08351 Development Project – Technical Documentation

Group 2 – 'Scenarios' Unity Application

Introduction

This report aims to explain and justify the design decisions used in the implementation of the Unity aspect of the Scenarios project. Additionally, it will outline the functionality of each game object and any attached programming logic.

A brief breakdown of the vocabulary used within Unity is provided:

Component – a modular piece of code or container that provides specific functionality.

Scripts – basically a component that contains programmed logic.

Game Object – an entity within Unity that holds components.

Scene – a scene is a data structure that holds game objects.

The high-level game object and class diagrams can be found in **(Appendix B), (Appendix C)** and **(Appendix D)**.

The 'ARC' Program

The arc Unity application allows the user to draw on where the fire and smoke will show up in the Scenarios application. It uses the same video playing (visual) code as the Scenarios application.

Data Structure

The decisions made in the Scenario Builder tool provide the data for the Unity application to display.

The process as shown in **(Table 1)**.

Step 1	➔	Step 2	➔	Step 3
Scenario Builder (Software Dev side)	JSON data is generated	'API' Data .JSON format	JSON data is passed to Unity	Unity Application (Games Dev side)

Table 1: Shows the high-level data flow of choices from the Scenario Builider tool to the Unity application.

A direct copy of the data template itself can be found in **(Appendix A)**. The data includes file names and file paths for the video and audio resources. The data, most importantly, includes the list of choices constructed in the scenario builder as well as any appropriate text that needs to be displays for those choices. It also specifies the file path to deposit the output from a scenario play-through in the Unity application. Additional variables within the data are

present to support tweaking scene and emergency lighting brightness, audio volume and particle effect presence.

Handling the API Data in Unity

The API data (**Appendix A)** is the format of the data generated by the Scenario Builder tool and imported into the Unity application. The Scenarios class converts this data into a meaningful data with the aid the JSONParser class and stores that data within the Scenario class. A list of these Scenario classes are held within the Scenarios class for use with the Unity application.

Input Control

The input control is handled via the CameraControler script attached to the Main Camera. This script checks for relevant key presses (see User Documentation for hotkeys) and applies a state to change relevant objects such as the fire extinguisher or emergency lighting. For the axial input (horizontal and vertical input axis), this is translated to rotate the camera. This allows for the application to work with keyboard input as well as HMD movement. The OculusCameraPositionLock script helps to maintain correct offset to and from the main camera.

Choice Selection

Choice selection is handled by the ChoiceTrigger script attached to the ChoicePrefab object, which is a prefab of the box displayed on screen to represent the selectable area to gaze at. This prefab is instantiated by the Choices script attached to the Choice Canvas object. This script generates the necessary amount of ChoicePrefab objects and places them evenly around an origin point located at the centre of the scene.

User Interface

The UI is handled by separate canvas objects for each UI element. Each Canvas has slightly different behaviour due to the nature of their functionality (this does not include particle canvases).

The Countdown Canvas is created by the Choice script and updated by the Choice Trigger object, updating when the crosshair interacts with a choice box. This canvas shows the countdown to lock in the selection based on gaze.

The Scenario Text object and Scenario Choice Text Canvas is initially set by the Scenarios class. These objects represent the current text for the scenario to allow additional information to be displayed on screen as part of setting a story for the user.

The Choice Text Canvas is attached to the ChoicePrefab object. This canvas displays what decision this choice box represents.

The Audio Visual Canvas object, contains the AudioVisual script. This script contains the file path of each audio source within the scene. The data for the audio is loaded via co-routines from the given file paths and once loaded, they begin to play. This script also handles loading the video into the video player object and playing the video.

The Timer and Score Canvases display the time since simulation start and the current score. This is managed by the Metrics Canvas object.

Transitions

There are fade-in and fade-out transitions between video segments. These are handled within the OutTransition and InTransition scripts. OutTransition is controlled by the ChoiceTrigger script, where if the user maintains gaze upon on the decision box for long enough, the ChoiceTrigger script will begin the transition. Once the out transition is complete, the OutTransition class will enable the InTransition functionality and disable itself. The OutTransition script can also be enabled by the AudioVisual in certain cases.

Particle Effects

The Fire Canvas and Smoke Canvas objects hold the Fire and Smoke scripts respectively. These scripts hold a reference to their respective FirePrefab and SmokePrefab. The Fire and Smoke scripts handle the arc, angle and positioning of the fire prefabs. These are contained in list collections.

The fire extinguisher effects are held within the Fire Extinguisher Visuals game object as children. The child objects contain the particles themselves as well as the WaterHoseParticles scripts, which holds a reference to the particle system. In addition to this, the script also applies physics to a transparent sphere around the user's position for the particles to interact with, adding to the realism of the particle effects.

Lighting

The emergency lighting is handled within EmergencyLight script on the Emergency Light object. This script changes the intensity of the light between 0 and a value defined within the imported dataset.

The brightness is handled within the Scenarios script when the data is imported from the JSON.

Output

The Unity application tracks certain metrics within the VR experience and outputs these metrics to file. The metrics currently tracked are decisions selected (tracked by ID), correct choices made (tracked by score) and time taken to make a decision, tracked by timestamp. An example can be found in **(Appendix E)**.
These are output to a file at a location specified by the imported JSON data.

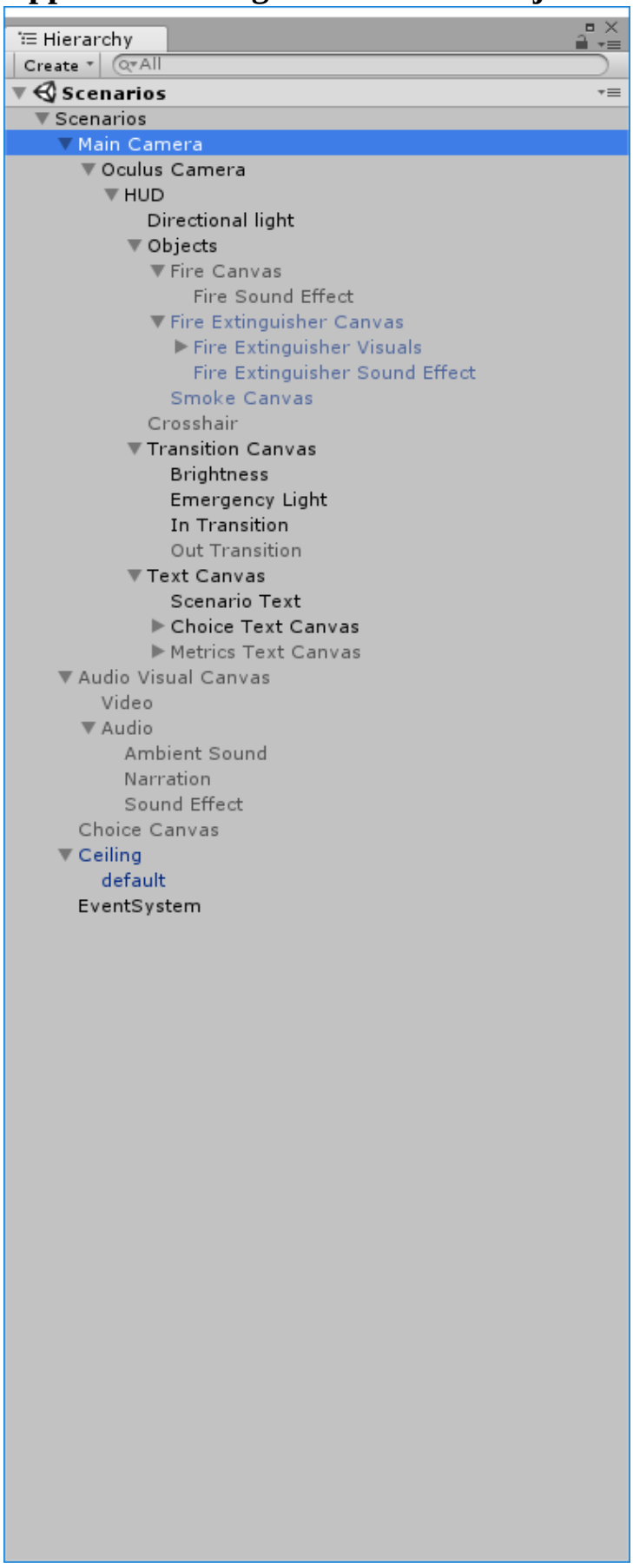
Appendix A – The API Data Format

```
        /// <summary>
        /// The list of choices that can be made from this scenario.
        /// </summary>
        private List<Choice> m_Choices;

        /// <summary>
        /// If empty string, go back to start. DO NOT ALLOW EMPTY STRING, ESSPECIALLY
ON FIRST SCENARIO.
        /// If video path but wrong, then display error.
        /// </summary>
        private string m_VideoPath;

        /// <summary>
        /// If empty string, no sound.
        /// Indicates the file path for the ambient sound file for the scenario.
        /// </summary>
        private string m_AmbientSoundPath;

        /// <summary>
        /// If empty string, no sound.
        /// Indicates the file path for the narration file for the scenario.
        /// </summary>
        private string m_NarrationPath;

        /// <summary>
        /// If empty string, no sound.
        /// Indicates the file path for the sound effect file for the scenario.
        /// </summary>
        private string m_SoundEffectPath;

        /// <summary>
        /// If empty string, no output. Must be empty after first scenario, I won't
check again.
        /// Indicates the file path for the output file.
        /// </summary>
        private string m_OutputPath;

        /// <summary>
        /// Empty indicates no scenario text.
        /// </summary>
        private string m_ScenarioText;

        /// <summary>
        /// Empty indicates no scenario choice text.
        /// </summary>
        private string m_ScenarioChoiceText;

        /// <summary>
        /// 0 - start of left right arc
        /// 1 - start of top bottom arc
        /// 2 - end of left right arc
        /// 3 - end of top bottom arc
        /// repeats
        /// </summary>
        private List<float> m_FireArc;

        /// <summary>
        /// 0 - start of left right arc
        /// 1 - start of top bottom arc
        /// 2 - end of left right arc
```

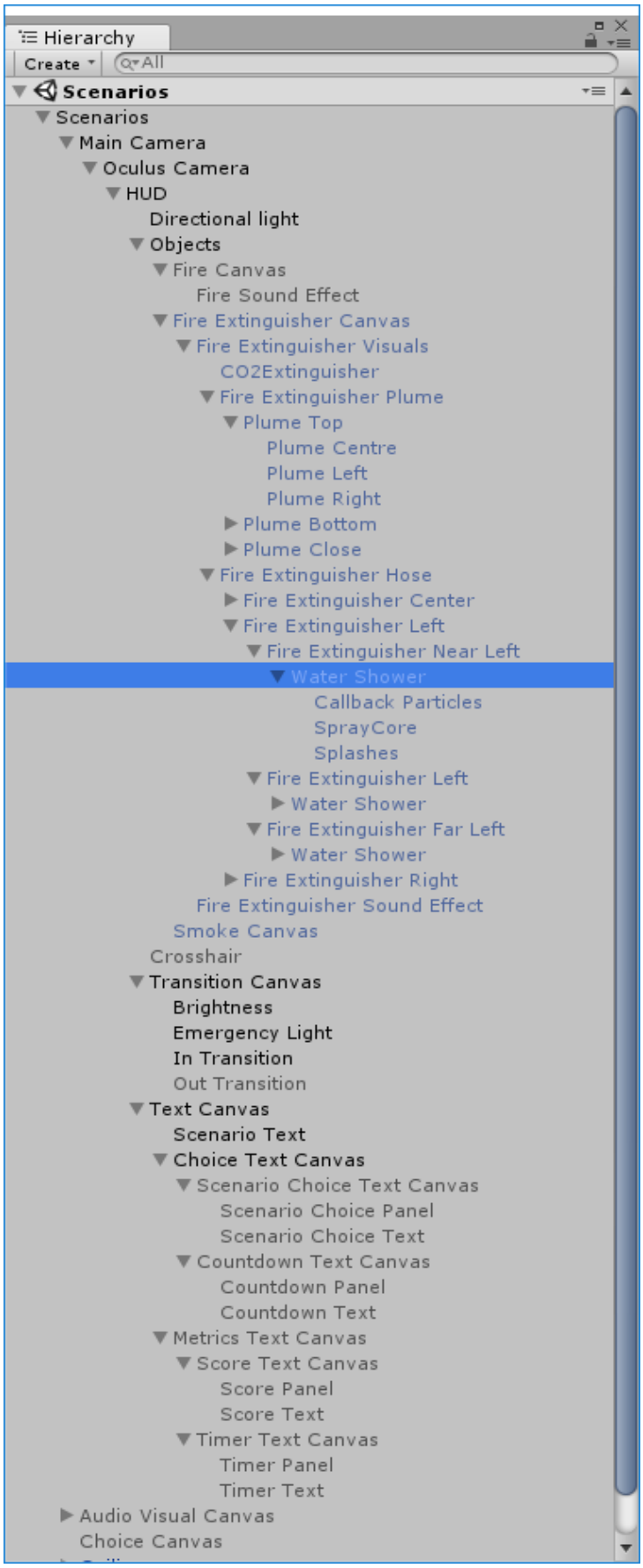
```
/// 3 - end of top bottom arc
/// repeats
/// </summary>
private List<float> m_SmokeArc;

/// <summary>
/// Sets in transition length.
/// </summary>
private float m_InTransitionLength;

/// <summary>
/// Sets emergency lighting intensity.
/// </summary>
private float m_VideoBrightness;

/// <summary>
/// Sets emergency lighting intensity.
/// </summary>
private float m_LightingIntensity;

/// <summary>
/// The ambient sound volume as a percentage (i.e. between 1 and 100).
/// </summary>
private float m_AmbientSoundVolume;

/// <summary>
/// The narration volume as a percentage (i.e. between 1 and 100).
/// </summary>
private float m_NarrationVolume;

/// <summary>
/// The sound effect volume as a percentage (i.e. between 1 and 100).
/// </summary>
private float m_SoundEffectVolume;

/// <summary>
/// Sets in choice length.
/// </summary>
private float m_ChoiceWaitLength;

/// <summary>
/// Indicates that the scenario has a smoke effect.
/// </summary>
private bool m_SmokeBool;

/// <summary>
/// Indicates the presence of a fire effect in the scenario.
/// </summary>
private bool m_FireBool;

/// <summary>
/// Indicates that the extinguisher effect should be activated.
/// </summary>
private bool m_FireExtinguisherBool;

/// <summary>
/// Indicates that the emergency light should be activated.
/// </summary>
private bool m_EmergencyLightBool;

/// <summary>
/// Indicates that the sound effect should be activated.
```

```
        /// </summary>
        private bool m_SoundEffectBool;
```

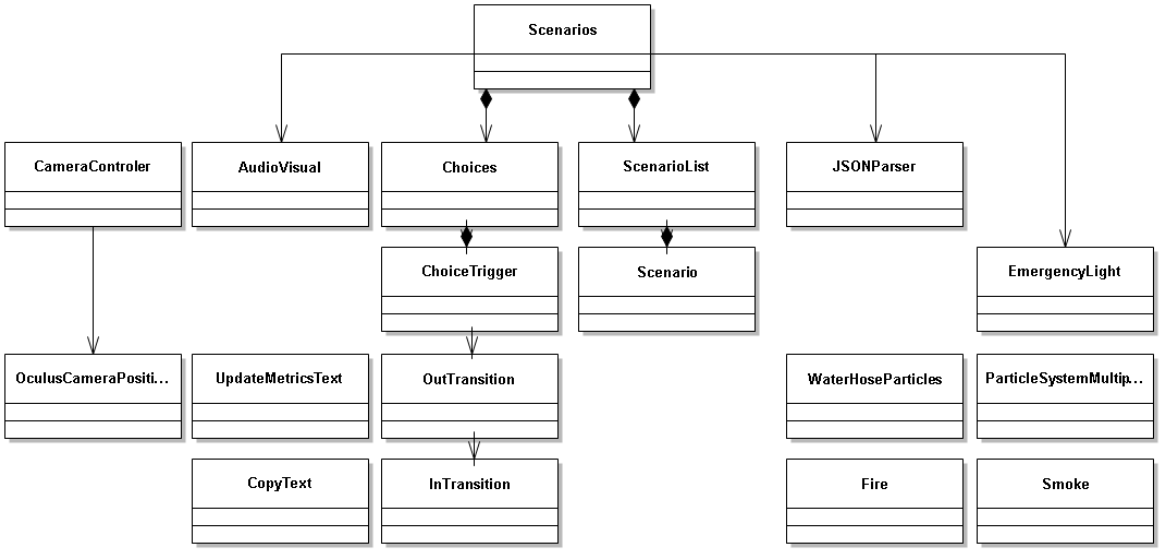
Appendix B – High-Level Game Object Hierarchy



Appendix C – Expanded Canvas Game Object Hierarchy



Appendix D – High-level Class Diagram



Appendix E – Example Output File

```
Went to 2
Current Score: 1
Current Time: 45.08372
Went to 3
Current Score: 2
Current Time: 66.99928
Went to 4
Current Score: 3
Current Time: 82.75295
Went to 5
Current Score: 4
Current Time: 110.8697
```