Dale Hunt

Student ID: 21839927

CI601: Computer Project

Accessibility Issues in Games

Supervisor: Robin Heath

Second Reader: Martin De Saulles

Abstract

This project explores the implementation of accessibility features within video games, with a specific focus on visual impairments such as colour blindness. The report revolves around the development of a tower defence game designed to demonstrate how different accessibility modifications can enhance gameplay for individuals with visual challenges. The project identifies and integrates three main accessibility features: colourblind modes, high contrast settings, and user-defined colours, each tailored to address the needs of players with varying types of colour vision deficiencies. Through a comprehensive development approach using the Unity game engine, the project evaluates the effectiveness of these features in improving game accessibility. The research also includes interviews with industry professionals and an analysis of existing games that have successfully incorporated similar accessibility options. The outcome aims to provide a framework that can be adapted by other developers to improve accessibility in different gaming genres with specific focus on the Unity game engine.

Table of Contents

Introduction

Accessibility Issues in Games is an incredibly broad field, there are plenty of accessibility issues to consider for all studios and some issues cannot be addressed appropriately. However, my project will be based on accessibility issues for the visually impaired. This term is just as broad, but it is primarily focused on those with colour-blindness or are unable to see certain elements of the user-interface.

This project will help highlight these issues by implementing features that are specifically designed to improve the user experience. In the below sections, I will make it clear how I will approach each accessibility issue and their corresponding solutions.

Additionally, I will be discussing the approaches that Game Engines currently implement but additionally how these can be both a help and a hinderance.

To highlight the visual impairments and the impacts of accessibility implementation, I am aiming to create a simple tower defence game. I am aiming for this genre as it is growing in popularity, but it also helps demonstrate what the project is based on.

GitHub Repository

For ease and exploration, I have provided a link to the GitHub repository where this project is stored. All issues – open and closed – as well as commits are available to view.

https://github.com/DaleHuntGB/TowerDefenseAccessibility

Aims, Objectives & Deliverables

Aims & Objectives

As I stated above, accessibility is a rather broad field and therefore, it would be almost impossible for every single game to include solutions to each accessibility issue that may be faced, this is largely to do with the fact that some features may not be particularly needed for that game or genre.

Therefore, I am going to be approaching this assessment with a particular focus on a tower defence game, I do believe that all the objectives I have listed below can be applied quite effectively to this genre and therefore, highlight how these features can improve the overall gaming experience for individuals who may be facing these issues.

Objective: Visual Impairment – Colourblind Modes

Create a feature rich colourblind mode option where the user can swap through commonly diagnosed colourblindness illnesses. As there are several types of colour-blindness's, I will be limiting to the three most common.

- Protanopia [Appendix #1]
- Deuteranopia [Appendix #2]
- Tritanopia [Appendix #3]

There is another type of colour-blindness, which is known as Monochromacy (Achromatopsia), which is incredibly rare (1 / 33 000), where sufferers are only able to see in different shades of grey, ranging from black to white.

(Colour Blind Awareness, 2022)

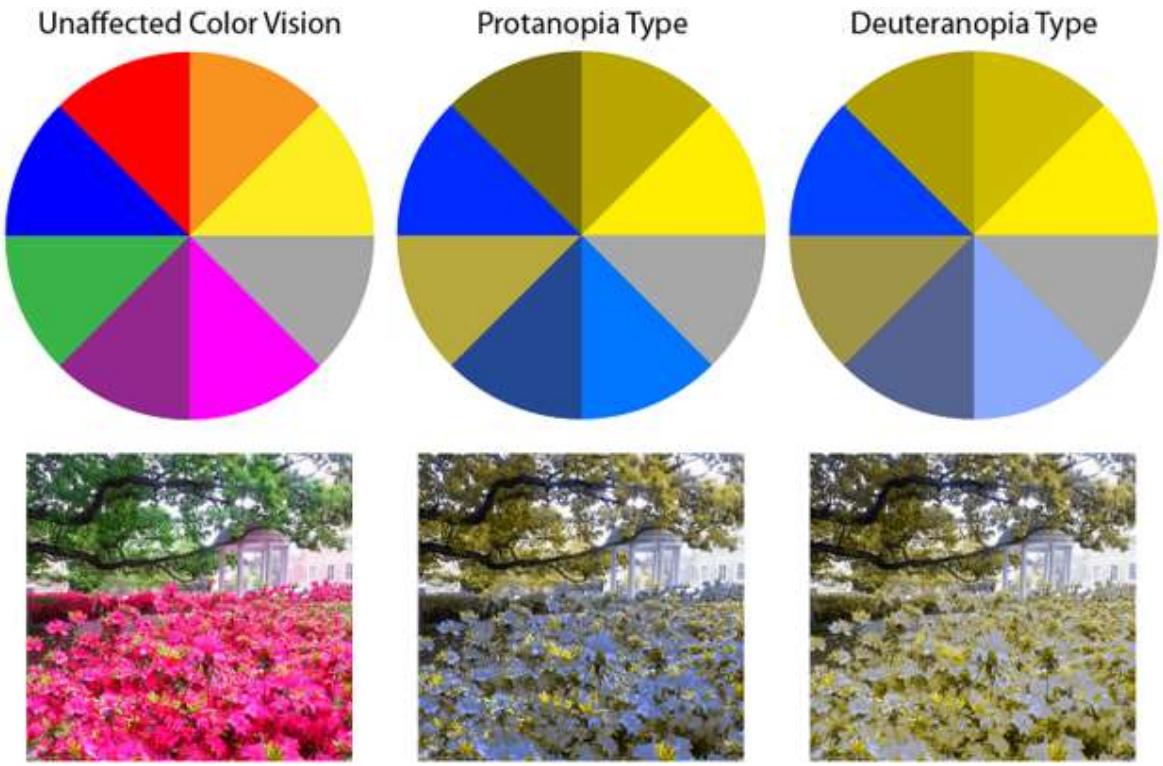


Figure 01: Colourblindness Colour Wheels

Objective: Visual Impairment – High Contrast

Visual impairments are not something that will be the same for every user, therefore, it is important to offer an alternative to best accommodate most users. Therefore, the other goal is incorporate a high contrast mode, this will allow for the users to be able to determine the difference between certain user-interface elements by merely adopting a high contrast difference between these elements.

The reason this is important is due to the fact that there is an incredibly rare colour blindness that will only allow users to see in grayscale, and therefore, colourblind modes will not be able to help them in this particular instance, however, users will still be able to determine the difference between high and low levels of contrast, therefore, allowing users to still interact with the game in some manner.

Figure 02: God of War – High Contrast Indication.

Objective: Visual Impairment – User Defined Colours

User defined colours is the best way to ensure that accessibility for all colourblind types is supported. This objective is the most complex, however, from a usability point, it is by far the most accessible and useful. This approach will give full access to the user to ensure that game objects can be converted to a colour that is easiest for them to interact with and visualize the game.

This objective will also require a custom colour picker that will be able to indicate to the user how the colour has been adapted as well as giving them freedom to change the colour however they feel fit.

Requirements

In this section, we will be covering what requirements are necessary for this project. I will be discussing how I decided that these should be the primary focus of this project, additionally, I will include ideas that I hope to accomplish before the end of this project, these will be specified appropriately below.

Visual Impairments

This requirement is the primary focus of this project and will be a section that I spend the most time implementing. As vision impairment is incredibly vast, I have decided to focus on three areas which are listed below.

Colourblind Modes Implementation

This is simply to add colourblind modes as a toggle for the player to utilize when playing the game. This should be the bare essential that all studios incorporate into their games, this will be important to develop throughout the course of the project as this is more complicated than how Game Engines incorporate it natively, this will be explained later in this report.

High Contrast Implementation

This focus is to create a naturally high contrast user-interface, rather than fixating on colourblind modes, I will create a user-interface that focuses primarily on developing high levels of contrast between important colours. An example of this can be found in Bloons TD 6, where they utilize contrast between teams so that players are easily able to establish where their team is and where the enemy is.

(Family Gaming Database, 2022)

User Defined Colour

The ultimate implementation for visual impairment accessibility is to create a user-interface where users can adjust their colours accordingly. This is a huge task and therefore is only a possible requirement but something that I will aim to incorporate as this will increase overall usability for my game and all games that incorporate this solution.

MoSCoW Prioritization

Must Have

- Colourblind Modes
  - Protanopia
  - Deuteranopia
  - Tritanopia

Should Have

- High Contrast Mode
- Greyscale / Monochromatic

Nice To Have

- User Colour Control

Deliverables

The final deliverable for this project will be a Unity Template that can be used by developers as a starting point. This template will include all managers, scenes, scripts for the developer to modify as they see fit for their game. The deliverable will be based on the tower defence genre merely for the ability to display the features implemented, however, can be modified to accommodate all genres.

In future updates, this template will be converted to not provide a genre starting point, therefore, giving more freedom and flexibility out of the box for the developer. All deliverables will require the developer to have a base understanding of C# and their required support for visual impairments.

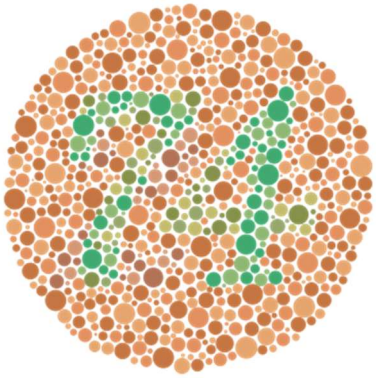
Problem Analysis

The topic I am exploring revolves around how video games address accessibility issues and requirements. It is interesting to note that some games, like Battlefield 3, introduced accessibility features quite some time after their initial release. Nowadays, game engines come with built-in support to encourage developers to incorporate these features right from the start. Not all accessibility features are necessary for every type of game. So, my research primarily focuses on adding essential accessibility options for developers to consider during their initial development.

Research

Colourblind Modes

Ishihara Colour Test



Ishihara Colour Test is a colour deficiency test that was created by Shinobu Ishihara, a professor from the University of Tokyo. This test is focused on the detection of red-green colour deficiencies and achieves this by creating several Ishihara plates – a type of pseudoisochromatic plate [Appendix #7]. Each plate depicts a solid circle of randomize coloured dots, within this pattern are dots that form a specific number or shape which are visible to normal vision users but almost invisible to those that have red-green deficiencies.

(Wikipedia Contributors, 2019)

Colourblind Mode in Fortnite

Fortnite includes three settings based on a player's colour vision deficiency. Players can choose between Protanopia, Deuteranopia and Tritanopia – which are the three most common types – and Fortnite also provide a severity slider, allowing players to customize the strength at which they want these colourblind modes to be applied.

Additionally, Fortnite also applies these changes to an Ishihara Colour Test, which allows for players to accurately adjust their colourblind mode settings. The importance of adding this test to the user-interface means that players will be able to make changes without having to constantly swap between the options and the game.

(AccessibleGames, 2023)

I used this example in my game since Fortnite utilizes this system to help players identify loot more easily as each loot type has a quality, quality is often highlighted by a particular colour which colour deficient players might find difficult to differentiate. As my game is based on team colours and identifying the difference between them, this test would be a vital addition to my artefact.

High Contrast

High Contrast is an approach that is used by some game studios for games, this approach uses shades of colours to help users identify different elements without having to rely on colours. The Last of Us: Part II is a great example of what can be achieved when accessibility is considered during development. The game has over 60 accessibility options - (Webster, 2020) - but the most impressive is the High Contrast implementation for those that are visually impaired. When toggled, the game will highlight main characters blue and enemies red, therefore becoming easily identified by these players.

Figure 03: Last of Us: Part II – High Contrast Mode

(CNN, 2020)

This article really helped me see the importance of including accessibility options for a wide range of individuals and impairments, this approach has changed the overall gaming atmosphere around this topic and equally increased the overall gaming experience for users with these impairments.

Interviews

All roles listed below were from a AAA gaming studio, however, they have asked to remain anonymous but are happy for me to use the interviews to help substantiate my report.

Lead UI Designer

I interviewed the Lead UI / UX Designer for the same AAA Gaming Studio; this individual gave me direction and inspiration to work on accessibility in general. This subject was of interest to me previously; however, this individual highlighted the importance of these issues in current gaming and non-gaming atmospheres and therefore this shifted my attention to helping create a more sustainable approach to solving these issues.

Games Accessibility Manager

I interviewed a Games Accessibility Manager for a AAA Gaming Studio; the primary role of this individual is to create programs within the studio to help increase awareness and better accessibility options throughout all the games they are planning on releasing in the future. This individual gave me incredibly good insight into what accessibility issues are and how they are approached by studios and shifted my focus of my project in a more positive direction.

Feedback #1

I wanted to incorporate different navigation implementations for the user-interface, however, after the interview above with the Games Accessibility Manager, they advised against this as it would also draw away from the possible direction that my project is taking. On this note, I have done some research regarding this, as there are specific guidelines that need to be followed - (kevinasg, 2022) – and in these guidelines it follows a huge range of accessibility and implementations that are required to create a fully-fledged, good user interface for both keyboard and controller.

This is accompanied by another two articles - (kevinasg, 2023) and (kevinasg, 2022b) – the first one being how focus is handled when utilizing different navigation tools. Focus implies how the user is aware of which user-interface item is currently highlighted so that correct selection can be made. It also goes into how this can be done and whether the method is sufficient for those with vision impairments. The second article is more in the direction of whether or not the user-interfaces have corrected structure and if navigation makes sense for the user.

If I was to do UI Navigation, these three articles would have provided great insight into how a good user-interface should be designed and all the guidelines that can be followed to ensure that it is both functional and aesthetic, whilst maintaining support for individuals that might suffer with vision impairment.

Feedback #2

After the initial interview, I continued with development and near the ending stages of this process, I reached out for some more direct feedback about my features and implementation. The feedback was incredibly positive, the main feedback was to swap from using medical terms to something more “understandable”, even individuals that suffer from certain colourblind illnesses are not aware of their medical terms.

Feedback #3

Lastly, he provided feedback with regards to the background image, “as it does not add a lot and usually a blank background is more accessible and usually, simplistic styles are better”. However, he did continue by adding that, “it is still great overall and what you have achieved is brilliant, just small improvements to be made”.

All of this feedback, although unable to impact my current project due to time constraints, is incredibly valuable and not only helps me further understand this complicated subject but also gives me approval that my approach, although not perfect, is a good starting point.

Methodology – Project Planning

Goal

Create an extendable and general solution to accessibility issues, specifically regarding visual impairments. The approach chosen should provide substantial flexibility to both the user and the developer. Due to the nature of this task, I will be focusing primarily on material colouring rather than textures. Textures are an incredibly complex implementation and would require an individual with relevant art techniques to help complete the artefact.

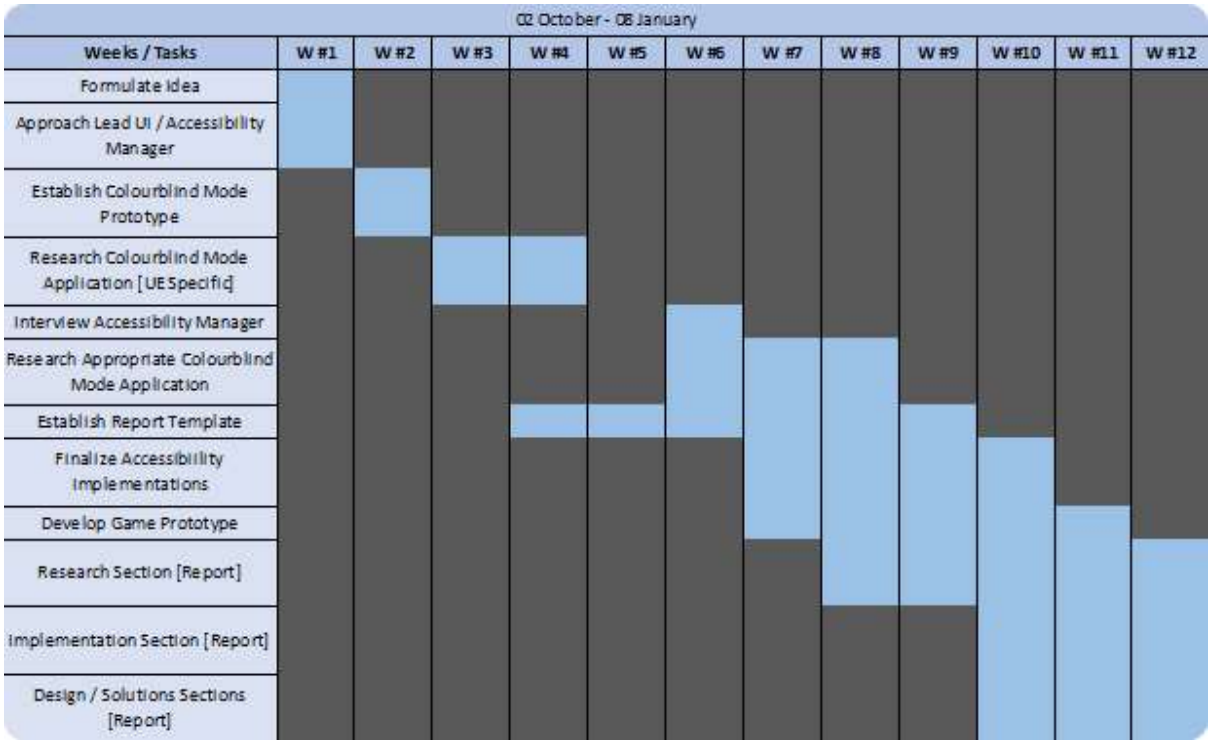
Project Management Style

This project integrated a project management style of “Agile”. This style is appropriate due to its ability to be flexible as the project structure, requirements and scope of the project can change at any given time.

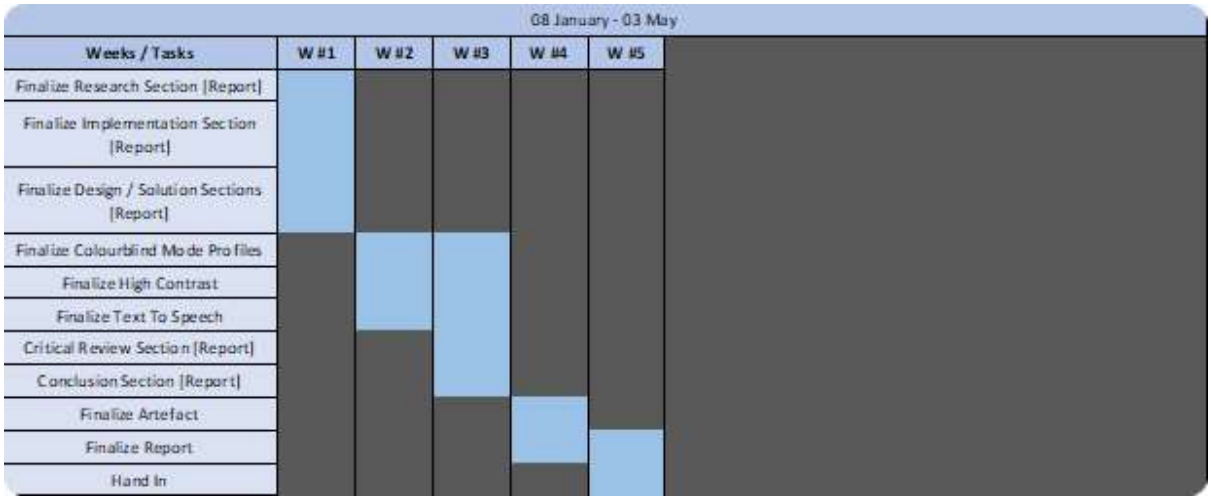
Obviously, this project was solely developed by myself, however, due to the nature of the project, I had to find a project management style that this would be best suited for. As the project developed and extra information from external sources were provided, I realized that the scope of my project had to change and be more specific to visual impairments rather than the original plan.

Schedule / Gantt Chart

October 2023 to January 2024



January 2024 to May 2024



Risk Analysis

Description	Severity	Explanation
Effectiveness	High	Due to the nature of ethics and ethical approval, I will not be able to test my project.
Health	Low	Due to on-going health conditions, I may need to change scope of the project.
General Purpose	Medium	Due to the complexity of accessibility and visual impairments, creating a general-purpose solution is not very likely.
User Defined Colour Profile	Medium	Due to complexity, this feature might not make it into the final software artefact. Not including this will reduce the effectiveness of my artefact.
Game Engine Support	High	This artefact was built in the Unity Game Engine, this means that support for Unreal Game Engine is not available, forcing developers into using Unity if they wish to incorporate the template.

Game Engine

During planning, I had to decide on a specific engine to work with. The reason for this is due to the nature of how each engine handles accessibility and their native support for these. Originally, I wanted to build this in Unreal Engine 5, as this engine already has some native colourblind accessibility implementation through their native user-interface. However, after some early development, I decided to swap to the Unity Game Engine. Reasons for this are listed below.

Unity Game Engine

The reason I wanted to use this game engine over the Unreal Game Engine is due to the nature of how projects are structured and the flexibility that exists in Unity that I, personally found, harder to achieve in Unreal Engine. All aspects of this project could be implemented in Unreal Engine, however, my familiarity with Unity is significantly less than Unreal Engine and I wanted to both test my ability and learn a new engine at the same time.

Additionally, Unreal Engine provides support for accessibility natively, and although their implementation is not particularly good, it is more than Unity has. Therefore, developing this template for Unity would be more beneficial in the long term.

Unreal Game Engine

As stated above, Unreal Engine has some accessibility options implemented by default and although this is a step in the right direction, their implementation is not particularly good. It merely provides a filter to be applied to the overall screen and this might help with certain visual impairments, the overall application is not very well received and therefore, I would need to create an additional implementation.

Knowing this, I felt it was more beneficial for myself and others if this was created in Unity. Through research, Unity has not got a great deal of accessibility options pre-included and therefore, this would be a welcome addition to the game engine for developers.

Trello Management

18th November 2023 Update

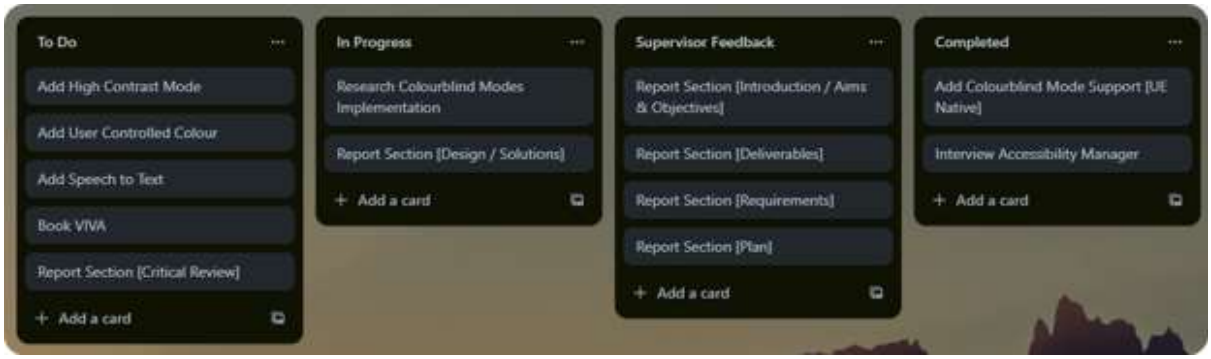


Figure 04: Trello Board Screenshot

This Trello Board was abandoned shortly after development began. GitHub was used to manage my project. Although this is not particularly good as far as a project management style, I wanted to ensure that my project was easier to manage and as I am familiar with GitHub, I decided that this would be a much better and more consistent approach. It provides a great framework for future projects, as I learnt a significant amount about version control, but it also allows me to create a customizable README which provides updates, information, and screenshots of my project. All of which I found a lot more appropriate for this project.

GitHub Management

Throughout development, I wanted to create a full-proof way of being able to transfer my project whilst keeping track of what changes were made during which period. Additionally, I wanted to have some way of explaining the current state of the project.

I decided that GitHub was by far the best use-case for this and therefore, become pushing and pulling my project.

A README has been included that will indicate features, development steps and all commits. Commit messages indicate what each push should have fixed/updated.

This also allowed for me to incorporate “issues”, which is a GitHub feature, this allowed for me to raise issues I found during development and assign commits to each issue when they were fixed.

GitHub Management: Issues

Issues Resolved

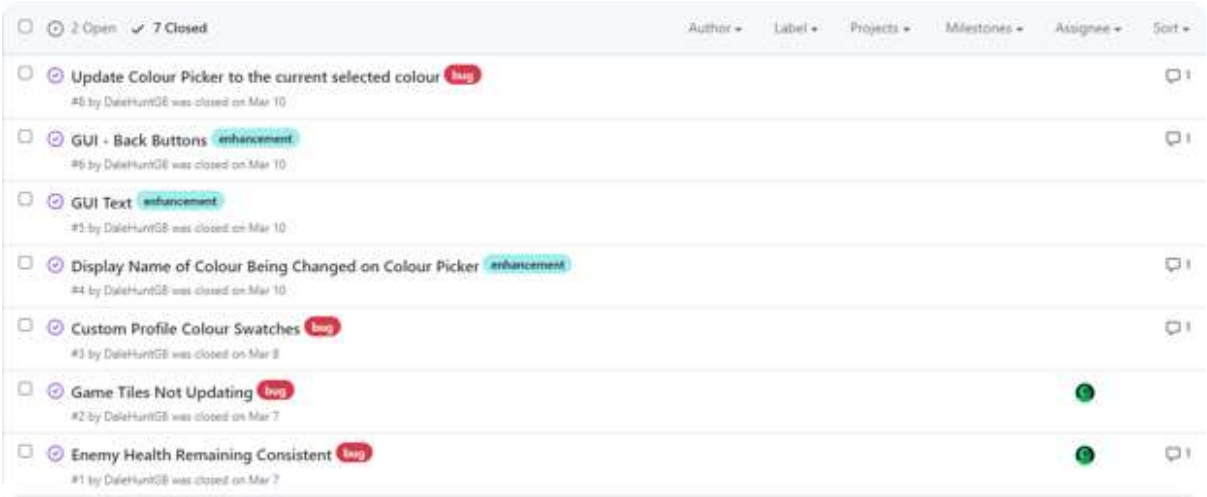


Figure 05: GitHub – Resolved Issues Screenshot

Issues Unresolved

Figure 06: GitHub – Unresolved Issues Screenshot

Issue Resolution

An issue was raised, with as much detail added as possible. Details that are included can include potential solutions, however, most included how the issue was discovered.

Figure 07: GitHub – Issue Resolution Screenshot Example

Once an issue was closed, the commit that included this fix was always included so that I could review the changes if need be.

GitHub Management: Commit History

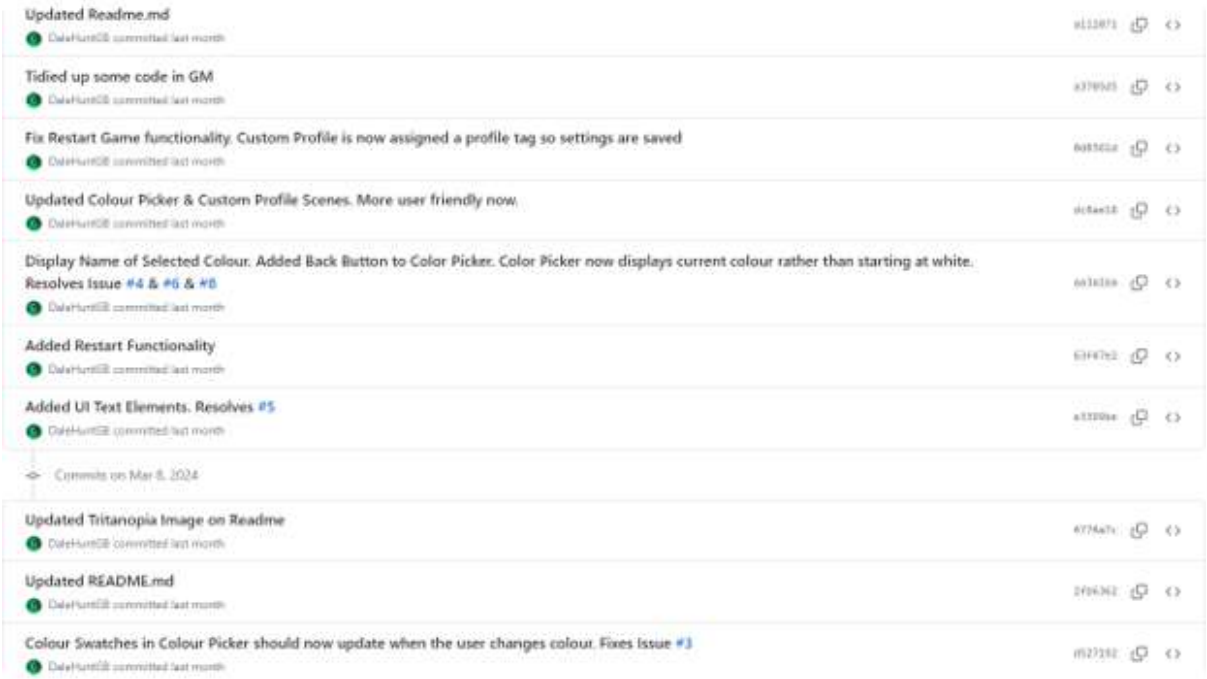


Figure 08: GitHub – Commit History Screenshot

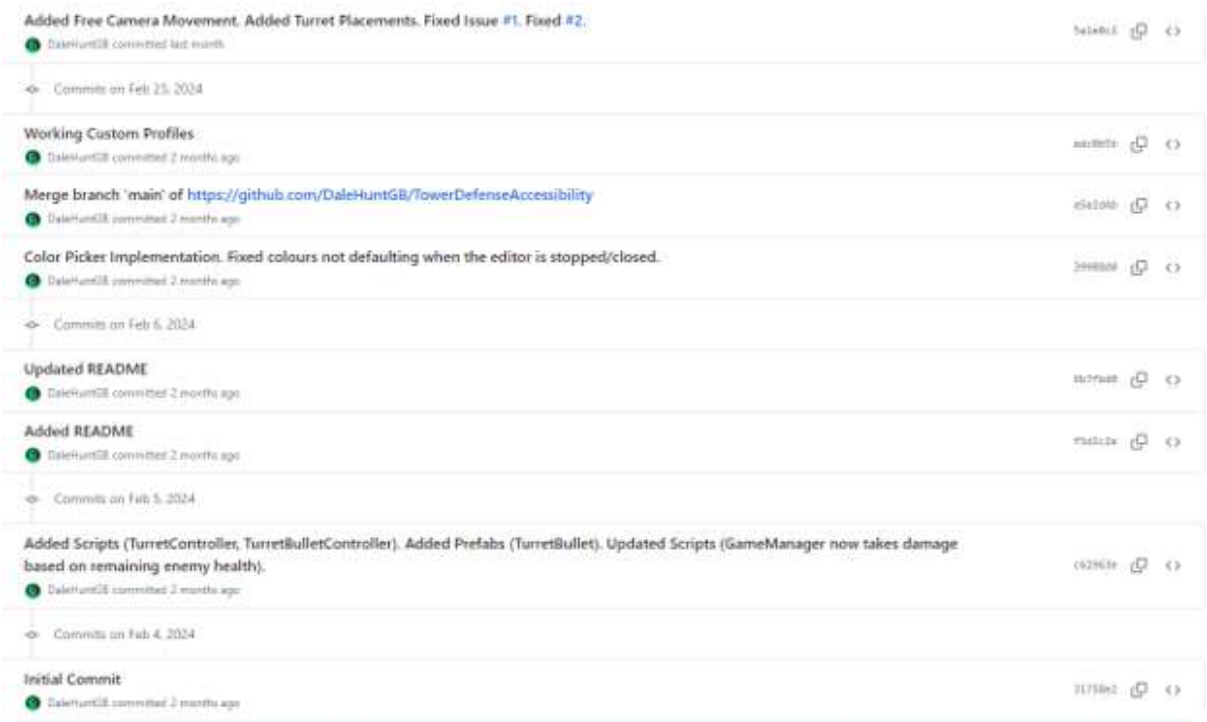


Figure 09: GitHub – Commit History Screenshot Continued

This is listed from top (being the latest) to bottom (being the first commit). Each issue that I raised, I tried to include as much detail as possible into what the issue was, all of this is public and can be viewed.

Methodology – Development

The section I will document and discuss the issues that I faced during development as well as all the research that I have done to correctly highlight the issues discussed in this paper. Additionally, this section will highlight some of the solutions that I had to these issues as well as the final approach taken. For ease, a "Useful Terminology" section has been added below to help better understand some of the discussions held later in this section.

Useful Terminology

Singleton: A singleton is a design pattern that restricts the instantiation of a class to one single instance, ensuring controlled access to the resource throughout an application.

(Singleton in C#)

Instantiate: A programmatical approach to spawning in objects that do not exist in the scene on start.

(Unity, Object.instantiate)

Enumerator / Enum: A user defined data type that is primarily used to assign names to constants.

(Watts, Enum Basics in Unity)

Scene: Scenes are primarily used in the Unity Game Engine and are responsible for containing and storing all game objects and assets for a particular scene. Complex games require multiple scenes.

(Unity, *Scenes*)

Unity Template: Templates provide preselected settings and, in this case, game objects. The goal is to streamline and speed up project creation whilst allowing the developer to discover features that might not be known to them.

(Unity, Project templates)

Switch – Case Statement: This is a selection statement, it will execute code based on the conditions that are met.

(Chand, C# switch with examples)

Software Artefact

Tower Defence game that will be singleplayer, the gameplay will be limited as the purpose of this assignment is to highlight the effects of visual impairment accessibility implementations, this game will be relatively simple and most functionality will lay in the functionality of the options implemented.

Implementation: Colourblind Modes

The player will be able to change through the different colourblind modes via the user-interface, highlighting how this is and is not an appropriate solution for individuals who suffer with colour blindness.

Implementation: High Contrast Mode

As stated above, some colourblind modes may not be suitable for players.

> *The effects of colour vision deficiency can be mild, moderate or severe and people with severe forms often think that their condition is mild and doesn't really affect them.*

(About colour blindness 2022)

In this case, a High Contrast mode can be toggled which will make it very clear for players to help identify between the multiple game objects on screen at any given point.

Implementation: User Defined Colours

Due to how colourblindness works and how everyone, even with the same visual impairment, can have varying severity levels, allowing the user to set the colour of every game object is by far the most appropriate solution and caters for more than one colourblind type at the same time. This implementation will require a custom scene that indicates the current colour of each game object as well as a custom colour picker that can be used to change the colours as necessary.

Approach #1: Colour Mapping

As stated above, my goal is to create an extendable and general approach to this solution. This approach was my first though into how that might be achievable. The idea was to assign each game object a specific colour, these colours would be set when the game would begin.

This approach, in theory, would allow the user to map any colour to a specific game object. This is achieved by reassigning (or mapping) the new colours to the game object. However, during development I discovered an issue where colours were being mapped correctly but would not apply to existing game objects, only newly created ones. Obviously, this is not ideal when considering the sgame genre and how some objects were going to remain consistent throughout the level.

The other issue that I discovered, which impacted more on performance than effectiveness, is that colour mapping only worked if the game object had the default colour applied beforehand. This caused issues when trying to swap between multiple colourblind modes, the solution was merely to reset all colours before the player applied a new colourblind mode.

Although this solution worked, it was apparent that when user defined colours were introduced, it would be a struggle to keep each game object a specific colour if they required resetting each time. This forced me into developing a more concrete and substantial approach.

Approach #2: Colour Profiles

This approach was my second and final implementation regarding this section of development. This approach fulfilled my goal of being extendable and general by allowing the developer to create colour profiles as they see fit, but it also allows for a user profile to be created, this user profile allows the user to assign colours to specific game objects and stores colours independently.

The colour profiles specifically for Protanopia, Deuteranopia, Tritanopia and High Contrast will need to be defined and hard coded by the developer, but the quantity of colour profiles and game objects is irrelevant. Developers can freely adapt the implementation as they require for their game.

Defining a Colour Profile

There is an existing class within the Accessibility Manager, this class will eventually store the defined colour for each game object that might be present in the game.

```
public class ColorProfile
{
    public Color gameWallClr;
    public Color gameTileClr;
    public Color startPointClr;
    public Color endPointClr;
    public Color enemyRouteClr;
    public Color enemyClr;
    public Color turretClr;
    public Color turretBulletClr;
    public Color highHealthClr;
    public Color lowHealthClr;
}
```

Figure 10: Colour Profile Class Definition

In this project, this class is referred to as the ColorProfile. Once this class has been created, the developer can create a new colour profile as follows:

```
public ColorProfile defaultProfile;
public ColorProfile protanopiaProfile;
public ColorProfile deuteranopiaProfile;
public ColorProfile tritanopiaProfile;
public ColorProfile highContrastProfile;
public ColorProfile greyscaleProfile;
public ColorProfile customProfile;
public ColorProfile currentProfile;
```

Figure 11: All Colour Profiles Included

As you can see, this is completely extendable. Each profile will inherit all game objects being stored by the ColorProfile class and allow for their colours to be defined as shown below:

```
defaultProfile = new ColorProfile()
{
    gameWallClr = new Color(15 / 255f, 15 / 255f, 15 / 255f, 255 / 255f),          // #0F0F0F
    startPointClr = new Color(64 / 255f, 64 / 255f, 255 / 255f, 255 / 255f),       // #4040FF
    endPointClr = new Color(128 / 255f, 64 / 255f, 255 / 255f, 255 / 255f),        // #8040FF
    enemyRouteClr = new Color(128 / 255f, 128 / 255f, 255 / 255f, 255 / 255f),     // #8080FF
    enemyClr = new Color(255 / 255f, 64 / 255f, 64 / 255f, 255 / 255f),            // #FF4040
    turretClr = new Color(255 / 255f, 128 / 255f, 64 / 255f, 255 / 255f),          // #FF8040
    turretBulletClr = new Color(0 / 255f, 170 / 255f, 187 / 255f, 255 / 255f),     // #00AABB
    highHealthClr = new Color(64 / 255f, 255 / 255f, 64 / 255f, 255 / 255f),       // #40FF40
    lowHealthClr = new Color(255 / 255f, 64 / 255f, 64 / 255f, 255 / 255f),        // #FF4040
};
```

Figure 11: Defining a Colour Profile with all Game Objects

The existing colours of the game objects are not inherently known by the Accessibility Manager so the developer will need to assign them here. This must be completed for each ColorProfile that might be implemented.

Unity Template

My goal, as stated above, was to create a general and extendable solution to accessibility issues with specific focus on visual impairments. However, as development continued, I realised that this might not be completely feasible and is more than likely the reason as to why this area of accessibility is severely underdeveloped. Due to this, I decided that I could develop a template that could be used by developers as a starting point instead.

This template will come with the following:

Game Manager

This template is currently built for tower defence games, however, in future updates I am wanting to generalise this so that it can be setup for any genre of game. The Game Manager, as in most games, will be responsible for controlling game states and user-input. In this template, the Game Manager is also responsible for applying the colour profile which is being stored by the Accessibility Manager.

Accessibility Manager

This template will include an Accessibility Manager, as seen above, this is where colour profiles are defined and stored. The Accessibility Manager is also responsible for updating the user defined profile with their colours when changed. Logically, it will be responsible for switch colour profiles when called.

Scenes

Start Menu

Simple template, it consists of two buttons – Start Game and Exit Game – this is merely an entry point for the user.

Settings Menu

This template provides an interface for the user to be able to swap between multiple colourblind modes. This template will load with a UI Manager that can handle all button presses and call appropriate functions when needed. This UI Manager is destroyed when the menu is closed.

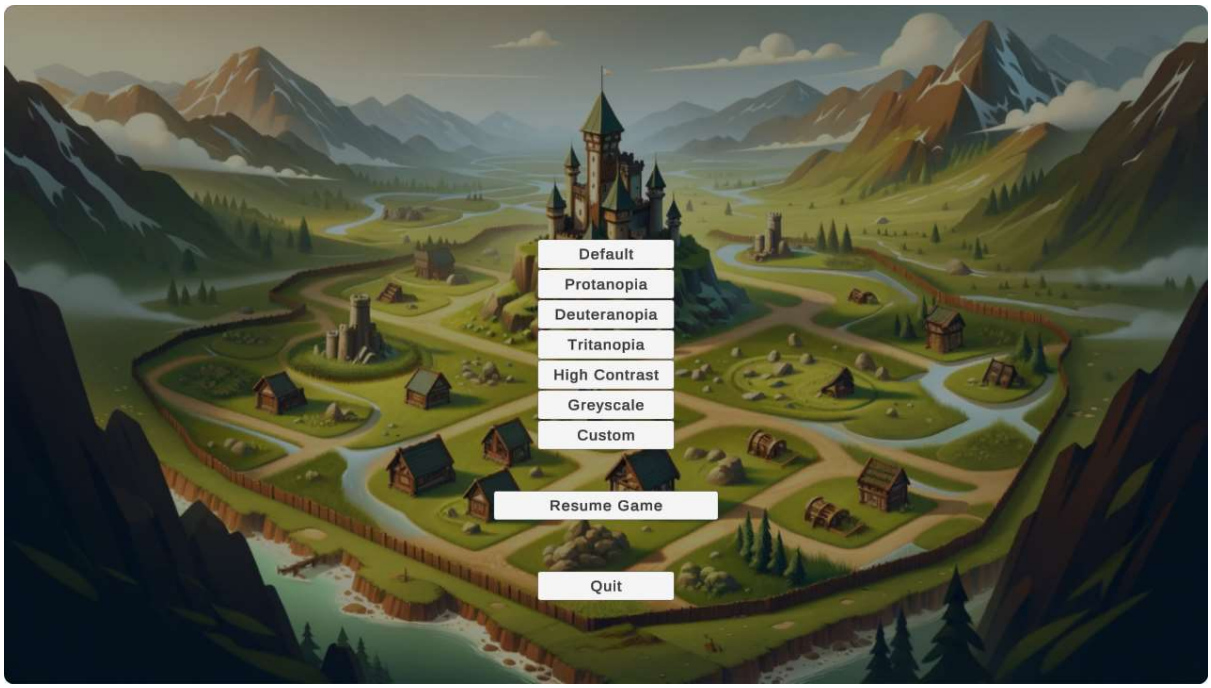


Figure 12: Unity Template – Settings Scene

This is obviously a very basic layout, but it provides a platform for developers to extend and expand on. Each button is assigned in the UI Manager, as follows:

```
void Start()
{
    defaultClrBtn.onClick.AddListener(() => OnColorProfileChanged("Default"));
    protanopiaClrBtn.onClick.AddListener(() => OnColorProfileChanged("Protanopia"));
    deuteranopiaClrBtn.onClick.AddListener(() => OnColorProfileChanged("Deuteranopia"));
    tritanopiaClrBtn.onClick.AddListener(() => OnColorProfileChanged("Tritanopia"));
    highContrastClrBtn.onClick.AddListener(() => OnColorProfileChanged("HighContrast"));
    greyscaleClrBtn.onClick.AddListener(() => OnColorProfileChanged("Greyscale"));
    customProfileBtn.onClick.AddListener(() => OnColorProfileChanged("CustomProfile"));
    resumeGameBtn.onClick.AddListener(() => GameManager.ResumeGame());
    quitGameBtn.onClick.AddListener(() => GameManager.QuitGame());
}
```

Figure 13: Unity Template – Settings Scene – Assigning Button Functionality

Once the user has pressed the corresponding button, the UI Manager will call the function assigned. In this case, the function will apply the assigned colours defined by the Accessibility Manager. This will be further explained below.

User Defined Colours Menu

If the user selects “Custom”, a new scene will be created. This scene, developed by the developer, will create a layout as shown below:

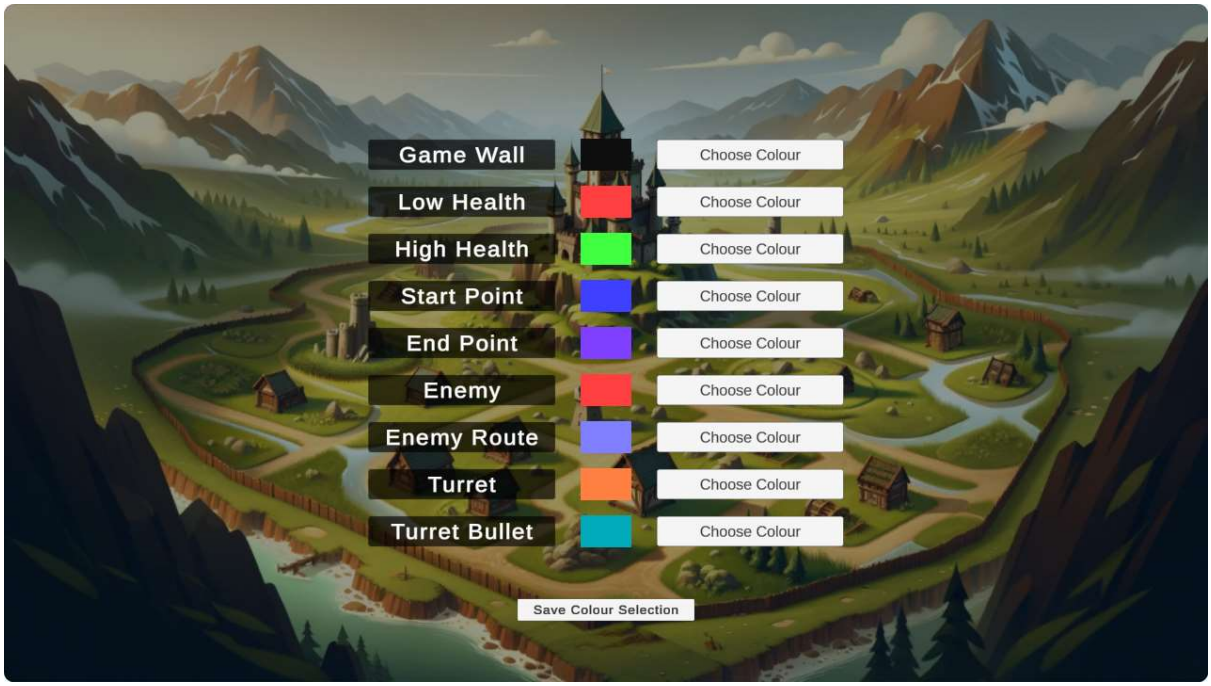


Figure 14: Unity Template – User Defined Colours Scene

The game object name and colour are generated using enumerator values. This also allows for the “Choose Colour” button to correctly select which game object should have their colour adjusted. This will be further explained below.

Colour Picker

Once the user has pressed the “Choose Colour” button, a self-built colour picker will open. This colour picker will display both the current colour as well as the new colour that the game object will become. The colour picker provides support for red, green, and blue via numerical input or sliders. An example of how this works is shown below:

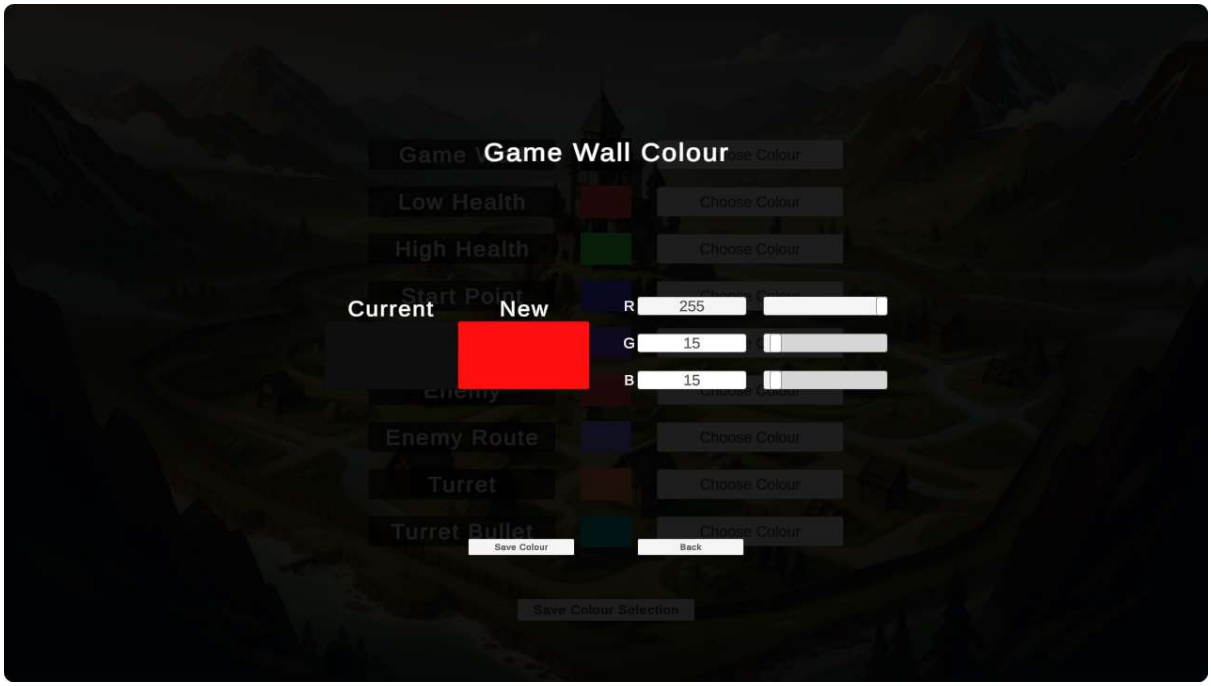


Figure 15: Custom Built Colour Picker

The title shown above the colours is created via the user of enumerator values to help the user identify which colours are going to be changed. Once the user selects “Save Colour”, the colour picker will close and the previous scene will be updated to show the updated colours, as shown below:

Figure 16: Unity Template – User Defined Colours Scene – Game Object Colour

Future Template Improvements

Currently, the template exists for tower defence games and is relatively effective for this genre. However, in future updates, I want to develop a more general template so that new genres are more easily supported.

Additionally, some usability implementations can be added for the developer, this can include generating the “User Defined Colours Menu” automatically based on the amount of game objects specified. Although this is not necessary, it provides additional support to the developer and makes implementation of this area much more likely.

Lastly, the template is relatively over-inflated with specific C# Scripts. Although these are specific to have a functioning game for demonstration, providing a cleaner starting template is less overwhelming and provides a much better foundation for the developer and ultimately, the end user.

Game Manager

As in most games, a Game Manager is required and often manages game states, variables, and specific functionality for game play. The manager is often also used for handling user input. In this template, the Game Manager also provides support for updating game tiles based on the players’ current health.

Update Game Tiles

As this artefact is based on visual impairments, I wanted to create a unique way to indicate the current health of the player without the normal on-screen text. The idea that I came up with is to adjust the colour of the game tiles in the floor of level. This solution is not only a smarter way of indicating information to the player but also provides a good insight into the effectiveness of my implementation.

This function will be called in the following conditions:

- Start Game
- Resume Game
- Player Health Reduced

Once this function is called, it will get the current colour profile from the Accessibility Manager. It will also calculate the remaining health of the player and store it as a percentage, this percentage will be used in a lerp calculation, which retrieves the low health and high health colours, and creates a colour between these two colours based on the current health percentage. Once this has been done, it will check all game objects with the appropriate tag and apply the colour.

```
public void UpdateGameTiles()
{
    var currentProfile = AccessibilityManager.GetCurrentProfile();

    float healthPercentage = currentHealth / maxHealth;

    Color tileColor = Color.Lerp(currentProfile.lowHealthClr, currentProfile.highHealthClr, healthPercentage);

    if (gameTiles == null || gameTiles.Length == 0)
    {
        gameTiles = GameObject.FindGameObjectsWithTag("GameTile");
    }

    foreach (var gameTile in gameTiles)
    {
        if (gameTile != null)
        {
            var renderer = gameTile.GetComponent<Renderer>();
            if (renderer != null)
            {
                renderer.material.color = tileColor;
            }
        }
    }
}
```

Figure 17: Game Manager – Update Game Tiles Function

This implementation was yet another reason I swapped approaches mentioned above, as my first approach could not dynamically update colours, this function would have struggled to determine which were the true low health and high health colours.

Accessibility Manager

This manager is responsible for specifically handling accessibility implementations, such as visual impairments. In this template, the Accessibility Manager will be responsible for defining, storing, and updating game object colours. The manager is instantiated as a singleton, allowing the Game Manager and UI Manager access to specific functions within this manager. These functions are explained below.

Applying Colour Profiles

This is a function that can be called by the Game Manager. It is responsible for assigning the correct colours – provided by the selected profile – to the correct game object. This function is specifically called by the Game Manager when the game is started to ensure that the correct colour profile is assigned.

The function will set the current profile to the one that is passed into the function when it is called, it will then check each game object that is defined by the class and set the colour of the game object to the colour that is defined.

In this version of the template, this will only support material colour and not textures.

```
public void ApplyColorProfile(ColorProfile profile)
{
    currentProfile = profile;
    gameWall.GetComponent<Renderer>().sharedMaterial.color = profile.gameWallClr;
    startPoint.GetComponent<Renderer>().sharedMaterial.color = profile.startPointClr;
    endPoint.GetComponent<Renderer>().sharedMaterial.color = profile.endPointClr;
    enemyRoute.GetComponent<Renderer>().sharedMaterial.color = profile.enemyRouteClr;
    enemy.GetComponent<Renderer>().sharedMaterial.color = profile.enemyClr;
    turret.GetComponent<Renderer>().sharedMaterial.color = profile.turretClr;
    turretBullet.GetComponent<Renderer>().sharedMaterial.color = profile.turretBulletClr;
}
```

Figure 18: Accessibility Manager – Apply Colour Profiles Function

Switching Colour Profiles

This is a function that is called by the UI Manager when the user is swapping between colour profiles. It applies a string value to each profile so that the function can be called via the buttons pressed in the User Defined Colours Menu. This is a simple switch – case, where the function will check a string value that is passed into the function and run the corresponding case to this switch.

```
public void SwitchColorProfile(string mode)
{
    switch (mode)
    {
        case "Default":
            ApplyColorProfile(defaultProfile);
            break;
        case "Protanopia":
            ApplyColorProfile(protanopiaProfile);
            break;
        case "Deuteranopia":
            ApplyColorProfile(deuteranopiaProfile);
            break;
        case "Tritanopia":
            ApplyColorProfile(tritanopiaProfile);
            break;
        case "HighContrast":
            ApplyColorProfile(highContrastProfile);
            break;
        case "Greyscale":
            ApplyColorProfile(greyscaleProfile);
            break;
        case "Custom":
            ApplyColorProfile(customProfile);
            break;
    }
}
```

Figure 19: Accessibility Manager – Switch Colour Profiles

The UI Manager will pass a string value into this function, this string value will then determine which case should be run. In this implementation, the case will run the above code and apply the corresponding colour profile.

Updating User Defined Colours

This functionality goes hand in hand with the Colour Picker and the use of enumerator values. Once the colours are saved via the colour picker, this function will be called and will assign the new colour that is defined by the to the corresponding value. This is done by passing in the current game object that is being adjusted and the new colour that it has been assigned.

```
public void UpdateCustomProfile(ColorProperty property, Color newColor)
{
    switch (property)
    {
        case ColorProperty.GameWall:
            customProfile.gameWallClr = newColor;
            break;
        case ColorProperty.StartPoint:
            customProfile.startPointClr = newColor;
            break;
        case ColorProperty.EndPoint:
            customProfile.endPointClr = newColor;
            break;
        case ColorProperty.EnemyRoute:
            customProfile.enemyRouteClr = newColor;
            break;
        case ColorProperty.Enemy:
            customProfile.enemyClr = newColor;
            break;
        case ColorProperty.Turret:
            customProfile.turretClr = newColor;
            break;
        case ColorProperty.TurretBullet:
            customProfile.turretBulletClr = newColor;
            break;
        case ColorProperty.HighHealth:
            customProfile.highHealthClr = newColor;
            break;
        case ColorProperty.LowHealth:
            customProfile.lowHealthClr = newColor;
            break;
    }
    ApplyColorProfile(customProfile);
}
```

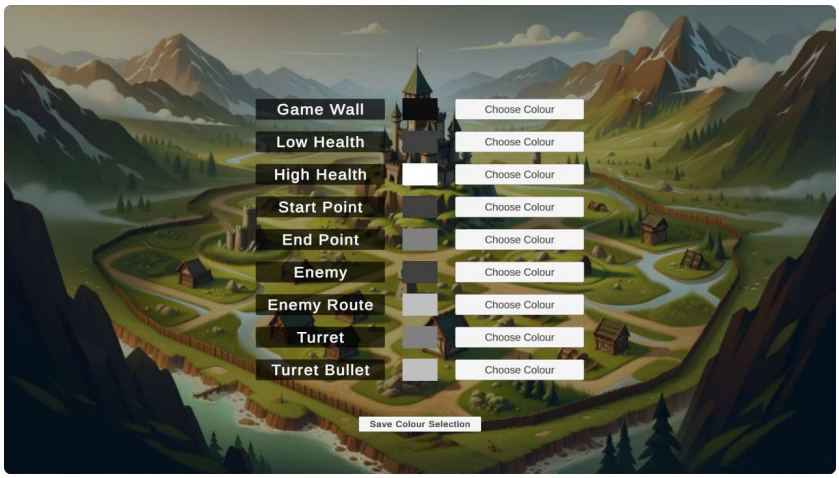
Figure 20: Accessibility Manager – Update User Defined Colours

Once this value has been updated, the profile is applied via the function stated above which will then update all game objects with their new or existing colours.

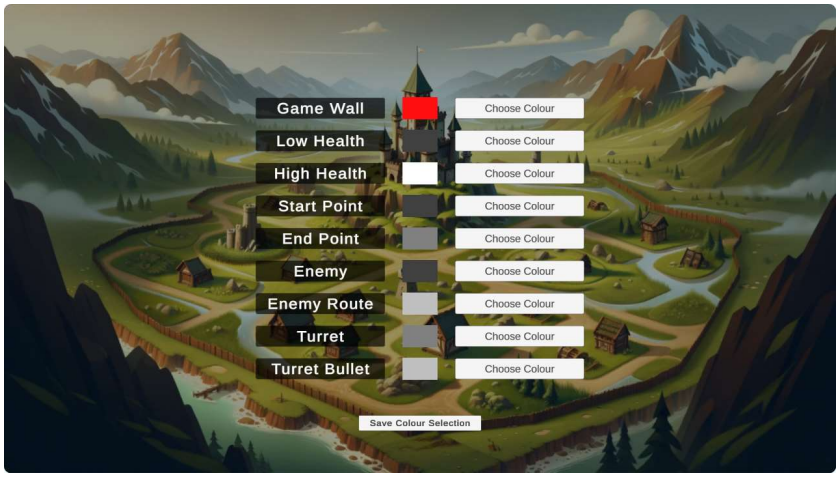
This approach, however, does create an issue where the colours that are defined by default for the user profile overwrite any existing colours until the user has manually saved them. This means that if the user swaps to the “Greyscale” profile, and then “Custom” profile, the colour blocks will indicate the greyscale variants but once the player has saved, the colours will revert to the default template except for the game object/s that were edited by the user. This bug is present on GitHub but due to time constraints, a solution has not yet been implemented.

Colour Profile Bug

1. Greyscale Selected → Custom Profile Selected.



2. Game Wall Colour Updated.



3. Colours Saved → Custom Profile Selected.

All colours, except the Game Wall, are reverted to default. Greyscale colours are not saved.

UI Manager

This is relatively common in most games and in this template, it does not do anything out of the ordinary. This manager is at the top of the list for future development as currently, each scene that requires a UI Manager has one custom designed for that specific scene. This is obviously not expandable but due to the time constraints, I was unable to develop an appropriate solution. This is something that will be further developed in future updates.

Outside of normal interactions, such as managing button presses or updating user interface texts, this manager helps communication between all parties involved in updating and storing colours of individual game objects.

Handling Colour Profiles

This is probably the most important function as it helps not only assigns the correct profiles to the correct buttons, but it will also help communicate the selected profile to both the Accessibility Manager and the Game Manager.

```
void Start()
{
    defaultClrBtn.onClick.AddListener(() => OnColorProfileChanged("Default"));
    protanopiaClrBtn.onClick.AddListener(() => OnColorProfileChanged("Protanopia"));
    deuteranopiaClrBtn.onClick.AddListener(() => OnColorProfileChanged("Deuteranopia"));
    tritanopiaClrBtn.onClick.AddListener(() => OnColorProfileChanged("Tritanopia"));
    highContrastClrBtn.onClick.AddListener(() => OnColorProfileChanged("HighContrast"));
    greyscaleClrBtn.onClick.AddListener(() => OnColorProfileChanged("Greyscale"));
    customProfileBtn.onClick.AddListener(() => OnColorProfileChanged("CustomProfile"));
    resumeGameBtn.onClick.AddListener(() => GameManager.ResumeGame());
    quitGameBtn.onClick.AddListener(() => GameManager.QuitGame());
}
```

Figure 21: UI Manager – Assign Button Functionality

This function runs when the UI Manager is instantiated into the scene, this will be specifically for the Settings Menu at this time. This assigned the correct function to each button to ensure that it calls the correct colour profile from the Accessibility Manager.

```
public void OnColorProfileChanged(string selectedProfile)
{
    AccessibilityManager.SwitchColorProfile(selectedProfile);
    PlayerPrefs.SetString("SelectedColorProfile", selectedProfile);
    PlayerPrefs.Save();
    GameManager.UpdateGameTiles();
    if (selectedProfile == "CustomProfile")
    {
        CustomProfileColorSelection();
    }
}
```

Figure 22: UI Manager – Colour Profile Change Function

Once the button is pressed, it will call this function that passes the string value assigned to the button and switches to the correct profile as stated by the Accessibility Manager. However, it also stores the selected colour profile locally using the built-in Unity save, which improves usability and ensures the player will not need to update colour profiles each time they play. This function will also automatically call the function to update all game tiles.

However, if the user has selected the “CustomProfile”, this will instead run a function that creates the User Defined Colours Menu.

User Defined Colours Menu

This scene is responsible for indicating the current set of colours applied to all existing game objects within the game. The text boxes are populated and updated by the developer, although, this will eventually be done programmatically. The developer will need to add indicators to display colours for each game object, the indicators can be of any shape, size and colour as the colours will be updated upon scene creation.

This scene is pre-configured for simplicity for developers and will look as follows, by default:

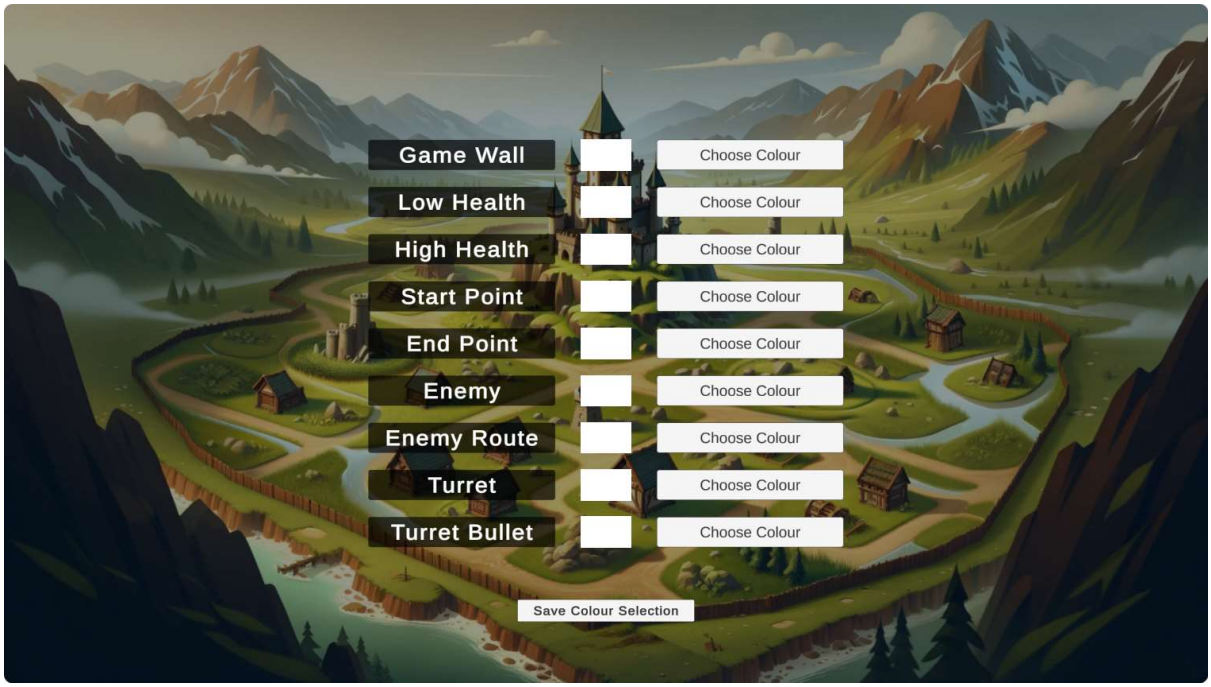


Figure 23: User Defined Colour Scene – Template

Generating and Updating Colour Indicators

The updating of colour indicators is relatively straight-forward, the developer will need to define each game object or image in the Custom Colour Manager. This manager is responsible for this scene and handles all tasks that are required by this. In future updates, this will hopefully move into the Accessibility Manager for simplicity.

However, the developer can create each game object or image as follows:

```
public Image gameWallClr;
public Image startPointClr;
public Image endPointClr;
public Image highHealthClr;
public Image lowHealthClr;
public Image turretClr;
public Image turretBulletClr;
public Image enemyRouteClr;
public Image enemyClr;
```

Figure 24: User Defined Colour Scene – Updating Colour Indicator for Game Objects

Once these are defined, the developer can drag the elements into the Unity Inspector. This process, although tedious, prevents the need for checking specific game object or image names and tags. Overall, this process is slightly more performance friendly. Once this has been done, and the scene is loaded, the function below will run.

```
void Start()
{
    // Assign Colours to specific colour indicators
    gameWallClr.color = AccessibilityManager.GetCurrentProfile().gameWallClr;
    startPointClr.color = AccessibilityManager.GetCurrentProfile().startPointClr;
    endPointClr.color = AccessibilityManager.GetCurrentProfile().endPointClr;
    highHealthClr.color = AccessibilityManager.GetCurrentProfile().highHealthClr;
    lowHealthClr.color = AccessibilityManager.GetCurrentProfile().lowHealthClr;
    turretClr.color = AccessibilityManager.GetCurrentProfile().turretClr;
    turretBulletClr.color = AccessibilityManager.GetCurrentProfile().turretBulletClr;
    enemyRouteClr.color = AccessibilityManager.GetCurrentProfile().enemyRouteClr;
    enemyClr.color = AccessibilityManager.GetCurrentProfile().enemyClr;
    // Assign functionality to specific colour picker buttons
    gameWallClrBtn.onClick.AddListener(() => ToggleColorPicker(ColorProperty.GameWall));
    startPointClrBtn.onClick.AddListener(() => ToggleColorPicker(ColorProperty.StartPoint));
    endPointClrBtn.onClick.AddListener(() => ToggleColorPicker(ColorProperty.EndPoint));
    highHealthClrBtn.onClick.AddListener(() => ToggleColorPicker(ColorProperty.HighHealth));
    lowHealthClrBtn.onClick.AddListener(() => ToggleColorPicker(ColorProperty.LowHealth));
    turretClrBtn.onClick.AddListener(() => ToggleColorPicker(ColorProperty.Turret));
    turretBulletClrBtn.onClick.AddListener(() => ToggleColorPicker(ColorProperty.TurretBullet));
    enemyRouteClrBtn.onClick.AddListener(() => ToggleColorPicker(ColorProperty.EnemyRoute));
    enemyClrBtn.onClick.AddListener(() => ToggleColorPicker(ColorProperty.Enemy));
    // Update colour selection.
    saveColorSelection.onClick.AddListener(() => SaveColorSelection());
}
```

Figure 25: User Defined Colour Scene – Applying Button Functionality

This function will retrieve the current profile from the Accessibility Manager and receive each of the corresponding colour values from these game objects. This colour will be applied to the assigned colour indicator. As this happens each time this scene is loaded, the user will always see the most up-to-date colours for every game object present.

Additionally, this function also assigns the correct colour property to each colour picker button. This is required so that when the user decides to choose a colour of a specific game object, when they save, the information that was passed when the button was pressed will update the specific game object with the new colour. Each game object will require a colour picker button, as this is the only way in the current project to ensure that specific colours can be assigned to specific game objects.

Dynamically Update Colour Indicators

During my interview with the Games Accessibility Manager, he spoke about the importance of indicating the current colours of the game objects to the user. This is useful for two reasons, the first is that they can easily and immediately see if this colour chosen is sufficient in providing more information without needing to go back into the game, and the second reason, is usability. Updating colour indicators in real-time will help users determine which colours they have and have not changed.

```
public void UpdateColourBlocks(ColorProperty currentColorProperty)
{
    switch (currentColorProperty)
    {
        case ColorProperty.GameWall:
            gameWallClr.color = AccessibilityManager.GetCurrentProfile().gameWallClr;
            break;
        case ColorProperty.StartPoint:
            startPointClr.color = AccessibilityManager.GetCurrentProfile().startPointClr;
            break;
        case ColorProperty.EndPoint:
            endPointClr.color = AccessibilityManager.GetCurrentProfile().endPointClr;
            break;
        case ColorProperty.HighHealth:
            highHealthClr.color = AccessibilityManager.GetCurrentProfile().highHealthClr;
            break;
        case ColorProperty.LowHealth:
            lowHealthClr.color = AccessibilityManager.GetCurrentProfile().lowHealthClr;
            break;
        case ColorProperty.Turret:
            turretClr.color = AccessibilityManager.GetCurrentProfile().turretClr;
            break;
        case ColorProperty.TurretBullet:
            turretBulletClr.color = AccessibilityManager.GetCurrentProfile().turretBulletClr;
            break;
        case ColorProperty.EnemyRoute:
            enemyRouteClr.color = AccessibilityManager.GetCurrentProfile().enemyRouteClr;
            break;
        case ColorProperty.Enemy:
            enemyClr.color = AccessibilityManager.GetCurrentProfile().enemyClr;
            break;
    }
}
```

Figure 26: User Defined Colour Scene – Updating Colour Indicator for Game Objects Dynamically

This function is called by the Colour Picker whenever a colour is saved. This will merely take the new colour that is assigned by the user via the colour picker and update the colour indicator that is associated with the enumerator. To ensure that colours are updated correctly, the colour picker will first save the colours to the user defined profile and this function will merely retrieve information from that specific profile.

Colour Picker

This colour picker is custom built by myself, although there are plenty of colour pickers present on the Unity Asset Store, I wanted to create something unique and more specific to my project. The layout of the colour picker is very similar to that of the normal colour picker; however, the current version only supports red, green, and blue values via input or sliders. In future updates, I would like to add some support for hex colours and possibly hue, saturation, and lightness. Although this is not something particularly related to the subject of this artefact, it would be nice to be able to complete and possibly release as an independent asset.

The colour picker is created when the user selected “Choose Colour” from the User Defined Colour Menu. By default, the colour picker is displayed as such:

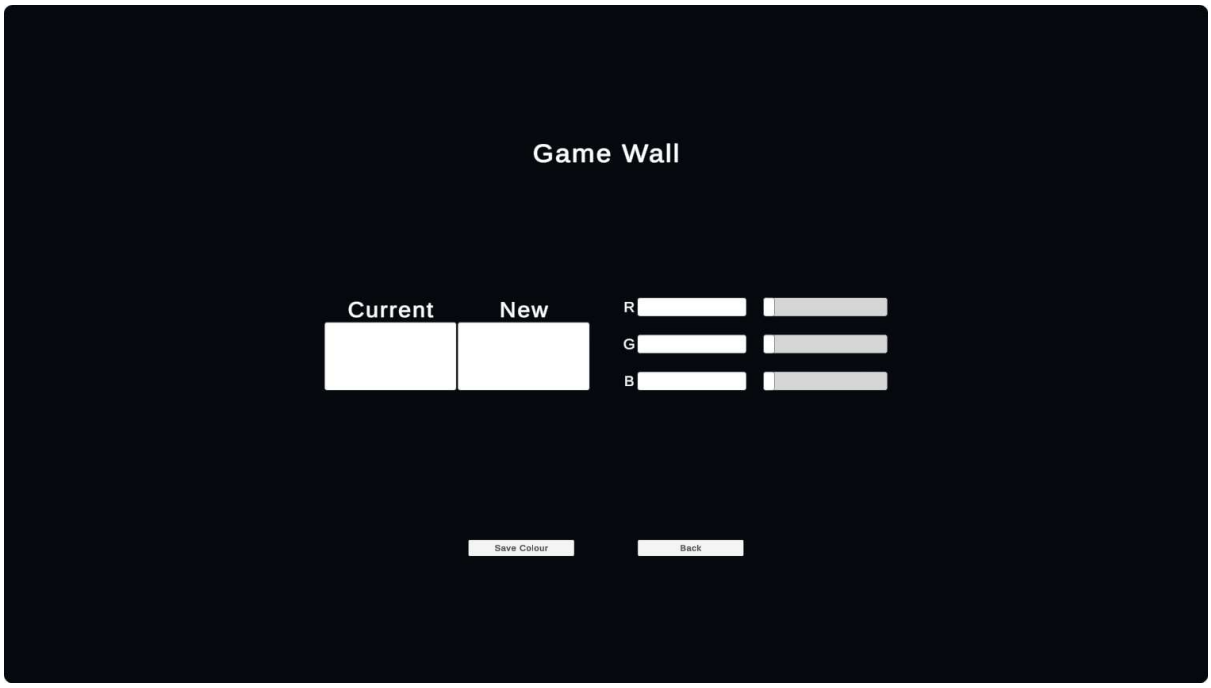


Figure 27: Colour Picker Scene – Template

However, the colours and values are populated based on the information that is passed into the button. In this case, we are passing a colour property which is associated with an enumerator that is attached to a specific game object.

Initialize Values

```
public void InitializeValues(ColorProperty colorProperty)
{
    Color initialColor = new Color();
    if (colorProperty == ColorProperty.GameWall)
    {
        initialColor = CustomColorManager.gameWallClr.color;
    }
    else if (colorProperty == ColorProperty.StartPoint)
    {
        initialColor = CustomColorManager.startPointClr.color;
    }
    else if (colorProperty == ColorProperty.EndPoint)
    {
        initialColor = CustomColorManager.endPointClr.color;
    }
    else if (colorProperty == ColorProperty.EnemyRoute)
    {
        initialColor = CustomColorManager.enemyRouteClr.color;
    }
    else if (colorProperty == ColorProperty.Enemy)
    {
        initialColor = CustomColorManager.enemyClr.color;
    }
    else if (colorProperty == ColorProperty.Turret)
    {
        initialColor = CustomColorManager.turretClr.color;
    }
    else if (colorProperty == ColorProperty.TurretBullet)
    {
        initialColor = CustomColorManager.turretBulletClr.color;
    }
    else if (colorProperty == ColorProperty.HighHealth)
    {
        initialColor = CustomColorManager.highHealthClr.color;
    }
    else if (colorProperty == ColorProperty.LowHealth)
    {
        initialColor = CustomColorManager.lowHealthClr.color;
    }

    float redValue = initialColor.r * 255;
    float greenValue = initialColor.g * 255;
    float blueValue = initialColor.b * 255;

    redInput.text = redValue.ToString();
    greenInput.text = greenValue.ToString();
    blueInput.text = blueValue.ToString();

    redSlider.value = initialColor.r;
    greenSlider.value = initialColor.g;
    blueSlider.value = initialColor.b;

    currentColorBlock.color = initialColor;
}
```

Figure 28: Colour Picker Scene – Initialize Colours Based on Selected Game Object

Once the button is pressed, this function is called. To ensure that there are no errors during loading, a coroutine is established in the Custom Colour Manager, this coroutine will only run this code once the Colour Picker scene has completed loading. A coroutine is a function that can be run when a specific condition is met, in this example, it will wait for the colour picker to finish loading before attempting to complete the function.

This function will merely run through an if-else statement and check which colour property was passed in via the button. Once the information matches any of the if statements, the colour of the "Current" block will be updated accordingly. For usability, I added the feature to update both red, green, and blue input and slider values so that the user could start editing from the existing colour.

Displaying Change in Colours

For usability, I have added the feature that the colour picker will display the existing colour as well as the previous colour. This helps the user visually determine if the colour is different enough but also helps remind them of the starting colour in case they wish to return. This functionality is handled in two separate functions, one specifically for input and the other for sliders.

```
private void UpdateColorsViaInput()
{
    if (updatedColorBlock != null)
    {
        float redValue = 0;
        float greenValue = 0;
        float blueValue = 0;

        if (float.TryParse(redInput.text, out redValue))
        {
            redValue = Mathf.Clamp(redValue, 0, 255) / 255;
        }

        if (float.TryParse(greenInput.text, out greenValue))
        {
            greenValue = Mathf.Clamp(greenValue, 0, 255) / 255;
        }

        if (float.TryParse(blueInput.text, out blueValue))
        {
            blueValue = Mathf.Clamp(blueValue, 0, 255) / 255;
        }

        updatedColorBlock.color = new Color((float)redValue, (float)greenValue, (float)blueValue);

        redSlider.value = redValue;
        greenSlider.value = greenValue;
        blueSlider.value = blueValue;
    }
}

3 references
private void UpdateColorsViaSliders()
{
    if (updatedColorBlock != null)
    {
        updatedColorBlock.color = new Color(redSlider.value, greenSlider.value, blueSlider.value);
        redInput.text = Mathf.Floor((redSlider.value * 255)).ToString();
        greenInput.text = Mathf.Floor((greenSlider.value * 255)).ToString();
        blueInput.text = Mathf.Floor((blueSlider.value * 255)).ToString();
    }
}
```

Figure 29: Colour Picker Scene – Update Colour Display on Colour Picker (Sliders / Input)

The input is adjusted by the starting function in the image above, it will merely create a new colour based on the `redValue`, `greenValue` and `blueValue`. To ensure always accurate colours are picked, each input is clamped between 0 – 255 which is the maximum colour value for red, green, and blue. The addition of `float()` merely helps cast the correct variable type to the value supplied so that errors can be avoided, this functionality will ensure each value returned is always going to be a float (which is a variable type specifically given to numbers).

The sliders are adjusted by the ending function in the image above, as sliders return a value, all I needed to do was create a colour based on these values, like the above function. For usability, this function will also update the text values in the input boxes for appropriate colour.

Testing / Evaluation

Due to the nature of my project and the inability to get ethical approval for testing, I am unable to test the final artefact developed. Additionally, if ethical approval was received, finding a wide range of individuals that suffer with visual impairments, in a wide range of severities would be near to impossible.

Evaluating the final artefact is therefore difficult but It is safe to assume, based on interviews with Game Accessibility Managers, that user defined colours would be the most appropriate to help all individuals that may have a visual impairment, however, this can unfortunately not be confirmed through testing.

Critical Review

Good Implementation

Colour Profile

I believe that this is implemented as well as it could be for this project. The system is extendable and easily adaptable to match the needs of the genre of the game being created as well as the specific requirements that the developer might need.

Additionally, this approach does not require any specific knowledge outside of your general C# practices, this means that it is easy to implement by both less and more experienced developers.

Game Manager and Accessibility Manager

The use of the Accessibility Manager and Game Manager promotes “modularity”. Creating each manager to handle very specific tasks but allow them to communicate to one another is a fundamental requirement to this type of project. It allows developers to choose which managers they wish to include and therefore, ultimately providing flexibility to each project that might incorporate this template.

General

Although accessibility is very much a project specific issue, I believe that this project has been as general as it could be. The systems implemented are modular and can be easily adapted to multiple different projects and game genres. The limitation of this generalisation is down to the fact that the developer will need to use this as a starting template, however, this is no different to starting any other Unity project.

Improvements

Effectiveness

Unfortunately, although the project is a working artefact both physically and theoretically, due to the nature of ethics and requiring ethics approval to do testing, I am unable to reliably substantiate whether this project is effective for those suffering from visual impairments.

More importantly to note, if ethics approval could have been acquired, the test group required would have needed to be a substantial subset – 30 or more – and ranging in both colourblindness types and severity levels.

UI Manager

Although I have created the other managers, mentioned above, as modular, and specific, I never had time to incorporate this with the UI Manager. The reason for this is due to the additional complexity of this. Currently, a new UI Manager is created to manage each scene and although this does work, it is not the best practice and will ultimately lead to more work for future developers.

Material Support

This template can only support the changing of material colours for game objects that are instantiated in the world. This is obviously not ideal when developers will want to begin including textures and more stylized content. The reason texture support was not added in this prototype is due to the nature of how complex it is and the time constraints of this project.

Texture implementation might be considered in future updates; however, this will require a complete refactoring of the code-base and possible work with external artists to ensure that textures for a specific game can be correctly produced.

Future Updates

Texture Support

As stated above, material support is currently the only aspect being supported. For this prototype to be effective, in future updates, I would like to add support for swapping textures of game objects based on the colour profile selected. This is obviously more complicated when considering user defined colours, therefore, this might be something that needs refactoring or removing.

UI Manager

As stated above, the UI Manager was not implemented as well as it should have been. This is something that I will be updating in the future, the ideal situation would be to assign all user-interface related tasks to this manager. Therefore, the game would consist of three managers in total, completing a wide range of tasks but remaining modular so that developers could swap and change as they require for their specific game.

Conclusion

In conclusion this project has taught me a substantial amount about Unity, C# and more importantly, accessibility issues with regards to visual impairment. Although this artefact is far from perfect, I am more than happy with my approach to solve an ever-growing and complex problem that is posed to all developers.

This artefact has increased my overall coding ability and to some degree, research, and willingness to learn about more complicated subjects. Discussing these areas with individuals in the field was by far the most enjoyable part of this project, from a personal view, as you are working with people that not only find this as interesting as you do but it provides an in-depth view into how the industry is tackling this same issue but with a much large team and development investment.

Overall, this project was incredibly enjoyable to undertake over the course of this year and generally, I am incredibly proud of the work that I have produced, both in this report and in the software artefact.

References

About colour blindness (2022) Colour Blind Awareness. Available at: https://www.colourblindawareness.org/colour-blindness/#:~:text=The%20effects%20of%20colour%20vision,doesn't%20really%20affect%20them. (Accessed: 10 May 2024).

AccessibleGames (2023). Color Vision Test. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Color_vision_test#Pseudoisochromatic_plates [Accessed 11 Nov. 2023].

Chand, M. (no date) C# switch with examples, C# Corner. Available at: https://www.c-sharpcorner.com/article/c-sharp-switch-statement/#:~:text=C%23%20switch%20case%20statement%20is,more%20than%20a%20few%20options. (Accessed: 10 May 2024).

CNN, A.E. (2020). A blind video gamer got emotional after seeing The Last of Us Part II's extensive accessibility options for players with disabilities. [online] CNN. Available at: https://edition.cnn.com/2020/06/19/us/the-last-of-us-part-ii-accessibility-options-blind-gamer-trnd/index.html.

Colour Blind Awareness (2022). About Colour Blindness. [online] Colour Blind Awareness. Available at: https://www.colourblindawareness.org/colour-blindness/.

Creative Assembly (2019). Designing for Colour Blindness in Games | Creative Assembly and BAFTA Games. [online] www.youtube.com. Available at: https://www.youtube.com/watch?v=4Vp9hxzW4yw.

Family Gaming Database (2022). Bloons TD 6 Series Accessibility Report - Amazon Fire, Android, Mac, PC, PS4, Switch, Xbox One and iOS - Family Gaming Database. [online] www.familygamingdatabase.com. Available at: https://www.familygamingdatabase.com/en-gb/accessibility/Bloons+TD+6 [Accessed 18 Nov. 2023].

Games, A. (n.d.). Distinguish This from That. [online] Accessible Games. Available at: https://accessible.games/accessible-player-experiences/access-patterns/distinguish-this-from-that/#:~:text=When%20a%20player%20turns%20on [Accessed 11 Nov. 2023].

kevinasg (2022a). Xbox Accessibility Guideline 112 - Microsoft Game Dev. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/gaming/accessibility/xbox-accessibility-guidelines/112 [Accessed 24 Nov. 2023].

kevinasg (2022b). Xbox Accessibility Guideline 114 - Microsoft Game Dev. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/gaming/accessibility/xbox-accessibility-guidelines/114 [Accessed 24 Nov. 2023].

kevinasg (2023). Xbox Accessibility Guideline 113 - Microsoft Game Dev. [online] learn.microsoft.com. Available at: https://learn.microsoft.com/en-us/gaming/accessibility/xbox-accessibility-guidelines/113 [Accessed 24 Nov. 2023].

Singleton in C# (no date) Refactoring.Guru. Available at: https://refactoring.guru/design-patterns/singleton/csharp/example#:~:text=Singleton%20is%20a%20creational%20design,and%20cons%20as%20global%20variables. (Accessed: 09 May 2024).

Technologies, U. (no date b) Project templates, Unity. Available at: https://docs.unity3d.com/2020.1/Documentation/Manual/ProjectTemplates.html#:~:text=Templates%20speed%20up%20the%20process,Shader%20Graph%2C%20and%20Post%20Processing. (Accessed: 09 May 2024).

Technologies, U. (no date a) Object.instantiate, Unity. Available at: https://docs.unity3d.com/ScriptReference/Object.Instantiate.html (Accessed: 09 May 2024).

Technologies, U. (no date) Scenes, Unity. Available at: https://docs.unity3d.com/Manual/CreatingScenes.html#:~:text=Scenes%20are%20where%20you%20work,%2C%20obstacles%2C%20decorations%2C%20and%20UI (Accessed: 09 May 2024). s

PlayStation (n.d.). God of War Ragnarök Accessibility. [online] PlayStation. Available at: https://www.playstation.com/en-gb/games/god-of-war-ragnarok/accessibility/.

Watts, J. (2021) Enum Basics in Unity, Medium. Available at: https://medium.com/@joshwatts592/enum-basics-in-unity-7e6e2742bd0 (Accessed: 09 May 2024).

Webster, A. (2020). The Last of Us Part II isn’t just Naughty Dog’s most ambitious game — it’s the most accessible, too. [online] The Verge. Available at: https://www.theverge.com/21274923/the-last-of-us-part-2-accessibility-features-naughty-dog-interview-ps4.

Wikipedia Contributors (2019). Ishihara Test. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Ishihara_Test.

Appendix

Colourblind Mode Statistics

Protanopia

Affects: Male.

Percentage: ~25%.

Cause: Reduced sensitivity to red light.

Most Common Colours Effected:

- Black and Red.
- Browns, Greens, Reds, and Oranges.
- Blues, Purples, and Dark Pink.

(Creative Assembly, 2019)

Deuteranopia

Affects: Male.

Percentage: ~75%.

Cause: Reduced sensitivity to green light.

Most Common Colours Effected:

- Red, Greens and Browns.
- Blue-Green and Greys.
- Light Greens and Yellows.
- Reds, Oranges, and Yellows.

(Creative Assembly, 2019)

Tritanopia

Affects: Male / Female.

Percentage: 1 / 10 000.

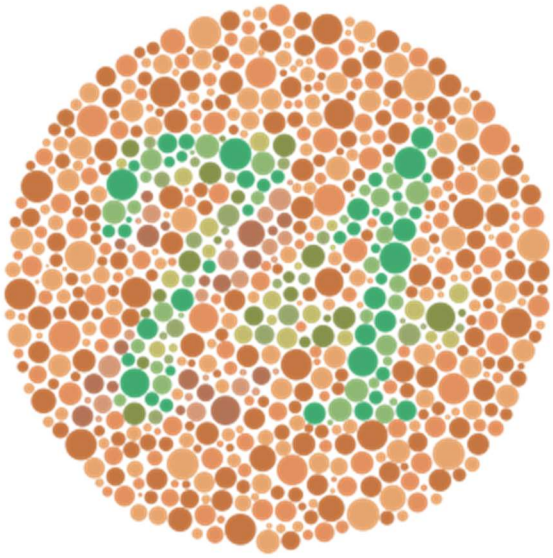
Cause: Reduced sensitivity to blue and yellow light.

Most Common Colours Effected:

- Blues, Greens, and Greys.
- Dark Purples and Black.
- Oranges and Red.

(Creative Assembly, 2019)

Ishihara Colour Test



Pseudoisochromatic Plate

A figure is embedding in the plate as several spots surrounded by spots of a slightly different colour. They are primarily used a screening tool due to how fast and simple they are to create.

(Wikipedia, 2023).

VIVA Feedback

Overall, VIVA feedback was positive. Both readers were happy with my understanding of the problem area and my proposed solution. The report was in good standing, however, could be strengthened by more technical information about my approach, and how I plan on testing the prototype. Additionally, discuss practical and ethical constraints to being able to test this artefact.

Supervisor Meetings / Logs

Meetings were not recorded, however, for the first 8 weeks of the project, I met with Robin Heath weekly to discuss progress and vision. He provided fantastic insight and questions that really helped me to decide on a particular area of accessibility. Post VIVA feedback, we met a couple more times for me to show him the current state of the project. My impression was that he was happy with the artefact, however, prompted me to delve further into generalisation rather than gameplay, which is much better suited to this style of project.