***Introduction to Graphics and Mobile Gaming***

**LAB 5**

**Virtual Reality**

**Issue 1.0**

Contents

[1 Introduction 1](#_Toc36132411)

[2 Creating a new scene for Google Cardboard 1](#_Toc36132412)

[2.1 Enabling VR support 1](#_Toc36132413)

[3 The scene 1](#_Toc36132414)

[4 The camera 2](#_Toc36132415)

[5 UI elements 2](#_Toc36132416)

[5.1 Code indications 3](#_Toc36132417)

[6 The menu 5](#_Toc36132418)

[7 VR and 3D modes 5](#_Toc36132419)

[7.1 Settings 5](#_Toc36132420)

[7.2 Toggle VR script 5](#_Toc36132421)

# Introduction

In this lab, we will adapt our existing game to allow the use of Virtual reality; we will do this by setting up a new scene. This will be linked to the main menu by adding a VR button that will load a version of our game that is VR enabled. We can use a number of different VR devices, but we will develop this for Google Cardboard, which is a cheap VR solution. Then, you will learn how you can extend the VR experience to other devices such as the Gear VR and the Oculus Rift.

# Creating a new scene for Google Cardboard

For this project, we will be working further on from our previous chess room scene, but we’ll be adding a new scene with the new VR mode.

## Enabling VR support

Unity has built in support for VR experience targeted at multiple devices (Google Cardboard, Oculus Rift, etc.). We chose Google Cardboard for this lab because it’s one of the cheapest and easiest options; you can build one yourself at home. More details on that can be found here: <https://vr.google.com/cardboard/manufacturers/>

To be able to use the VR features in Unity, please enable VR support.

In Build Settings:

* Select Android and Switch Platforms
* Navigate to Player Settings and then Other Setting. Make the minimum supported API 4.4
* Now select XR Settings. Tick the Virtual Reality Supported box. You will then be prompted to select a Virtual Reality SDK. Click on the little plus and select Cardboard.

# The scene

We’ll repurpose the main scene of the game for this lab, with minor adjustments as GUI elements such as buttons don’t work the same way in VR. Press Ctrl+D to duplicate the main scene and rename it “Game VR”.

* + - * Open this scene and delete the Canvas from the Hierarchy panel and any other objects that contain buttons. Buttons will not work properly in VR, and we’d like a clean interface for the user.
* Delete the room’s graphics elements, all text and all the GUIElements. VR does not display text very nicely, so we will look at another way to do this later.

# The camera

To create a VR scene onto a VR device, the images must be rendered on a SBS view (side by side). Consider how our eyes work; these have a slight separation of a couple of centimeters in between each other’s. Look to the image to the right; this is a rough estimation of how our view works. To create a VR experience close to reality as much as possible, the camera needs to recreate these visual conditions.

In order to trick the human brain, we need to give it two slightly different viewpoints of the same object, simulating the behavior of the eyes. To do this, we need to have two cameras with padding, one left and one right, to supply different viewpoints.

Thanks to the VR support enabled in Unity; the Main Camera of the game will now automatically render to the head-mount display you have selected (Cardboard) and track head movement to be able to change the field of view.

The scene will render to the two separate images, corresponding to each eye, and the device screen will eventually look similar to the picture on the right.

# UI elements

As mentioned before, text is not displayed properly in the VR setting. We will explore using ***TextMeshPro*** instead.

* Add a new empty GameObject and name it “textObject”
* Now try to add a TextMeshPro from GameObject -> UI-> ***Text-TextMeshPro***
* You will be prompted to import the TextMesh Pro package. You can import just the essentials, or alternatively if you wish to explore this more, import the examples and sample scenes as well
* Add a *Text-TextMeshPro* game object to the scene. We would like this text to display how much time we have let in the game. Name it “timeCounter”
* Add another *Text-TextMeshPro*. We would like this text to be displayed when the time is up. Name it “timeUp”

In VR, setting the location of the text is tricky because if it’s fixed, it won’t always appear in the field of view. Because of that, we would like our text to move along with the camera.

* Make the Canvas of your TextMeshPro objects a child of the “textObject” and then make “textObject” a child of the Main Camera so they can move together
* In the Inspector panel for the Canvas, change the Render Mode to World Space and then drag and drop the Main Camera in the Event Camera slot.
* Try to switch to the Game view instead of the Scene view so you can see the field of view. Now adjust the position of the TextMeshPro objects accordingly, resize the canvas, if necessary, so it fits inside the FOV and displays the text. Try to place the *timeup* text in the middle of the screen and the *timeCounter* in one of the corners so it doesn’t obstruct the view. You might notice you will have to place these texts quite close to the camera and decrease the font size so the text doesn’t interact with the meshes in the room.

Before, the text changes were handled by the scoreOnCollision script. You can go ahead and remove the component from where it’s attached as we can’t yet move the chess piece in VR, so we can’t score.

Instead, we will create a new script to change the text and attach it to an empty object that we will call “TextControl”. What we need to write here is similar to what was in the scoreOnCollision script, but we are no longer concerned with scoring or keeping track of winning conditions. Just the time.



## Code indications

First, we need to be using the text mesh pro namespace, so add this like to the top of your script:

using TMPro;

Next, we still want a float variable that keeps track of the time left in the game. We also want variables for our TextMeshPro objects. Make them *public* so we can set the objects in the Inspector panel, just like we did previously with the texts.

Finally, we want to keep track of whether the game has ended or not.

public float timeLeft = 20.0f;

public TextMeshProUGUI timeUp;

public TextMeshProUGUI timeText;

bool endGame;

Now, in the *Start()* function, we want to initialize the texts and our variables. The timeTextvariable will be initialized just as before, the timeUp text is an empty string for now, and the endGme is false.

void Start () {

 timeText.text = "Time: " + ((int)timeLeft).ToString () ;

 timeUp.text = "" ;

 endGame = false ;

 }

Next, we want our *Update()* function to update the time left in the game:

void Update () {

 if (endGame == false) {

 if(timeLeft >= 0){

 timeLeft -= Time.deltaTime;

 Debug.Log(timeLeft.ToString());

 timeText.text = "Time: " + ((int)timeLeft).ToString ();

 }

 if (timeLeft <= 0.2) {

 gameOverFunc ();

 }

 }

}

And finally, we want our *gameOverFunc(),* which will update the timeUp text:

void gameOverFunc(){

 timeUp.text = "Time's up!";

 endGame = true;

 }

Don’t forget to drag and drop the text objects in the fields of the script in the Inspector tab.

That is everything for the VR scene, now let’s explore the menu scene so we ca access the VR.

# The menu

In order to access the VR, we need to add a button that will handle the change of scenes. We already have a button ***in the menu scene***, so please proceed and add another one:

* File -> Open Scene and select the menu scene
* Find the Canvas in the Hierarchy, right-click -> UI -> Button
* Change the colour and the text to something suggestive like “VR GAME”
* In the Inspector tab, in the *On Click()* section, drag and drop the Canvas element; in the NoFunction section, select menuControl:changeScene and type in the name of your VR scene

The menuController script should already be attached, and it will handle the change of scenery.

# VR and 3D modes

## Settings

Now we have another problem to deal with: we’d like our menu and normal game scenes to be simple 3D scenes, but our VR to be in actual virtual reality mode. For that, all that has to change is the camera rendering.

Enabling native VR support in a project makes all the scenes in that project appear in virtual reality mode. In order to not have that, we will make the following changes:

* Navigate to Player Settings -> XR Settings
* Click on the little Plus that will show you different VR devices. Choose ***None*** to add to the list
* Drag it on top of the Cardboard device in order to prioritize it

Since ***None*** is prioritized, the scenes will open by default in normal 3D mode. To make our VR scene open in virtual reality mode, we need to write a script that enables the Cardboard mode and add it to the scene.

## Toggle VR script

Navigate to the scripts folder and create a new C# script and name it “VRToggle”. All we need to do is enable the XR settings for the Google Carboard, once, when the scene opens. There is no need for the Update() function, so go ahead and delete that.

We need to update the namespace to access the XRSettings:

using UnityEngine.XR;

In the Start() function, type the following:

StartCoroutine(LoadDevice("Cardboard"));

We use a coroutine to make sure that all changes are completed within once single frame update. Now we need to declare the LoadDevice coroutine:

IEnumerator LoadDevice(string newDevice)

 {

 XRSettings.LoadDeviceByName(newDevice);

 yield return null;

 XRSettings.enabled = true;

 }

Save the script and attach it to the Main Camera in the VR scene. You can attach it to any object; it won’t make a difference. The script will run once when the scene starts, and it will put it in virtual reality mode.

Make sure you have all scenes added to your build settings, with the menu being on top! Now build and run the app on your device and you’re good to go! Don’t forget your Cardboard!

This is the end of this lab and this series of labs. In this lab, we have implemented basic VR functionality into our game; this should provide a good basis for any VR content you wish to develop moving forward. We should now have a fundamental grasp on using Unity to design a full-fledged game experience for others to enjoy.