Stroop Test

Graduate Programmer Test

Brainstorming/Pre-Planning

Start a new game
Have a scene dedicated to starting a game
When the game has been started the timer should start

Colour word
Have a list of colour words
Get one at random
Check to see if the colour word doesn't match the visual colour
Then add the colour to the string (cw)

4 button options
A list of buttons
Keep the colour chosen add it to one of the buttons at random
Keep the word chosen add it to one of the buttons at random

Winning conditions
On button pressed()
Get the text
If the text equals colour then you win
Else try again

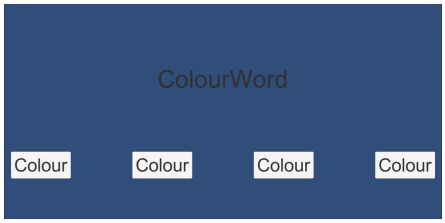
On win
On win change the scene to score scene
There will be a test box with the players score
There will also be a play again button that will use the same functionality as the **start a new game button**

Tracking Score

1. Score is kept by the time it takes the player to finish the test
2. Each attempt to correctly identify the colour must be timed.
3. The summation of the ten correct attempts must be averaged.
4. The average summation time must be displayed to the user after the tenth attempt

Day 1

I created the Unity projected along with a Github Repository. Then I proceeded to add game objects to the scene such as a Canvas so all my UI elements had a place to go. Lastly with GameObjects I added the correct UI elements shown in the reference picture.



After setting up my scene and adding a scripts folder, I created my first script called ColourWord.

The ColourWord Script handles the randomisation of the colour word and the button underneath. The script has the ability to have as many or as little colours you want in the Stroop Test by simply going to the Canvas GameObject and clicking on it. Then direct yourself to the inspector to view the ColourWord script attached.

```
public class ColourWord : MonoBehaviour
{
    [System.Serializable]
    private struct Colours
    {
        public string[] colourText;
        public Color[] colours;

    }
    [System.Serializable]
    private struct ButtonTexts
    {
        public Text[] buttonTexts;
        public Text correctButtonText;
    }

    [SerializeField] private Colours colourList;
    [SerializeField] private ButtonTexts buttonList;

    private List<int> usedColourWordNums = new List<int>();
    private int currentColourNum;
    private int currentColourWordNum;
    private int previousColourNum;

    private Text colourWord;

    // Start is called before the first frame update
    void Start()
    {
        colourWord = gameObject.transform.Find("Colour Word").GetComponent<Text>();
    }

    // Update is called once per frame
    void Update()
    {
        ColourWordTest();
    }

    private void SetColourWord()
    {
        int randColourWordNum = Random.RandomRange(0, colourList.colourText.Length);
        int randColourNum = Random.RandomRange(0, colourList.colours.Length);

        // Checks to see if the colour word is the same as the colour and
        // Checks to see if the next colour is the same as the previous colour
        if (randColourNum != randColourWordNum && randColourNum != previousColourNum)
        {
            for (int i = 0; i < colourList.colourText.Length; i++)
            {
                if (i == randColourWordNum)
                {
                    colourWord.text = colourList.colourText[i].ToString();
                    currentColourWordNum = i;
                }
            }
            for (int t = 0; t < colourList.colours.Length; t++)
            {
                if (t == randColourNum)
                {
                    colourWord.color = colourList.colours[t];
                    currentColourNum = t;
                    previousColourNum = randColourNum;
                }
            }
        }
        else
        {
            SetColourWord();
        }
    }
```

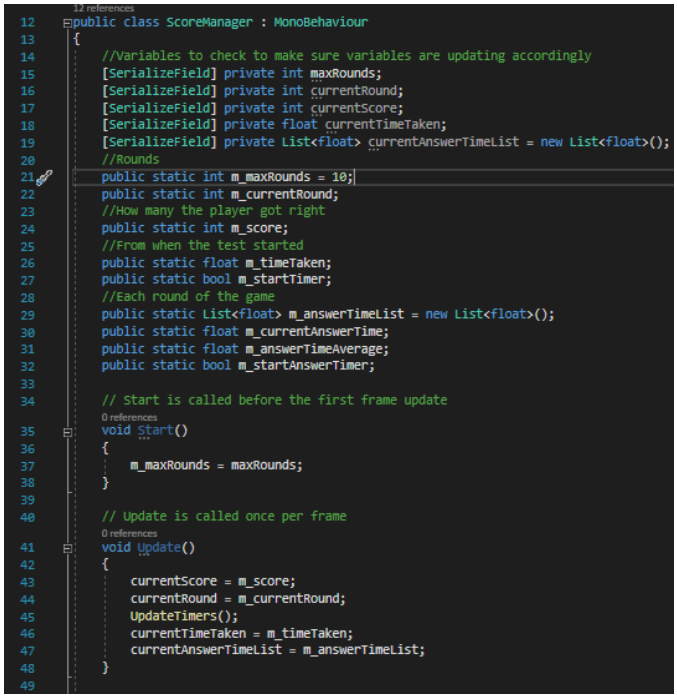
```
    private void SetButtons()
    {
        usedColourWordNums.Clear();
        for (int i = 0; i < buttonList.buttonTexts.Length; i++)
        {
            buttonList.buttonTexts[i].text = null;
        }

        usedColourWordNums.Add(currentColourNum);
        usedColourWordNums.Add(currentColourWordNum);

        for (int i = 0; i < buttonList.buttonTexts.Length; i++)
        {
            int randColour = Random.RandomRange(0, colourList.colourText.Length);

            // Checks to see if the number that was randomly generated has been used already
            if (!usedColourWordNums.Contains(randColour))
            {
                buttonList.buttonTexts[i].text = colourList.colourText[randColour].ToString();
                usedColourWordNums.Add(randColour);
            }
            else
            {
                i--;
            }
        }
        //Get a random button
        int randomButtonNumOne = Random.RandomRange(0, buttonList.buttonTexts.Length);
        // Set that button to be the correct word for the colour shown
        buttonList.buttonTexts[randomButtonNumOne].text = colourList.colourText[currentColourNum].ToString();

        // Get another random button
        int randomButtonNumTwo = Random.RandomRange(0, buttonList.buttonTexts.Length);
        // Checks if both button are not the same
        if (randomButtonNumOne != randomButtonNumTwo)
        {
            // Set this button to be the word shown
            buttonList.buttonTexts[randomButtonNumTwo].text = colourList.colourText[currentColourWordNum].ToString();
        }
        else
        {
            randomButtonNumTwo++;
            // Checks to see if the button is within the array length
            if (randomButtonNumTwo >= buttonList.buttonTexts.Length)
            {
                randomButtonNumTwo = randomButtonNumOne - 1;
            }
            buttonList.buttonTexts[randomButtonNumTwo].text = colourList.colourText[currentColourWordNum].ToString();
        }
    }


    private void ColourWordTest()
    {
        if (Input.GetKeyDown(KeyCode.E))
        {
            SetColourWord();
            SetButtons();
        }
    }
}
```

Day 2

To start off with day 2 I added a new script called Answer, for my Buttons to use when they are pressed by the user. In the Answer script it has one function to check whether the Text on the button is equal to the colourText String Array at the position of the colour number from the colours Color Array. If the texts match then you got the answer right if not you got it wrong.

```
0 references
public class Answer : MonoBehaviour
{
    private ColourWord colourWord;
    private Text buttonText;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        colourWord = transform.parent.GetComponent<ColourWord>();
        buttonText = transform.GetChild(0).GetComponent<Text>();
    }

    0 references
    public void CheckAnswer()
    {
        ScoreManager.EndTest();
        if (buttonText.text == colourWord.colourList.colourText[colourWord.currentColourNum])
        {
            Debug.Log("Correct");
            ScoreManager.ScoreTracker();
        }
        else
        {
            Debug.Log("Incorrect");
        }
        ScoreManager.m_startAnswerTimer = false;
        ScoreManager.AddAnswerTime();
        colourWord.GenerateTestLevel();
    }
}
```

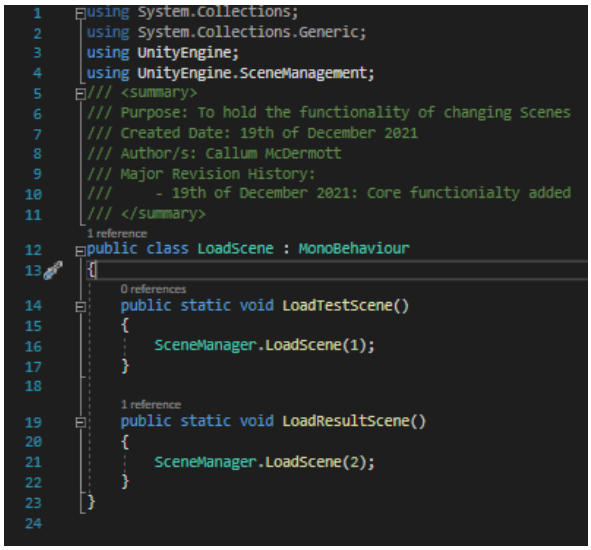
Then the next round is generated. The next step I took in making this Test was to add a way of scoring. Since I already had a way to tell when you got a question right or wrong it was fairly straight forward. I created a script named ScoreManager which I added a few ints and multiple functions. The first function was called ScoreTracker and all it does is adds one to m_score each time the function is called. I then made sure there wasn't an infinite amount of rounds by adding 2 ints m_maxRound and m_currentRound. You are able to set m_maxRound at say number 10 and m_currentRound adds 1 each time a new round is generated. I also added ways to track the time it takes you to do the entire test and your average round time.



```
12 references
public class ScoreManager : MonoBehaviour
{
    //Variables to check to make sure variables are updating accordingly
    [SerializeField] private int maxRounds;
    [SerializeField] private int currentRound;
    [SerializeField] private int currentScore;
    [SerializeField] private float currentTimeTaken;
    [SerializeField] private List<float> currentAnswerTimeList = new List<float>();
    //Rounds
    public static int m_maxRounds = 10;
    public static int m_currentRound;
    //How many the player got right
    public static int m_score;
    //From when the test started
    public static float m_timeTaken;
    public static bool m_startTimer;
    //Each round of the game
    public static List<float> m_answerTimeList = new List<float>();
    public static float m_currentAnswerTime;
    public static float m_answerTimeAverage;
    public static bool m_startAnswerTimer;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        m_maxRounds = maxRounds;
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        currentScore = m_score;
        currentRound = m_currentRound;
        UpdateTimers();
        currentTimeTaken = m_timeTaken;
        currentAnswerTimeList = m_answerTimeList;
    }
```

```
1 reference
public static void UpdateTimers()
{
    if (m_startTimer)
    {
        m_timeTaken += Time.deltaTime;
    }
    if (m_startAnswerTimer)
    {
        m_currentAnswerTime += Time.deltaTime;
    }
}

1 reference
public static void ResetScores()
{
    m_score = 0;
    m_currentRound = 0;
    m_timeTaken = 0;
    m_startTimer = true;
    m_answerTimeList.Clear();
}

1 reference
public static void EndTest()
{
    if (m_currentRound >= m_maxRounds )
    {
        m_startTimer = false;

        LoadScene.LoadResultScene();
    }
}

2 references
public static float ScoreTracker()
{
    return m_score++;
}

1 reference
public static float TimeTakenToComplete()
{
    return m_timeTaken;
}

1 reference
public static void AddAnswerTime()
{
    m_answerTimeList.Add(m_currentAnswerTime);
    m_currentAnswerTime = 0;
}

1 reference
public static float AverageTimeToAnswer()
{
    foreach (var item in m_answerTimeList)
    {
        m_answerTimeAverage += item;
    }

    return m_answerTimeAverage = m_answerTimeAverage / m_answerTimeList.Count;
}
}
```

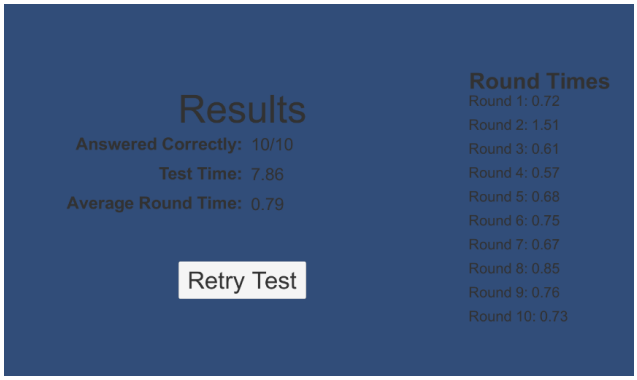
Finally throughout the day I was adding parts to a new script called LoadScene. LoadScene contains functions to load the TestScene and ResultScene.



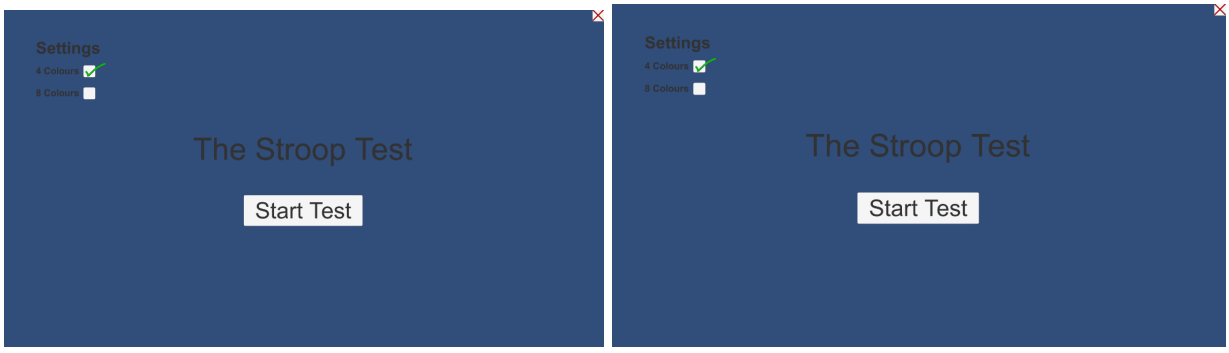
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
/// <summary>
/// Purpose: To hold the functionality of changing Scenes
/// Created Date: 19th of December 2021
/// Author/s: Callum McDermott
/// Major Revision History:
///     - 19th of December 2021: Core functionialty added
/// </summary>
1 reference
public class LoadScene : MonoBehaviour
{
    0 references
    public static void LoadTestScene()
    {
        SceneManager.LoadScene(1);
    }

    1 reference
    public static void LoadResultScene()
    {
        SceneManager.LoadScene(2);
    }
}
```

Day 3

Day 3 was mostly spent making my code more clean and efficient. I did this first by adding all the variables to do with each round of the test, such as the time it takes to complete each round and the round number into a struct called Round. I then created a list of Rounds so that we could keep the information of each round and display it in the ResultScene.



The last thing I did today was add a way to change from having 8 random colours and the 4 buttons changing accordingly.To having 4 random colours and the 4 buttons always set to the 4 default colours



Day 4

Day 4 was the last day I worked on this Test/Project I went through my code making sure it's presentable and clean. By double checking the names of variables and functions whilst adding tooltips to help designers. But before I did that I added another way to check out your score by simply highlighting the best round time, with another piece of text next to it saying "Best".