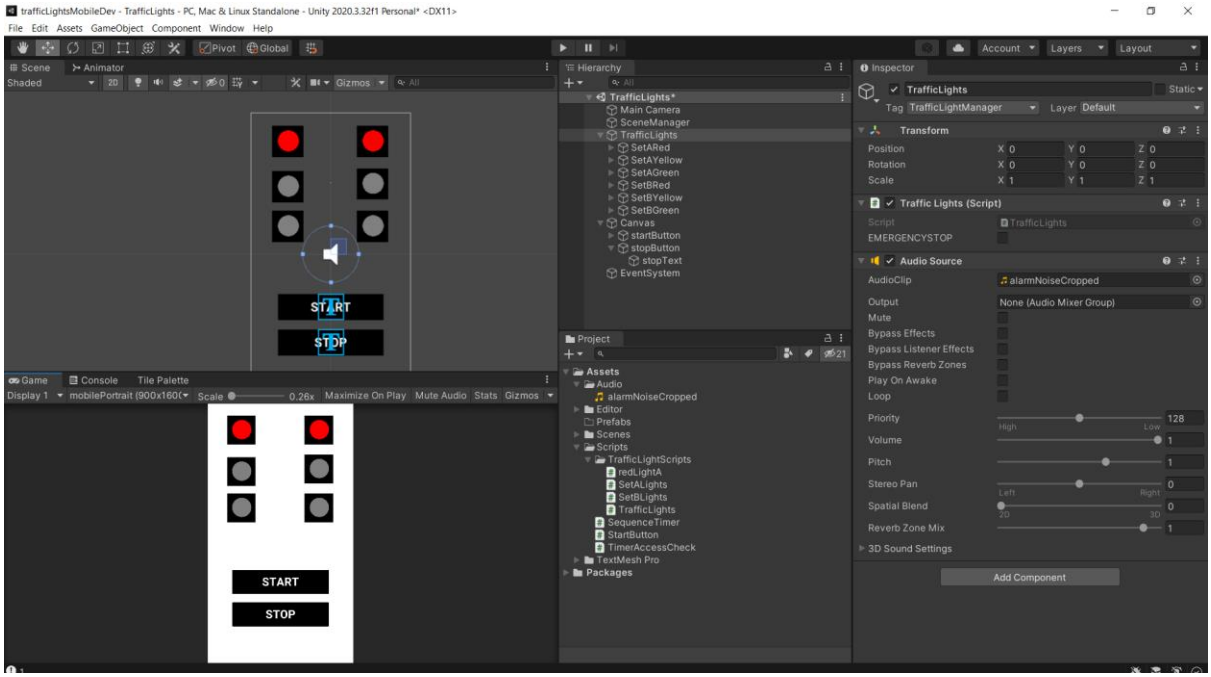
```
#region Private Variables
#endregion
#region Components
TrafficLights TLManager;
#endregion

Unity Message | 0 references
private void Start()
{
    TLManager = GameObject.FindGameObjectWithTag("TrafficLightManager").GetComponent<TrafficLights>();
}

Unity Message | 0 references
private void Update()
{
    timer += Time.deltaTime; //Updates timer variable every frame equal to deltaTime
                             //DeltaTime is a standardiser that ensures the timer
                             //runs at the same speed everytime
}

#region Functions
0 references
public void SetFalse()
{
    TLManager.EMERGENCYSTOP = false;
    timer = 0;
}
#endregion
```

This is the code which allows me to change the emergency stop bool value, which is then assigned to the start button. It also resets the timer so that the program continues from the start.

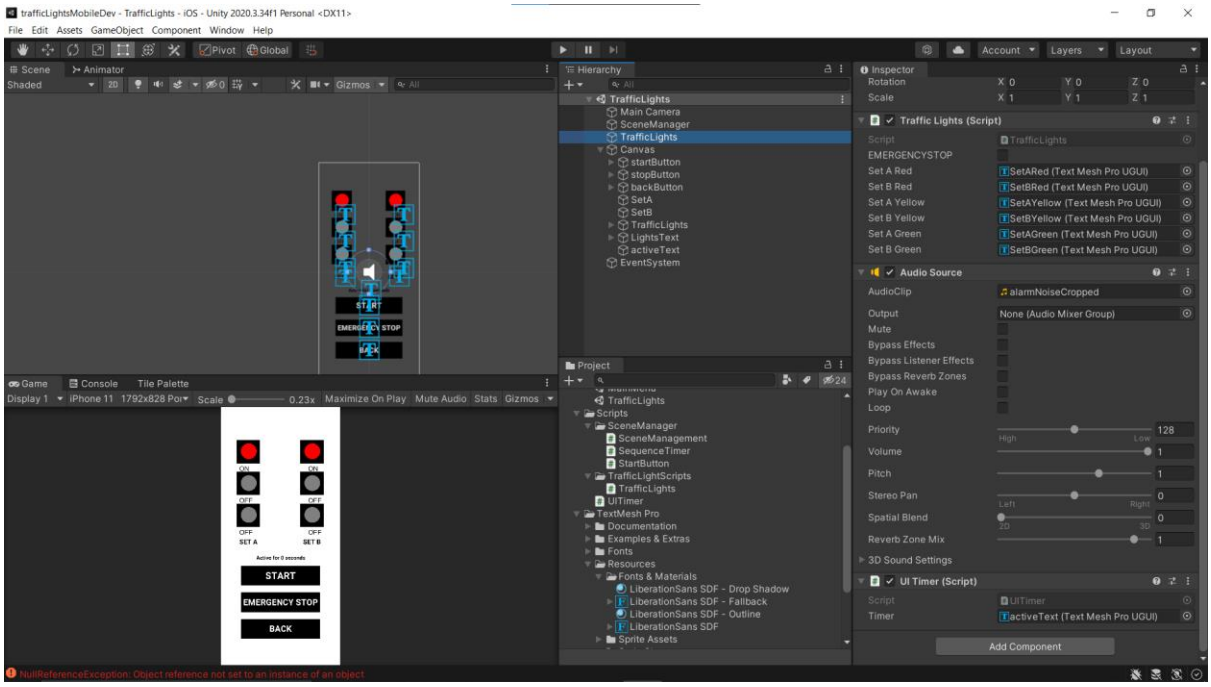


All the stop button does is call the “set true” function which sets emergency stop to true and therefore stops the program running.

I will now focus on implementing the sound for the emergency stop to alert the user.

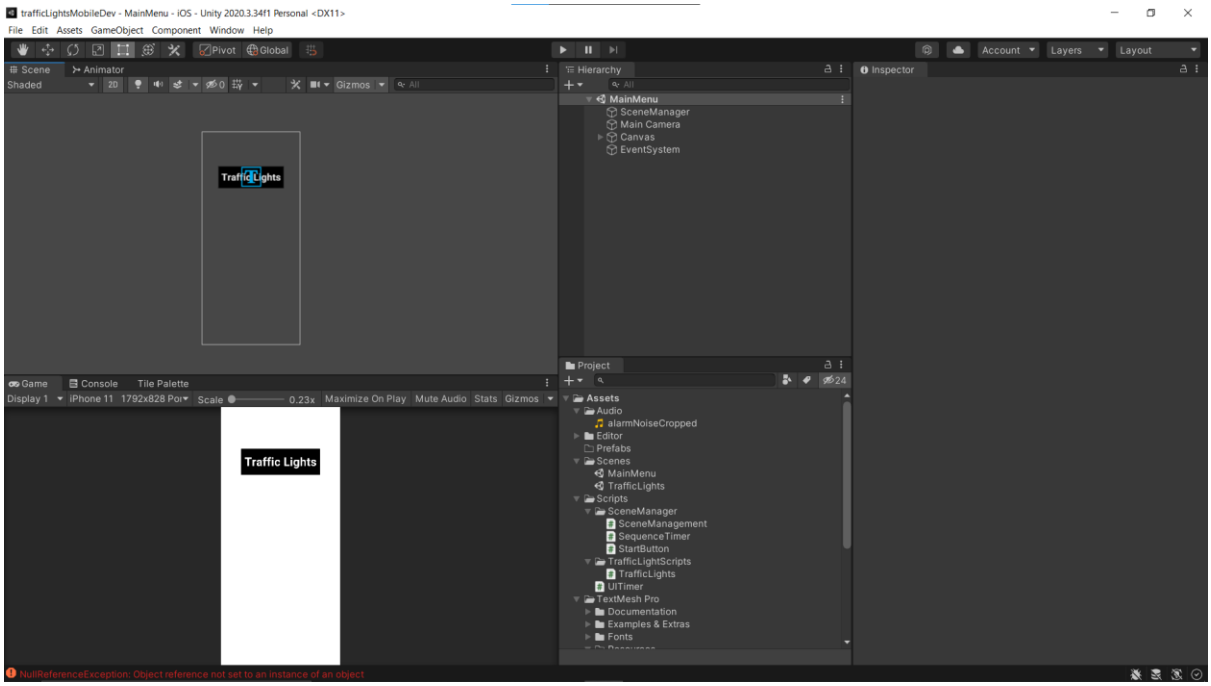


Here I have added the audio component which runs when the user presses the stop button.



In the screenshot above I have added the “Active for x seconds” to the script so the user has an idea for how long the traffic lights have been running since they started.

I also added text which tells the user whether the light is on or off, for accessibility options.



Here I am developing the main menu for the application that the user loads into when they open the app.

```
Unity Script (1 asset reference) | 0 references
public class UITimer : MonoBehaviour
{
    #region Public Variables
    #endregion
    #region Inspector Variables
    [SerializeField] private TextMeshProUGUI timer;
    #endregion
    #region Private Variables
    private float timeKeep;
    #endregion
    #region Components
    private StartButton timerGet;
    #endregion

    Unity Message | 0 references
    private void Start()
    {
        timerGet = GameObject.FindGameObjectWithTag("SceneManager").GetComponent<StartButton>();
    }

    Unity Message | 0 references
    private void FixedUpdate()
    {
        UpdateTimer();
    }

    #region Functions
    1 reference
    private void UpdateTimer()
    {
        timeKeep = timerGet.constTimer;
        int timeKeepInt = Mathf.RoundToInt(timeKeep);
        timer.text = "Active for " + timeKeepInt.ToString() + " seconds";
    }
    #endregion

}
```

Here I am making the code for the “Active for x seconds”, which first gets the timer that does not reset after the cycle is complete, and then rounds the value of it to an integer value, from which it applies to the timer.text and updates into the application.

```
Unity Script (1 asset reference) | 4 references
public class StartButton : MonoBehaviour
{
    #region Public Variables
    public float timer; //Sets up a private float called "timer"
    public float constTimer;
    #endregion
    Inspector Variables
    Private Variables
    Components

    Unity Message | 0 references
    private void Start()
    {
        TLManager = GameObject.FindGameObjectWithTag("TrafficLightManager").GetComponent<TrafficLights>();
    }

    Unity Message | 0 references
    private void Update()
    {
        timer += Time.deltaTime; //Updates timer variable every frame equal to deltaTime
                                 //DeltaTime is a standardiser that ensures the timer
                                 //runs at the same speed everytime
        constTimer += Time.deltaTime;
    }

    #region Functions
    0 references
    public void SetFalse()
    {
        TLManager.EMERGENCYSTOP = false;
        timer = 0;
        constTimer = 0;
    }
    #endregion

}
```

Here is the updated Start Button script.

```
#region Functions


public void SetTrue()
{
    EMERGENCYSTOP = true;
    alarm.Play();

    timerCheck.timer = 0;

    SARed.color = Color.red;
    SBRed.color = Color.red;
    SetARed.text = "ON";
    SetBRed.text = "ON";
    SAYellow.color = Color.grey;
    SBYellow.color = Color.grey;
    SetAYellow.text = "OFF";
    SetBYellow.text = "OFF";
    SAGreen.color = Color.grey;
    SBGreen.color = Color.grey;
    SetAGreen.text = "OFF";
    SetBGreen.text = "OFF";
}
```

And here is the function that defines what happens when “emergency stop” is pressed. It first sets the bool value to true so it stops the program from running, and resets the timer to 0. It also plays the alarm noise that alerts the users that the emergency stop has been pressed, and then resets the traffic lights.

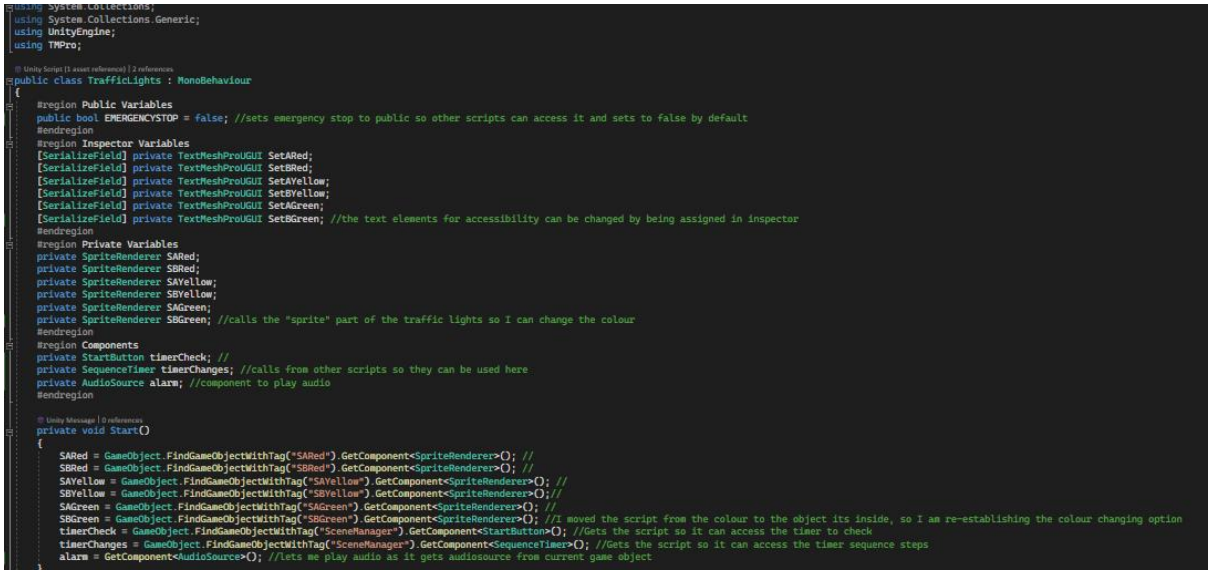
```
public class SequenceTimer : MonoBehaviour
{
    public int step0TL;
    public int step1TL;
    public int step2TL;
    public int step3TL;
    public int step4TL;
    public int step5TL;
    public int step6TL;
    public int step7TL;
    public int step8TL;
    public int resetTimer;

    private void Start()
    {
        step0TL = 0;
        step2TL += step1TL;
        step3TL += step2TL;
        step4TL += step3TL;
        step5TL += step4TL;
        step6TL += step5TL;
        step7TL += step6TL;
        step8TL += step7TL;
    }
}
```

This script defines how long the timer waits until it changes the colour, and the constant repetition of “x += y” is there so users can input the time differences without having to calculate what second they want the lights to change, and only have to add how long they want the lights to be a certain step in the sequence. On start, the script automatically calculates the differences between all of them.

Adding comments

In order to make my code easier to maintain and update, I will now go through and add comments to all of my scripts, and then showcase that here.

First is the traffic lights script, which is rather big so there will be a few screenshots.



```
```

```
Unity Message | 0 references
private void Update()
{
    if (!EMERGENCYSTOP)
    {
        if (timerCheck.timer > timerChanges.step0TL)
        {
            SequenceStep1();
        }
        if (timerCheck.timer > timerChanges.step1TL)
        {
            SequenceStep2();
        }
        if (timerCheck.timer > timerChanges.step2TL)
        {
            SequenceStep3();
        }
        if (timerCheck.timer > timerChanges.step3TL)
        {
            SequenceStep4();
        }
        if (timerCheck.timer > timerChanges.step4TL) //changes traffic lights depending on what the value of "timer" is, and if its larger than the next step
        {
            SequenceStep5();
        }
        if (timerCheck.timer > timerChanges.step5TL)
        {
            SequenceStep6();
        }
        if (timerCheck.timer > timerChanges.step6TL)
        {
            SequenceStep7();
        }
        if (timerCheck.timer > timerChanges.step7TL)
        {
            SequenceStep8();
        }
        if (timerCheck.timer > timerChanges.step8TL)
        {
            timerCheck.timer = 0; //resets to 0 to start loop all over again
        }
    }

}
```

```
#region Functions

0 references
public void SetTrue() //function to set emergency stop true. assigned to "emergency stop" button
{
    EMERGENCYSTOP = true;
    alarm.Play(); //plays alarm audio clip component, aka alarm noise

    timerCheck.timer = 0;

    SARed.color = Color.red;
    SBRed.color = Color.red;
    SetARed.text = "ON";
    SetBRed.text = "ON";
    SAYellow.color = Color.grey;
    SBYellow.color = Color.grey;
    SetAYellow.text = "OFF";
    SetBYellow.text = "OFF";
    SAGreen.color = Color.grey;
    SBGreen.color = Color.grey;
    SetAGreen.text = "OFF";
    SetBGreen.text = "OFF";
}

1 reference
private void SequenceStep1() //a lot of repetition. runs through sequence and changes to what is needed for that sequence.
{
    SARed.color = Color.red;
    SBRed.color = Color.red;
    SetARed.text = "ON";
    SetBRed.text = "ON";

    SAYellow.color = Color.grey;
    SBYellow.color = Color.grey;
    SetAYellow.text = "OFF";
    SetBYellow.text = "OFF";

    SAGreen.color = Color.grey;
    SBGreen.color = Color.grey;
    SetAGreen.text = "OFF";
    SetBGreen.text = "OFF";


}
```

Next is the start button script.

```
public class StartButton : MonoBehaviour
{
    #region Public Variables
    public float timer; //Sets up a private float called "timer"
    public float constTimer; //Same
    #endregion
    #region Inspector Variables
    #endregion
    #region Private Variables
    #endregion
    #region Components
    TrafficLights TLManager; //calls traffic light script
    #endregion

    Unity Message | 0 references
    private void Start()
    {
        TLManager = GameObject.FindGameObjectWithTag("TrafficLightManager").GetComponent<TrafficLights>();  //gets traffic light script from TrafficLightManager
    }

    Unity Message | 0 references
    private void Update()
    {
        timer += Time.deltaTime; //Updates timer variable every frame equal to deltaTime
                                 //DeltaTime is a standardiser that ensures the timer
                                 //runs at the same speed everytime
        constTimer += Time.deltaTime;
    }

    #region Functions
    0 references
    public void SetFalse() //assigned to "start" button
    {
        TLManager.EMERGENCYSTOP = false;
        timer = 0;
        constTimer = 0;
    } //When called, sets emergency stop to false and sets the timers to 0
    #endregion

}
```

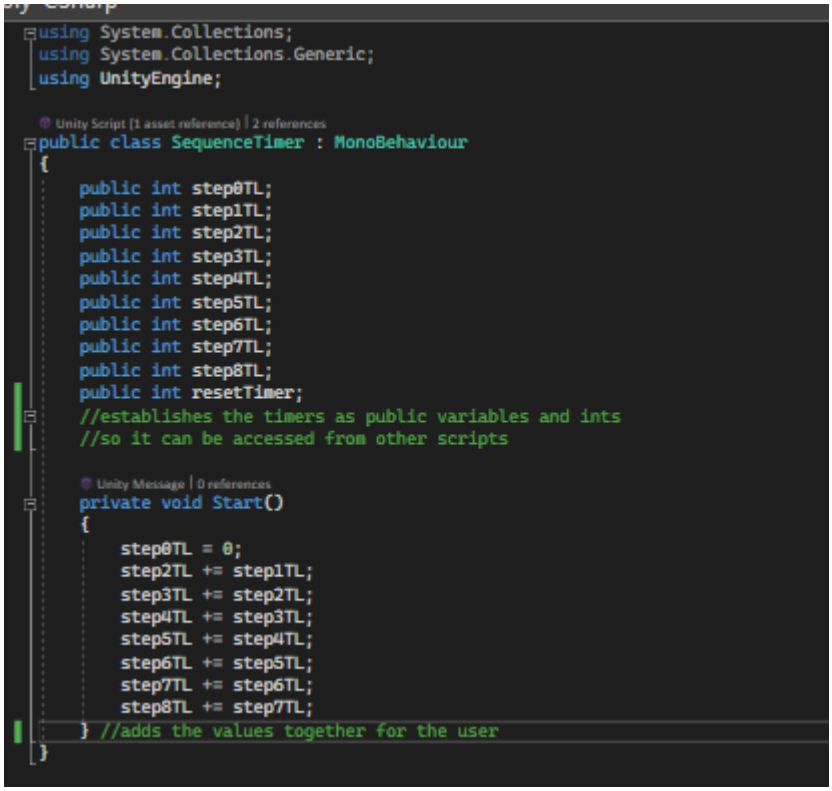
Next is the timer that appears on the actual traffic light application.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;

Unity Script (1 asset reference) | 0 references
public class UITimer : MonoBehaviour
{
    #region Public Variables
    #endregion
    #region Inspector Variables
    [SerializeField] private TextMeshProUGUI timer; //exposes to inspector so unity knows what "timer" is and what text to change
    #endregion
    #region Private Variables
    private float timeKeep; //priv variable especially for this script to keep track of the Timer that doesnt change when emergency stop is pressed
    #endregion
    #region Components
    private StartButton timerGet; //calls start button script so it can be used
    #endregion

    Unity Message | 0 references
    private void Start()
    {
        timerGet = GameObject.FindGameObjectWithTag("SceneManager").GetComponent<StartButton>(); //loads start button script from scene manager and shortens to "timerGet"
    }

    Unity Message | 0 references
    private void FixedUpdate()
    {
        UpdateTimer();
    }

    #region Functions
    1 reference
    private void UpdateTimer()
    {
        timeKeep = timerGet.constTimer;
        int timeKeepInt = Mathf.RoundToInt(timeKeep);
        timer.text = "Active for " + timeKeepInt.ToString() + " seconds";
    } //rounds the value of timeKeep to an integer and updates text with new number everytime it changes
    #endregion
```

Next, the script that allows me to load the main menu and the traffic light scenes.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

Unity Script (3 asset references) | 0 references
public class SceneManagement : MonoBehaviour
{
    0 references
    public void ChangeScene(string name)
    {
        SceneManager.LoadScene(name);
    } //changes the scene based on the scene name that the user writes on the function in unity
}
```

And finally, the script that allows me to change what step the traffic lights are on based on assigning the times.



```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script (1 asset reference) | 2 references
public class SequenceTimer : MonoBehaviour
{
    public int step0TL;
    public int step1TL;
    public int step2TL;
    public int step3TL;
    public int step4TL;
    public int step5TL;
    public int step6TL;
    public int step7TL;
    public int step8TL;
    public int resetTimer;
    //establishes the timers as public variables and ints
    //so it can be accessed from other scripts

    Unity Message | 0 references
    private void Start()
    {
        step0TL = 0;
        step2TL += step1TL;
        step3TL += step2TL;
        step4TL += step3TL;
        step5TL += step4TL;
        step6TL += step5TL;
        step7TL += step6TL;
        step8TL += step7TL;
    } //adds the values together for the user
}
```

Currently, there are only 5 scripts in my program, and it is simple and easy to understand, especially with all the comments, and therefore easy to make changes to.

Feedback

Now I will be asking for feedback on my code and my program to see what peers think and what changes can be made to the code.

Mobile application testing

Having already developed and made my mobile application, and not relying on something like Xamarin or Android Studio, I am now able to begin testing and document my tests on the final product.

These tests will include:

- Functionality
- Acceptance
- Performance
- Usability
- Compatibility

Next I will have the application tested by someone else running on an older version of iOS, as opposed to these previous tests done with an Android phone.

- Another users iOS test

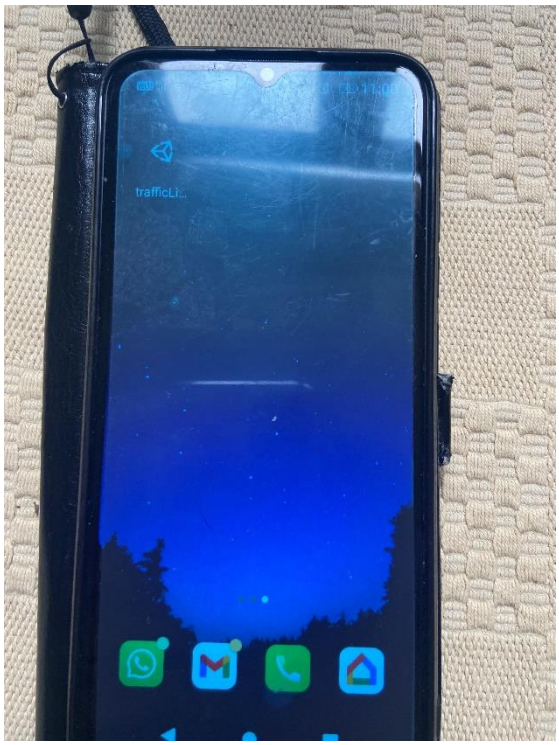
Then, I will ask for feedback, asking questions regarding UI and overall functionality of the app and more.

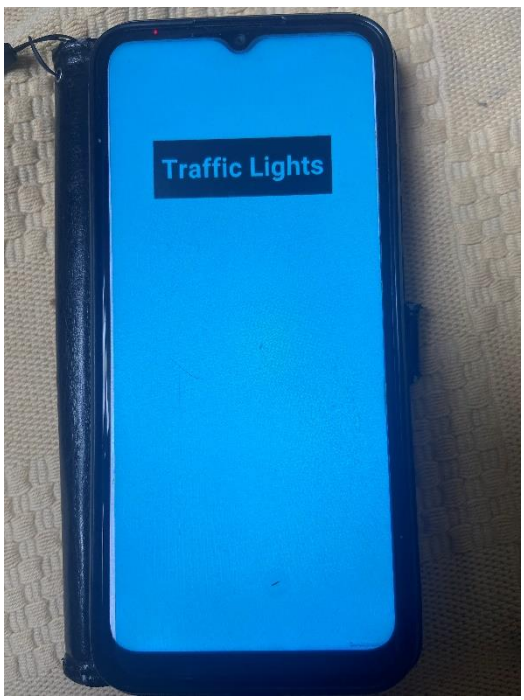
- Feedback

And finally, I will be making improvements based on this feedback if there is any available

- Improvements

Samsung Testing






ON
OFF
OFF
SET A
ON
OFF
OFF
SET B
Active for 0 seconds
START
EMERGENCY STOP
BACK


ON
OFF
OFF
SET A
ON
OFF
OFF
SET B
Active for 2 seconds
START
EMERGENCY STOP
BACK


ON
OFF
OFF
SET A
ON
ON
OFF
SET B
Active for 6 seconds
START
EMERGENCY STOP
BACK


ON
OFF
OFF
SET A
OFF
OFF
ON
SET B
Active for 9 seconds
START
EMERGENCY STOP
BACK


ON
OFF
OFF
OFF
SET A
OFF
OFF
ON
SET B
Active for 16 seconds
START
EMERGENCY STOP
BACK


ON
OFF
OFF
OFF
SET A
OFF
ON
OFF
SET B
Active for 19 seconds
START
EMERGENCY STOP
BACK


ON
OFF
OFF
SET A
ON
OFF
OFF
SET B
Active for 22 seconds
START
EMERGENCY STOP
BACK


ON
ON
OFF
SET A
OFF
OFF
SET B
Active for 26 seconds
START
EMERGENCY STOP
BACK


OFF
ON
OFF
OFF
ON
OFF
SET A
SET B
Active for 30 seconds
START
EMERGENCY STOP
BACK


OFF
ON
ON
OFF
OFF
OFF
SET A
SET B
Active for 39 seconds
START
EMERGENCY STOP
BACK


ON
OFF
OFF
SET A
ON
OFF
OFF
SET B
Active for 43 seconds
START
EMERGENCY STOP
BACK

Emergency Stop Testing

Functionality
The functionality of my application is perfectly fine, with all the stages and the emergency stop working as intended.

Acceptance
It fulfils the criteria of the assignment, a traffic light application.

Performance
Being a low-cost application, it runs perfectly fine without any performance issues.

Usability
This is clearly useable by all users, with a simple pressable interface designed to minimise mistakes.

Compatibility
It works on both Samsung and iOS devices, as well as windows. Shown below on iOS.

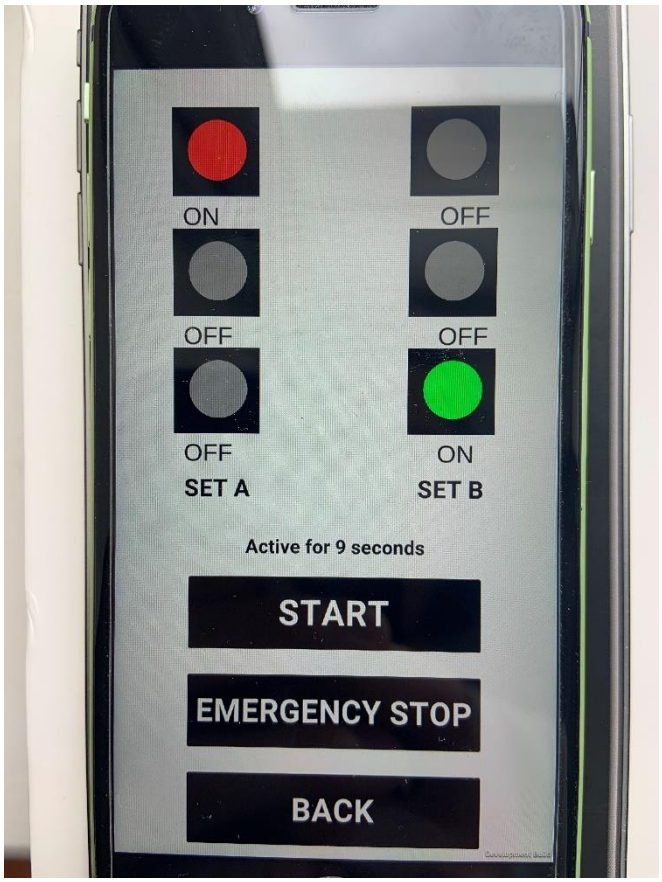
Another users iOS test
As shown in the screenshots below, the application managed to load and perform all of the different sequences just fine, without any issues. This also lines up with compatibility as it is now proven to work across two devices, not including windows on the computer.

The home screen with the app icon. I could add something to replace this in the future for a better and cleaner style.

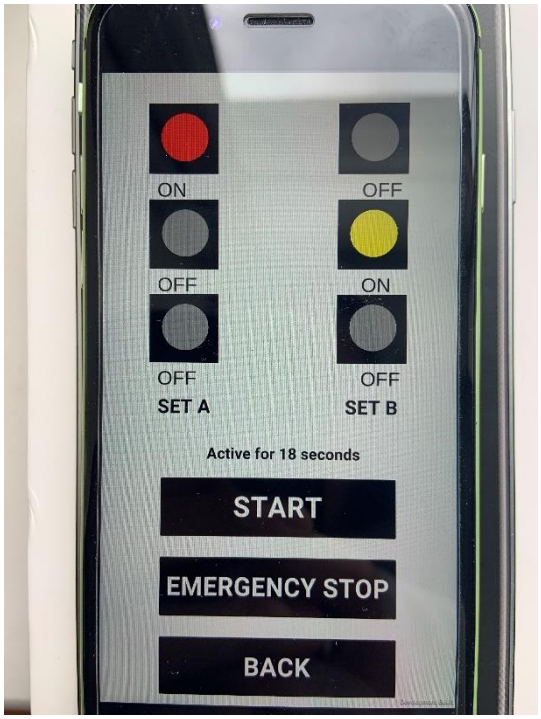
The main menu loaded on the iPhone.

The menu that the user loads into with the traffic lights on. Before any buttons are pressed.

The start button has been pressed and it has changed after passing 5 seconds.



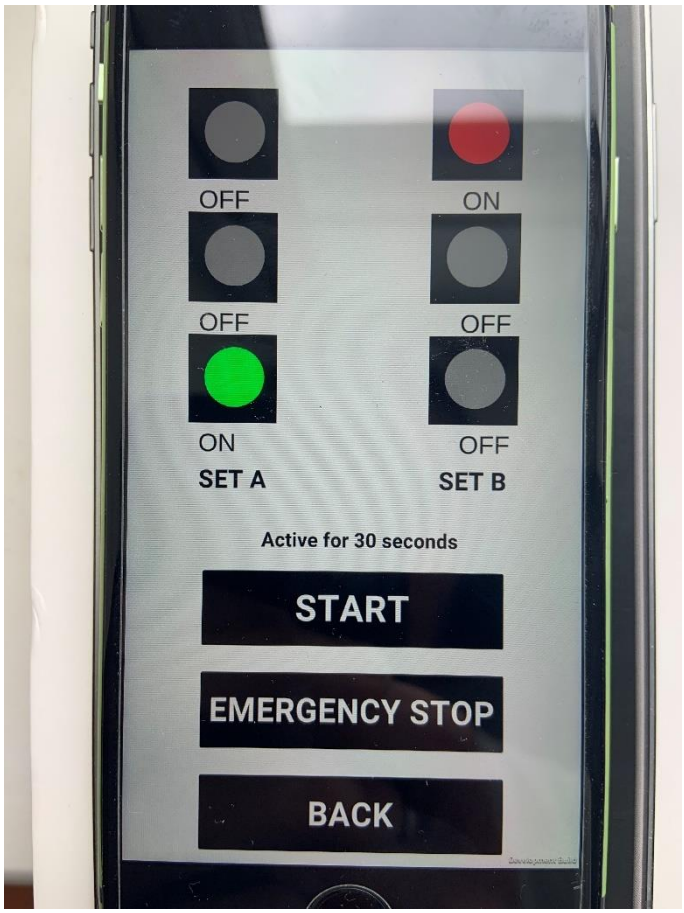
After passing a few more seconds, the green light is now on, with the text changed as well. Working as intended.



After the green light turns off after 10 seconds, back to yellow.

After a few seconds on yellow, it goes back to red for both.

From there, it starts the same sequence but on the other side.



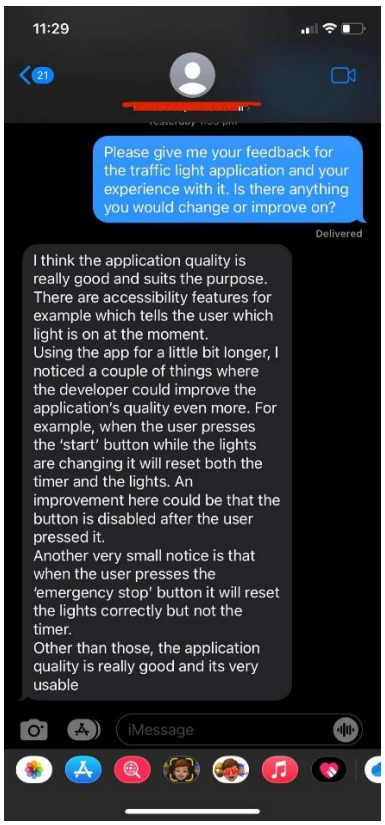
It then changes to green as intended.

Finally, the emergency stop is tested, which can be proven as the value for the timer has not changed, and even though the program should be in Set A Green, it is currently both on red.

Now I will ask for feedback on the application from the user who tested it on iOS, along with general feedback on the UI from other users.

Feedback

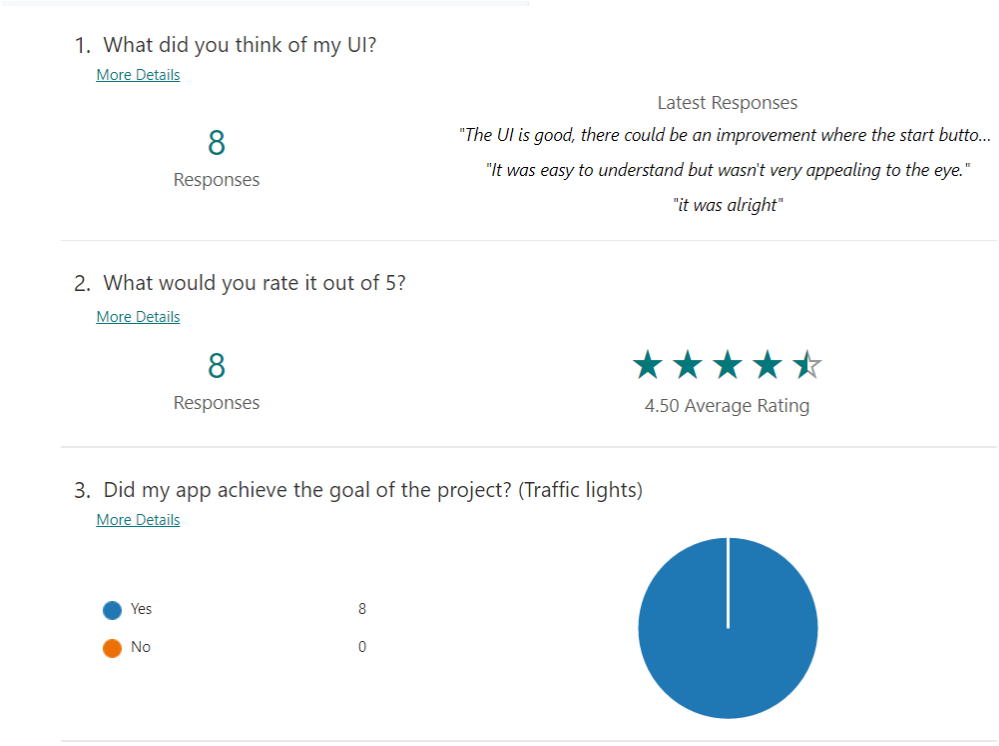
The feedback from the user who tested on iOS.



In regards to the things that are a part of my application, the timer not resetting when the user presses the emergency stop is a feature, as it allows for users to see how long their traffic lights were running for before they had to stop, and therefore will not be changed.

On the other hand, I agree with the review that the start button can be pressed multiple times, and I will make an effort to change this.

Next I asked for generalised feedback from others who had tested the application, not just the iOS user.



With regards to my application not being appealing to the eye, I believe that a simpler design and view will be easier to understand and less complicated, minimising human error.

1. What did you think of my UI?

8 Responses

ID ↑	Name	Responses
1	Tudor Bejan	Very user friendly
2	Raul-Alberto Prisecaru	The UI looks very user-friendly which is great for new users trying out the app for the first time.
3	Milad Karimi	preeeeetty good
4	Ahmed Ahmed	It was amazing
5	Matthew Deloughery	Very informative and self explanitory
6	Luka Radosavljevic	it was alright
7	George Evans	It was easy to understand but wasn't very appealing to the eye.
8	Tamas Tokics	The UI is good, there could be an improvement where the start button is still enabled after its already started. Other than that it looks really good

The last feedback which includes the mention of the button is very valid, and I will work to improve my application to ensure that the stop button and the start button can only be active when the other has been pressed.

4. Did you notice any bugs?

More Details

8

Responses

Latest Responses

"If the user presses the emergency stop button before the start button,..."

"No"

"no"

5. What other features would you add?

More Details

8

Responses

Latest Responses

"n/a"

"Music"

"nothing"

6. How would you rate the functionality? e.g. working as intended

More Details

8

Responses

Latest Responses

"10/10 if the button will be disabled"

"10/10 IGN"

"pretty good"

I personally would not add any music, as I do not believe it fits with this application.

4. Did you notice any bugs?

8 Responses

ID ↑	Name	Responses
1	Tudor Bejan	not any I was aware of
2	Raul-Alberto Prisecaru	N/A
3	Milad Karimi	no
4	Ahmed Ahmed	None
5	Matthew Deloughery	Nope
6	Luka Radosavljevic	no
7	George Evans	No
8	Tamas Tokics	If the user presses the emergency stop button before the start button, it wont start

The above comment I made will fix this bug.

7. How would you rate the acceptance? e.g. fit for purpose

More Details

8
Responses

Latest Responses

"10/10"

"It does what intended 10/10 IGN"

"yes "

8. How would you rate the performance? e.g. stress loading

More Details

8
Responses

Latest Responses

"10/10"

"10/10 Performs well"

"good"

9. How would you rate the usability? e.g. easy to complete tasks

More Details

8
Responses

Latest Responses

"10/10"

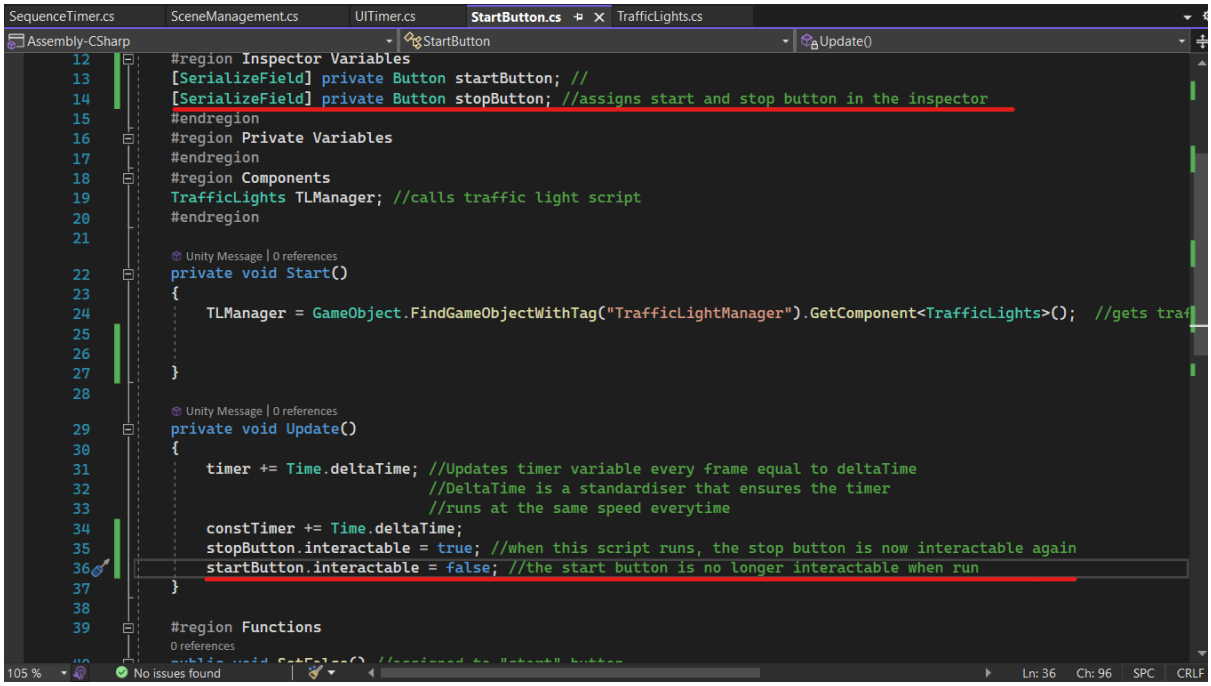
"10/10 Easy to use."

"good"

I am glad that the app functions just fine, and I will work to improve it from here.

Improvements

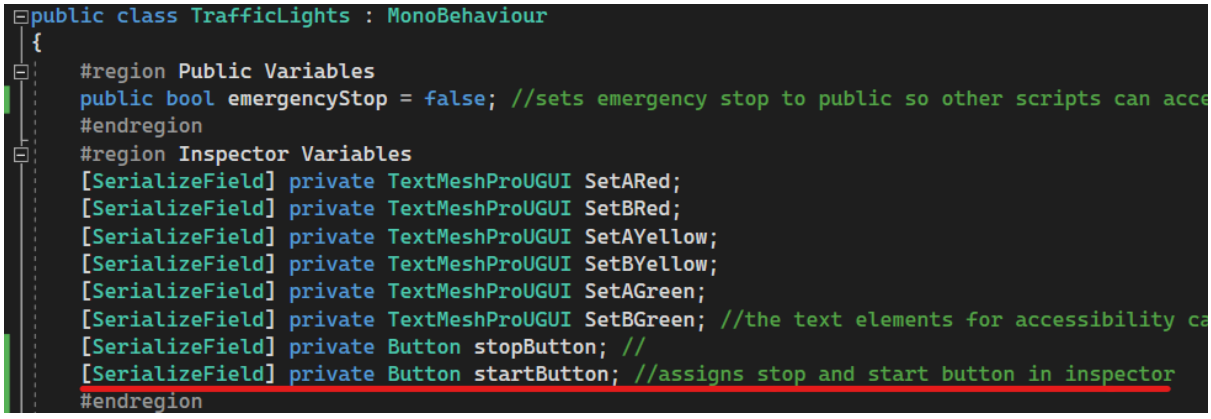
Here I am making the above improvements on my code and my program.



```
#region Inspector Variables
[SerializeField] private Button startButton; //
[SerializeField] private Button stopButton; //assigns start and stop button in the inspector
#endregion
#region Private Variables
#endregion
#region Components
TrafficLights TLManager; //calls traffic light script
#endregion

Unity Message | 0 references
private void Start()
{
    TLManager = GameObject.FindGameObjectWithTag("TrafficLightManager").GetComponent<TrafficLights>();  //gets traf

}

Unity Message | 0 references
private void Update()
{
    timer += Time.deltaTime; //Updates timer variable every frame equal to deltaTime
                             //DeltaTime is a standardiser that ensures the timer
                             //runs at the same speed everytime
    constTimer += Time.deltaTime;
    stopButton.interactable = true; //when this script runs, the stop button is now interactable again
    startButton.interactable = false; //the start button is no longer interactable when run
}

#region Functions
0 references
```

I assigned the variables in the inspector, and then used the script to set them to be interactable or not depending on whether it was used or not.

For example, in this every time the frame updates, it will set stopButton.interactable to true, so that users can interact with it, and startButton.interactable to false so that users cannot spam the start button.



```
public class TrafficLights : MonoBehaviour
{
    #region Public Variables
    public bool emergencyStop = false; //sets emergency stop to public so other scripts can acce
    #endregion
    #region Inspector Variables
    [SerializeField] private TextMeshProUGUI SetARed;
    [SerializeField] private TextMeshProUGUI SetBRed;
    [SerializeField] private TextMeshProUGUI SetAYellow;
    [SerializeField] private TextMeshProUGUI SetBYellow;
    [SerializeField] private TextMeshProUGUI SetAGreen;
    [SerializeField] private TextMeshProUGUI SetBGreen; //the text elements for accessibility ca
    [SerializeField] private Button stopButton; //
    [SerializeField] private Button startButton; //assigns stop and start button in inspector
    #endregion
```

Here I did the same thing inside of the traffic lights script so I can affect the buttons when the emergency stop button is pressed.

```
0 references
public void SetTrue() //function to set emergency stop true. assigned to "emergency stop" button
{
    emergencyStop = true; //changed to camel naming convention
    alarm.Play(); //plays alarm audio clip component, aka alarm noise

    timerCheck.timer = 0;

    SARed.color = Color.red;
    SBRed.color = Color.red;
    SetARed.text = "ON";
    SetBRed.text = "ON";
    SAYellow.color = Color.grey;
    SBYellow.color = Color.grey;
    SetAYellow.text = "OFF";
    SetBYellow.text = "OFF";
    SAGreen.color = Color.grey;
    SBGreen.color = Color.grey;
    SetAGreen.text = "OFF";
    SetBGreen.text = "OFF";

    stopButton.interactable = false; //once this function is called, the stop button stops being interactable
    startButton.interactable = true; //the start button is now interactable again
}
```

I also changed the name to the correct naming convention.

Here the buttons to the exact opposite to the other script. When this function is called, which is when the stop button is pressed, the stop button is no longer interactable until the start button runs again, and the start button is now interactable again.

And finally for the code, I also changed the onscreen timer script as I noticed that the value for the timer would change half a second before the lights, and this was due to the script that updates the timer. It would round to the nearest integer as it originally starts as a float, however that meant that if it crossed the .5 value, it would update half a second too early. E.g.

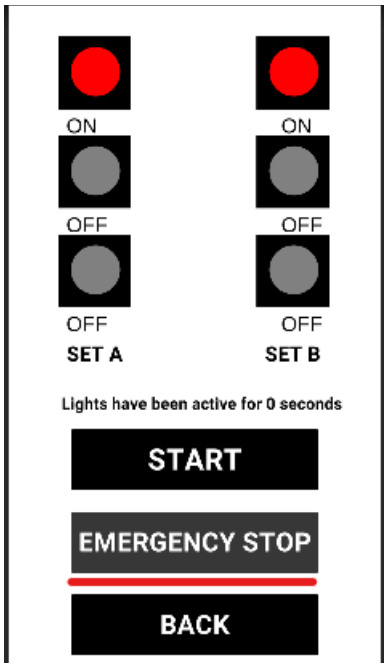
4.5 = 5 However the lights only change on 5, so while: 4.5 , 4.6 , 4.7 , 4.8 , 4.9 all took their time to run, the timer would have already changed to 5, while the lights are lagging behind.

```
Unity Script (1 asset reference) | 0 references
public class UITimer : MonoBehaviour
{
    #region Public Variables
    #endregion
    #region Inspector Variables
    [SerializeField] private TextMeshProUGUI timer; //exposes to inspector so unity knows what "timer" is a
    #endregion
    #region Private Variables
    private float timeKeep; //priv variable especially for this script to keep track of the Timer that does
    #endregion
    #region Components
    private StartButton timerGet; //calls start button script so it can be used
    TrafficLights TLManager;
    #endregion

    Unity Message | 0 references
    private void Start()
    {
        timerGet = GameObject.FindGameObjectWithTag("SceneManager").GetComponent<StartButton>(); //loads st
        TLManager = GetComponent<TrafficLights>();

    }

    Unity Message | 0 references
    private void FixedUpdate()
    {
        UpdateTimer();
    }

    #region Functions
    1 reference
    private void UpdateTimer()
    {
        timeKeep = timerGet.constTimer;
        int timeKeepInt = Mathf.FloorToInt(timeKeep);
        timer.text = "Lights have been active for " + timeKeepInt.ToString() + " seconds";

        if (TLManager.emergencyStop == true)
        {
            timer.text = "Lights were active for " + timeKeepInt.ToString() + " seconds";
        }

    } //rounds the value of timeKeep to an integer and updates text with new number everytime it changes
    #endregion
```

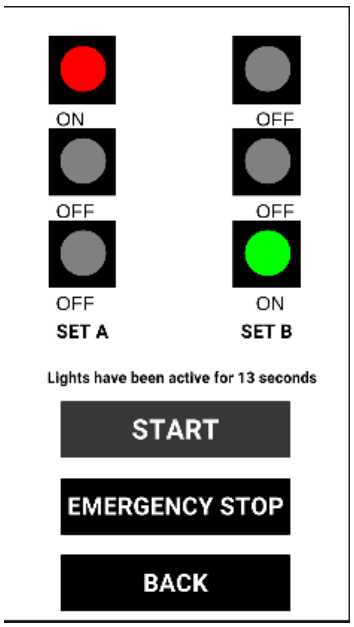
I fixed this by using “FloorToInt” instead of “RoundToInt”, which rounds down instead of the nearest integer.

I also added that the text for how long the lights were active for to change in order to make it clearer that it is not a bug, but a feature for the user to know.

In the actual program, the changes look like so:



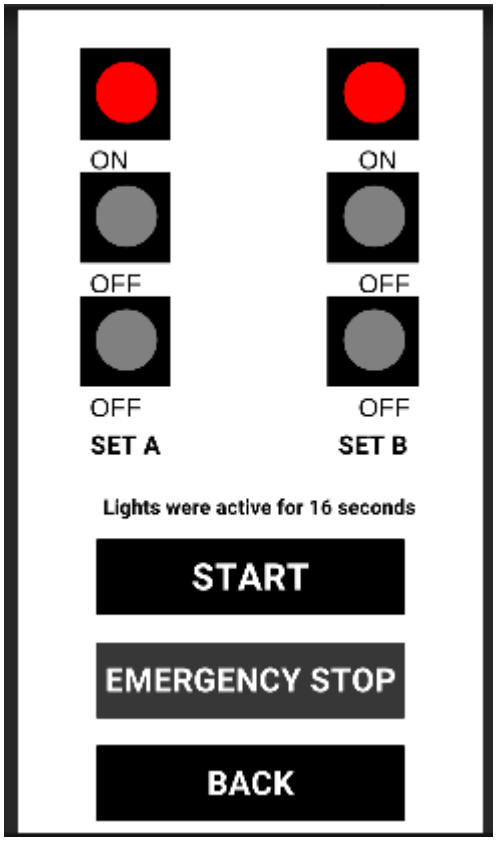
Here, emergency stop is slightly lighter than the rest of the buttons, in order to showcase how it is disabled.



Here the start button is slightly lighter now, showing how it is disabled.

In addition to this, the text has also changed.

Finally, here is the new look when the emergency stop is pressed.



The text has changed, and the emergency stop is disabled as shown by the colour.

These changes also prevent the bug appearing where the start button will not change the colour if the emergency stop button was pressed before the start button.

Review and evaluation of software solutions

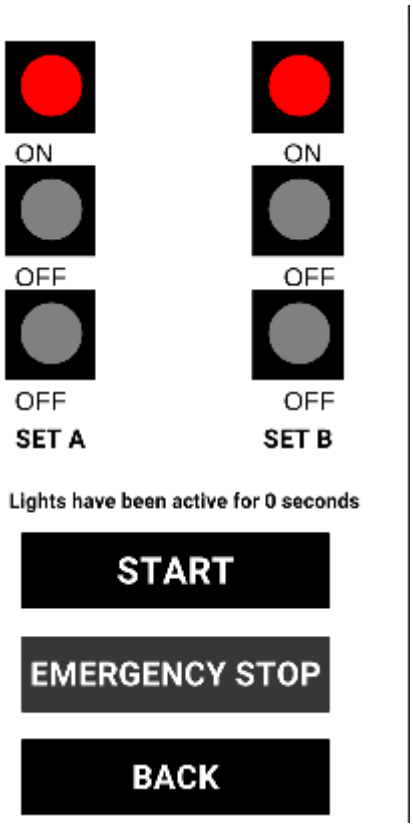
Now that my application has been made and I know it runs on both mobile and on desktop depending on what is needed, I will now be reviewing my software/program and evaluating

I will evaluate this based on:

- Suitability for audience and purpose
- Ease of use
- Quality of software solution
- Constraints of programming language
- Other constraints
- Strengths and weaknesses
- Improvements that could be made
- Optimising software solutions

Suitability for audience and purpose

Based on my audience and the purpose of my application, I believe that the final outcome of my application is good and effective. It runs on both iOS and Android, and remains simple in order to ensure that everyone, even those not knowledgeable on technology, are able to make use of it and therefore the software solution and application fulfils the suitability for my audience and purpose.



This is the final design of my application, which clearly labels which sets of the traffic lights as A and B and has the black backdrop with the labels for the buttons clearly outlining what they are supposed to do.

Ease of use

I believe that my program is very easy to use, needing just one button click to start it, and in the case of something going wrong, all that is needed is one button to stop the program. Due to the simplicity of my program and only two core buttons involved, I believe that it is easy to use, and does not require instruction.

Quality of software solution

This section includes

- Reliability
- Usability
- Efficiency/performance
- Maintainability
- Portability

Reliability

The chances of failure or something going wrong with my application are minimal, as the code required to run it and operate is very simplistic and runs autonomously from the moment that the start button is pressed, and therefore completely reduces the risk of human error breaking the program. In regard to the program breaking on its own, there is almost no risk of this as the code, although repetitive, has no space for bugs and works fine.

Usability

The usability of the program is highly optimised to perform the specified function, and therefore the usability is naturally high as well, successfully performing all required tasks and completing the job required of the program.

Efficiency/Performance

Considering the simplicity of the program, its performance and efficiency is maximised in order to ensure that it runs without issue and indefinitely until the user wants to stop the program.

Maintainability

Due to the comments in the code and the minimal amount of code, 5 scripts and roughly 380 lines of code, it would be easy to maintain and for any developer to pick up where I have left off and make changes as necessary to the program.

Portability

It is very easy to move my software to another computer or device, and to edit it, as long as the device has unity installed on it, which is the same as any other program and programming application/IDE.

Constraints of programming language

Some of the constraints of C# are:

- Poor x-platform GUI
- C# is an internal part of .NET framework, so it relies on windows-based applications to run
- C# is less flexible as it depends on .NET

Other constraints

As far as other constraints go, the most appropriate would programmer knowledge, as if I had more knowledge on C# and unity itself, or other programming applications, then I would have been able to explore options that work for a traffic light application, however I had to stick to this as I was certain this would work, and given that we were not provided an adequate opportunity to learn about other available options.

Strengths and weaknesses of software solution

Strengths

Some of the strengths include:

- Easy to build to mobile
- Works on desktop

- Easy to make changes
- You can use Unity's existing features
- Lots of existing code out there
  Support available

Weaknesses

Some of the weaknesses include:

- Limited by Unity
- Limited by C#
- Unity might not let you do exactly what you want to do
- Updating the application might be harder than expected
- Cannot update value of steps during the application, has to be done in Unity

Improvements

One of the improvements I would like to make in the future is that I would like to add another menu where users are able to input the values for the sequences, meaning how long the traffic lights steps are supposed to be, and update them inside of the application rather than in Unity.

Another feature I would like to add would be some sort of diagram or animation displaying cars passing on a street within the application, however I do not think this is necessary and purely for aesthetic purposes.

Optimising software solutions

One way I would optimise the software solutions, is by finding a way to remove the large amounts of repetitive code if it was possible, as there is currently a large amount of it, and therefore it may be confusing or easy to make mistakes if you are uncertain of what it refers to.

Another way would be to add more menu options, such as volume adjustments for the alert noise that plays in case it is too loud or vice versa.

Lessons learned from developing a mobile application

This section will cover:

- The extent to which the solution met the identified requirements
- Issues arising during testing and refinement
- How the final app could be improved to better meet the requirements of the users
- Alternative solutions if I was to repeat this task

The extent to which the solution met the identified requirements

The solution I came up with, my traffic light application, met all requirements, and also added my own features, such as a timer to tell the user how long the application has been running for. Therefore, I believe that my application goes above and beyond what was asked of me to develop.

Issues arising during testing and refinement

Some issues that arose during testing and refinement include the emergency stop button breaking the start button and not letting the lights change. I managed to fix this before I finished my application and therefore my issue was resolved.

How the final app could be improved

The final app might have been able to have been improved if I had included cars on my screen to help visualise to the user, or if I had added sounds to indicate that the lights were changing. Both of which I would use in the future.

Alternative solutions if I was to repeat this task

If I was to repeat this task, I would attempt to make it in Xamarin or Android Studio in order to challenge myself and discover other features and programming languages. Furthermore, I would also add sounds to the traffic lights changing and play audio files when the lights change if necessary for accessibility options for users.

Conclusion

In conclusion, I believe that my assignment and my program successfully fulfilled the criteria. I created a mobile traffic lights application that works on both pc and mobile, with lights that change depending on the sequence and a functioning emergency stop and start button. In addition to this I also added a timer that keeps track of how long the program has been running and therefore gone above and beyond what the assignment asked of me, and “enhanced user experiences” as asked for by the assignment. As a result of this, I believe that I have created an adequate solution to the traffic light problem that was assigned.