Suman Kharal

Game Development for International Red Cross Virtual Reality

Game Engine, Interaction with Gameobjects, Colliders, Raycasting and Game Development in Gear VR for International Red Cross

Helsinki Metropolia University of Applied Sciences

Bachelor of Engineering

Information Technology

Thesis

22 April 2018

Helsinki Metropolia University of Applied Sciences

Author(s) Title Number of Pages Date	Suman Kharal Virtual reality in Unity 3D Game Engine 40 pages + 2 appendices 22 April 2018
Degree	Bachelor of Engineering
Degree Programme	Information Technology
Specialisation option	Game Programming
Instructor(s)	Kari Aaltonen, Senior Lecturer

The goal of this thesis is to examine how the Game Engine Unity3D, can be used for the development of security assessment based Virtual Reality application. This study briefly discusses Virtual Reality covers the basic interface of the game engine. Also, how to customize content by writing scripts through programming language is covered as well as building the virtual reality application for International Red Cross.

Another aim of this study is to find out ways how a Virtual Reality based application can be used for educational purposes for Red Cross members. Moreover, it describes the important aspects of programming in Virtual Reality technology.

As the result the project, an Oculus Store, Google Play and Apple Store that can run on GearVR and Cardboard were created.

Keywords	Unity 3D, Virtual Reality, Gear VR, Cardboard, RayCasting And Programming, Mobile Game, Red Cross Security Assessment

Contents

Abbreviations

3D	Three Dimensional
2D	Two Dimensional
VR	Virtual Reality
HMD	Head Mounted Display
6DoF	Six Degrees Of Freedom
UI	User Interface
IDE	Integrated Development Environment
PC	Personal Computer
SDK	Software Development Kit

1 Introduction

Virtual reality has been one of the growing businesses in the current gaming and education industries. It can immerge people to the different environment without the need to travel and experience the place physically. This thesis is based on a project which was done while the author of this study was working with a company named Lyfta. Lyfta has been making three dimensional human stories from 2016 especially for the schools so that the children can understand more about empathy, critical thinking and break negative stereotypes. The Awra Amba Experience, Dinnertime 360 and Secrets of the Opera are the three products of Lyfta. Besides, Lyfta also works as the agency for the other projects. Lyfta was offered to make the Virtual Reality based application for International Red Cross staff members. The name of the application was Humanitarian Security - Stay Safe [1]

Training the staff members can be challenging in many ways as the person has to go to the Red Cross location in different countries to take the lesson about safety and security. To overcome this challenge the 360 degree storytelling virtual reality application was created to train the staff members about the safety and security measures they need to think about before moving to the new location. In this Virtual reality application person is taken to the 360 degree video space in a location of a country. After that the person is presented with three different locations in a map where the person can move to. The person needs to analyse the three locations based on the factors like the service stations near the place, natural hazards, crimes and route to the emergency places. After choosing the correct location person is furthermore taken into the series of different spaces representing different part of houses like kitchen, bedroom, garden where person needs to look around the space and figure out the most common safety and security measures to consider in each spaces. The goal of project was to build virtual reality application for three different platforms. The project was done in the period of six months which gave me really good experience in programming. Besides, the applications were distributed in Google Play, IOS Store and Oculus Store and project could run in GearVR and Cardboard. I was assigned as a lead programmer for this project.

2 Virtual Reality

2.1 Overview

Virtual reality can be described as a three dimensional computer generated environment. It is a graphical representation of real life combined to function with our senses so that it gives the feeling of being immersed in the environment. The more senses of our brain like our vision, touch, smell, hearing it stimulates in certain manner, the more immersed we feel to the Virtual Reality. In real life we depend on a lot of different senses to cope with daily life activities and the virtual reality devices should be able to stimulate these senses to take us to realism. There are a lot of different kind of devices which can be used for the virtual reality experiences like different kind of head mounted display (HMD), controllers, globes or even clothes.[2]

The immersion in virtual reality can be in different levels. These can be categorized in as Non-immersive, Semi-Immersive and Fully-Immersive simulations. Non-immersive simulations are the ones which a lot of us are familiar with. The use of any desktop computers with mouse and keyboards can be considered as Non-immersive simulations. Semi-Immersive simulations use more complex sensors than non-immersive simulations and keep your presence more in the experience. Flight simulators or dark driving test in driving lessons can be considered as Semi-Immersive simulations. The Fully-Immersive simulations are a lot more complicated. In fully-immersive simulation, the person's field of view is replaced with the computer generated graphics which takes person to different state of reality. Head mounted display (HMDS) is an example of the fully-immersive simulations. Examples of HMD devices are Oculus Rift, Gear VR, and Google Daydream etc. The HMD devices use many kinds of sensors, lenses, and different kinds of input devices. [2]

2.2 History of Virtual Reality

Although the hype of Virtual Reality is really high at the moment. The history of it goes way back as far as 1957. In 1957 the filmmaker named Morton Heilig invented the machine called Sensorama. This machine gave the people complete immersion into 3D world and had effects such as vibration, stereo sound and even atmospheric effects like wind blow. This invention was followed by first head mounted display(HMD) developed by Ivan Sutherland in 1968. This “ultimate display” was capable of displaying the simple virtual wireframe by connecting the stereoscopic display which

can be also seen as birth of first Augmented Reality. However the project was not used much due to its massive size and was given the nickname of “Sword of Damocles”. This trend was followed by The Aspen Movie Map in 1978 and Sega VR in 1991. The Sega VR releases the press note stating that “As your eyes shift focus from one object to the next, the binocular parallax constantly changes to give you the impression of a three-dimensional world”. They also released few games for this console though it was not successful with some fearing that this might have negative impact on the users. [3]

The first major development was seen in 2010 by the company named Oculus. Till this day Oculus remains one of the leading platforms for the virtual reality. Oculus was founded by Palmer Luckey and later sold to the Facebook in 2014 [3]. The oculus rift has its own distribution channel for the virtual reality based applications. Not long ago in 2017 there has been major development in the VR devices. Technology giant such as Google, Microsoft has been in constant race for the development of such technology and providing the necessary development tools for the developers. In 2017 Unity had built in support for the different VR platforms like Oculus, Gear VR, Hololens and much more. [3]

2.3 How Virtual Reality Works.

There are a lot of key components included when it comes to virtual reality. The most common HMDS include components such as sensors, head tracking, motion tracking, lenses, display, audio and much more. The sensor includes mostly of accelerometer, gyroscope, and magnetometer. These three sensors have different purposes. Just as defined, the accelerometer measures the acceleration of the device .Accelerometer is also used for finding which way is up. Multiple accelerometers are assembled together for measuring device orientation in relation to gravitational pull. Gyroscope is used for finding the orientation of device which is a key component in VR. By measuring the magnetic fields magnetometer tells the direction the device is facing in the earth. All of these three components are combined together for tracking in HMDS. And this is used for gaining the six degree of freedom knows as (6DoF).

The head tracking tracks your direction of our head like left, right, down up. The head tracking uses all of the sensors such as accelerometer, gyroscope, magnetometer to find the person's position, rotation, and distance in three dimensional spaces in terms in x, y and z axis as demonstrated in the Figure below:

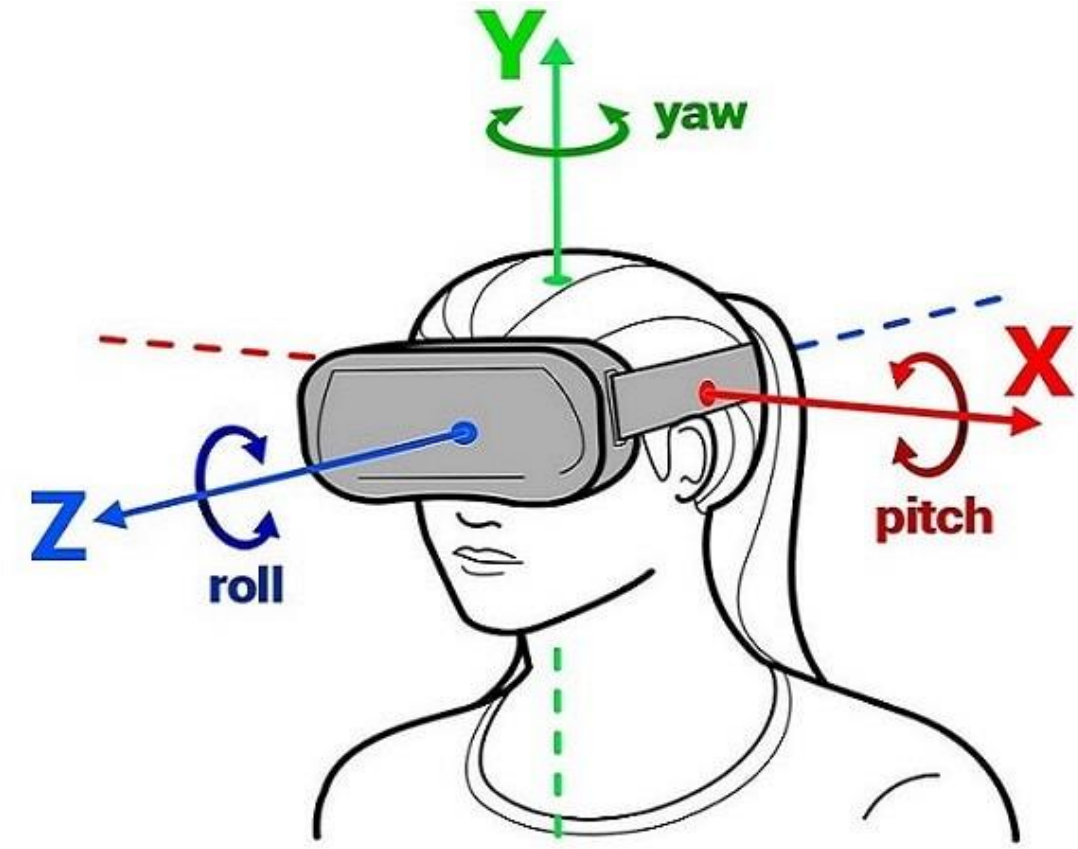


Figure 1. Head Tracking [4]

Display and lenses are also important part of VR. The lenses are placed between the eye and the display. The lenses mimic how our real eyes perceive the world by focusing and reshaping. It angles the two dimensional images in terms of world view also known as stereoscopic images. The display can be of different kind and it is a video feed that can be rendered directly from our smartphone or from the computer display. It renders the images that the user sees through the lenses. The display can be one eye or two eye display depending on the headset.

Besides, other key components are the field of view, which is the area or the angle the person can see from the headset which is normally 110 degrees. When running VR applications it should run at the minimum frame rate of 60fps to avoid legacy and the discomfort when using the device. Low frame rates can cause the feeling of nausea and motion sickness. Audio in VR gives the feeling of more immersion in the VR. Spatial sounds can be used for VR apps which give the feeling of the sound coming from different places in three dimensional spaces which are really important when it comes to VR because our brain is really aware of sounds coming from right or left ears. There are also different kinds of input devices which act as the mouse or keyboard in

our general use of computers. Theses inputs can be interpreted and used for performing desired action when the physical button is pressed. The input devices may also contain the motion sensors just like in HMDS for the motion tracking which can act as our hands in VR. [4]

3 Game Engine Unity 3D

Unity3D supports both 2D and 3D game creation. This study focuses on the three-dimensional environment rather than the two-dimensional environment. It is a rich and extendable editor, it has powerful graphics rendering such as real time rendering and Native Graphics APIs. Besides, it supports programming in C# and can deploy project in more than 25 different platforms across TV, VR, AR, desktop, mobile, console and web. Many companies and teams use Unity as it supports the team collaboration by synchronization, cloud build and also supports version control like git. The built-in Monetization is very helpful for individuals and companies for revenue. In the topics below we will be discussing about the basics of Unity editor.

3.1 Scene View

Scene view is where all the magic happens. Most of the development of the game is done in the Scene view. Everything from Lights, Models, and Colliders are constructed in there. It is the editor window where you can easily change the view of your game through camera, look around and position all the elements in the game as demonstrated in the Figure below. Most of the visual work is done here as it is easy to look around place the elements of the game in 3D space.

Figure 2. Screenshot of Scene View in Unity [Screenshot 2017]

As shown in Figure 2, for development of Gear VR and Cardboard environment, the 360 image is placed around the sphere and the elements such as the menu in the game can be easily dragged on different positions using the gizmos shown with green, blue, and red line. Where red represents X axis, green Y axis and blue Z axis in 3D space. All the elements in the scene view are made of different types of GameObjects. Sphere present on the scene is GameObject. Likewise, the different buttons present on scene are also GameObjects. [5]

GameObjects

In Unity GameObjects are like an empty pot. They can range anywhere from Characters, Lights, Camera, Environment, Weapons to Coins or even empty elements with certain characteristics. We assign certain characteristics to those GameObjects by adding certain components to them. These components can range from Transforms, Colliders, and different physics components to self-made scripts. You can create simple shapes such as Sphere, Cube, Plane, and Terrain from Unity Itself. It also contains other elements like lights, camera and UI elements [5].

3.2 Game View

Game view is the representation of the view which the player sees when the person plays the game. It is the initial representation of the camera placed on the 3D space, as Game View works in correlation with the camera positioned in the Scene View rotating or positioning the camera effects how the game looks in the initial state and during gameplay.

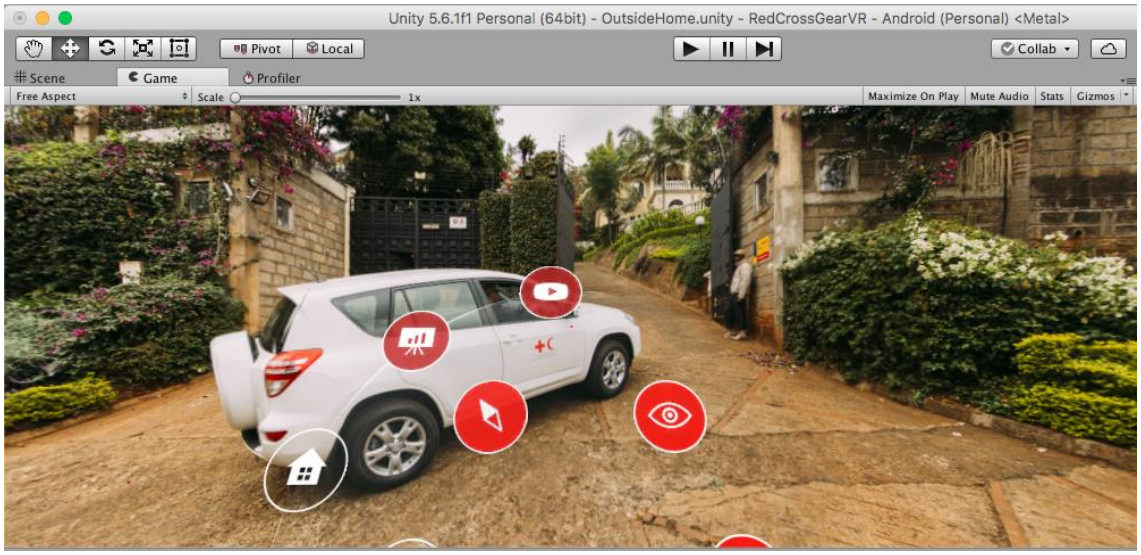


Figure 3. Screenshot of Game View in Unity [Screenshot 2017]

As seen in Figure 3, the Game view differs completely from Scene View as the camera is positioned on the centre of world axis and the game view displays the original state of game in any devices when built. [5]

3.3 Inspector Window

Inspector View is one of the most important aspects of programming in Unity as it contains the detailed information of the Meshes, Transform (Containing information of Position, Rotation and Scale of 3D object), Scripts and much more of the currently selected GameObjects. The behaviours of the GameObjects can be easily modified from this Window as the initial behaviour of any GameObjects.

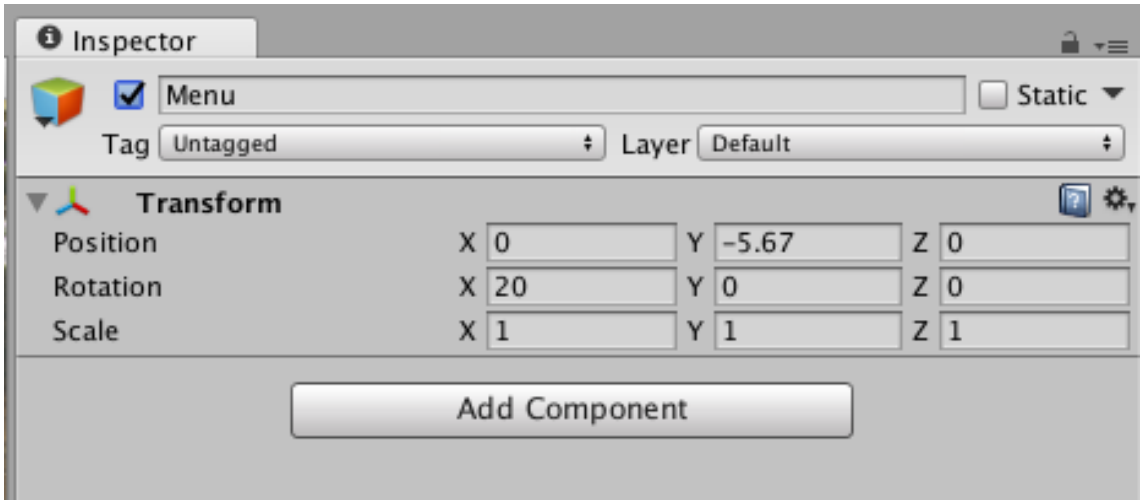


Figure 4. Screenshot of checked GameObject in Unity. [Screenshot 2017]

As shown in Figure 4 the Transform element contains the information of the Position, Rotation and Scale of a specific GameObject called Menu. The small blue check button on side of Menu means that the GameObject is presently active in the scene.

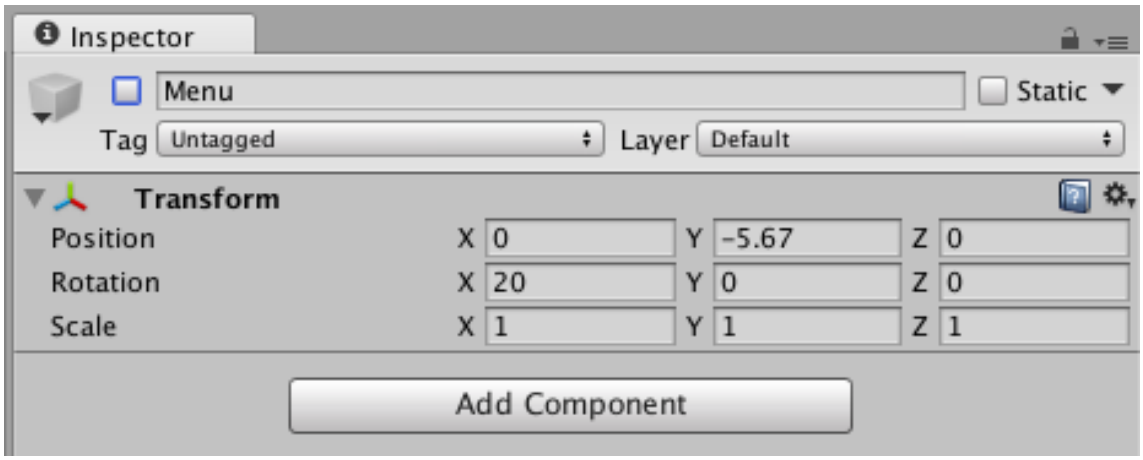


Figure 5. Screenshot of unchecked GameObject in Unity. [Screenshot 2017]

As seen in Figure 5, unchecking the GameObject is done by clicking again on the box left to Menu Text. [5]

3.4 Hierarchy Window

Hierarchy Window has list of all the GameObjects present on the scene. Parent and Children is one of the important aspects of the GameObjects present on the Hierarchy View. You can make the GameObject Children of any Parent by simply dragging the object into the GameObject. Some of the behaviours of Children objects are affected by the parent GameObjects as they are associated with parent objects.

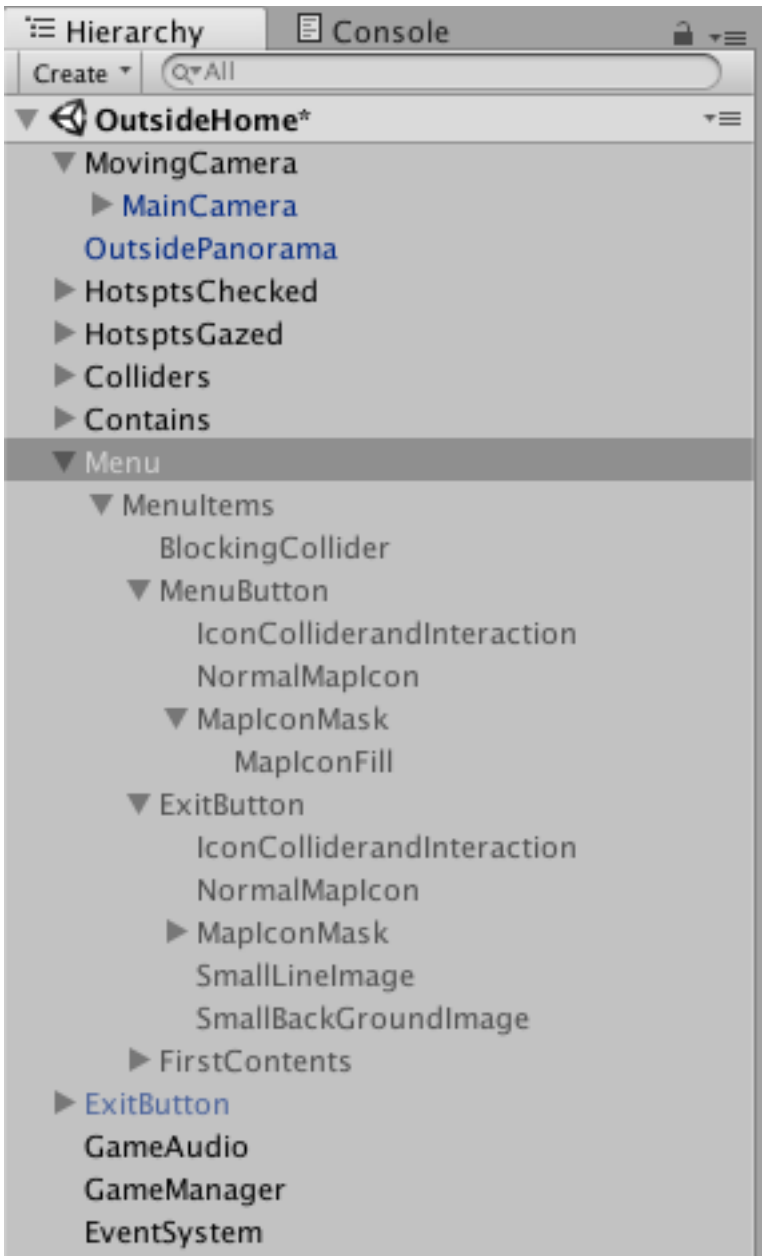


Figure 6. Screenshot of Hierarchy Window in Unity 3D [Screenshot 2017]

As demonstrated in Figure 6, Menu contains a lot of elements inside it, making it a parent object and rest of the objects children of parent objects. Unchecking the menu icon in inspector window had unselected all the children objects also. This is represented by greying out all the children objects. Changing the transform component in Figure 4 will have effect on all of the children objects also. [5}

4 Programming in Game Engine Unity 3D

Although Java script is not supported anymore by Unity 2017.2 beta which was released in 2017, Unity editor analytics statistics done in September 3, 2014 showed that 80.4 percentage users used C#, 18.9 percentages used Unity Script (also known as JavaScript in Unity) and rest of 0.44 percentages used Boo as their programming language for Game Engine Unity [6]. See Figure 7 below:

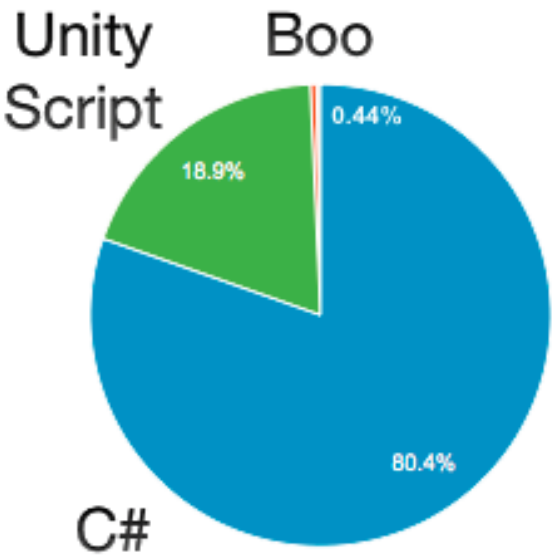


Figure 7. Use of Different Programming Languages in Unity [Picture 2014] [6]

As we also use C# programming language for our project and C# is most widely used language in Unity Game Engine, we would be focusing on C# language throughout the thesis.

4.1 C# programming

According to Microsoft Corporation “C# is an elegant and type-safe object-oriented language that enables developers to build a variety of secure and robust applications that run on the .NET Framework”. [7]

C# is widely popular object oriented programming language and used for development of variety of applications. C# is quite similar to other programming languages such as

C, C++, and Java but it is more advanced and supports more features like delegates, enumerators and provides direct access to memory also.

C# in Unity

C# is more widely used in Unity than in other programming languages. Most of the tutorials in the Unity official website are written in C#. Monodevelop and Visual studio were supported for creating scripts in Unity. But during 2017 monodevelop was not supported anymore. We will be mostly using Visual Studio for this project as it makes reading and formatting code easier and support wide range of tools.

Creating Scripts

Creating a script is like creating a certain characteristics. In Unity this scripts can be implemented to any GameObjects and their characteristics can be tweaked during the gameplay. Unity software comes itself with different built in components which makes the game making process relatively easy but you can also create your own components using scripts and make the GameObjects in game function on your own desire. You can create a new script in Unity by going to Create and create new script.

As Unity is really a vast interface we will be focusing mostly on the programming used in the current project.

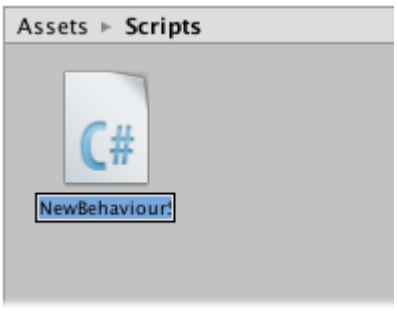


Figure 8. Creating a new script in Unity

After creating the new script in Unity, it comes with default Start () and Update () function as can be seen in Figure 9.

```
using UnityEngine;
using System.Collections;

public class MainPlayer : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

Figure 9. Default state when new script is created in Unity.

The scripts help to build the connection between the Monobehaviour which is a built in class with the internal working of Unity. The initialization of GameObjects are done usually in Start () function, which means that the Start () function is called at the initial frame of gameplay. However, the Update () function is called every frame of gameplay. So, all the movements, calculations and detection of inputs are handled in the Update () function. Basically this function is mainly responsible for handling all the events over the time that happens during the gameplay. 8]

4.2 Gameplay Programming

When programming in Unity, GameObjects are the important part of programming, you can refer to GameObjects and access the various components of the GameObjects. Since, GameObjects have different properties on them like Transforms, Colliders, Rigid body to all the properties you assign to them through scripts. It is important that you can access this component in runtime so that you can make the GameObjects function as you want. Let's suppose you want to access the Transform property of a GameObject called crossedCube, which has information on the Position, Rotation and Scale of it in the Global space. First you need to reference to the GameObject you want to change properties of.

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

public class CubeMovement : MonoBehaviour {
    public GameObject crossedCube;
    private Transform cubeTransform;
    // Use this for initialization
    void Start ()
    {
        cubeTransform = crossedCube.GetComponent<Transform> ();
    }

    // Update is called once per frame
    void Update ()
    {
        cubeTransform.position += new Vector3(0.0f, Time.deltaTime, 0.0f);

    }
}

```

Listing 1. Movement of cube in y-axis.

Listing 1 illustrates how the GameObject is referred to as public property and the transform values can be accessed from it. Getting the component from a GameObject is done by GetComponent<> () method, where the <Transform> is the type of the component. As demonstrated below in Figure 10, we are moving the cube along the Y-axis in world space coordinates. The movement part is done in update function and the transform position in Y axis is added with Time.deltaTime (frame float value) which moves the cube in Y direction.

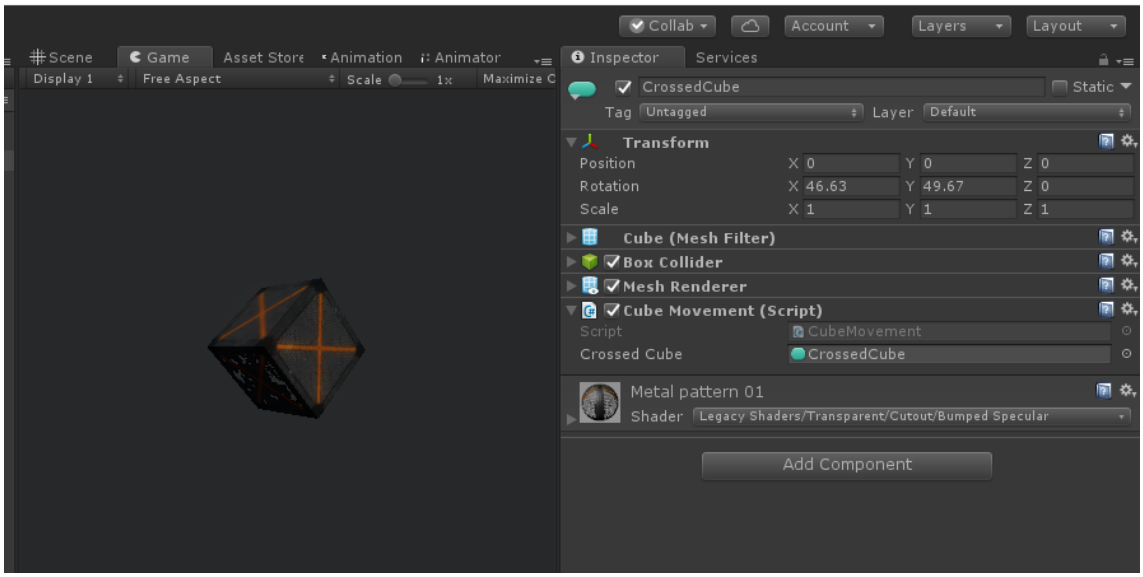


Figure 10. Cube moment along the Y-axis

As can be seen from Figure 10, the CrossedCube GameObject has the cube moment script attached to it. And the public variable Crossed Cube from Figure 10 has reference to CrossedCube GameObject which is referenced by dragging and dropping the GameObject in the Inspector window. As the CrossedCube GameObject has been referenced, the components from it can be edited in runtime through scripts. Like the Transform, Box Collider, Mesh Renderer, Mesh Filter and the materials which are attached to the GameObject.

4.3 Getting Input in Game Engine Unity 3D.

GameObjects can be interacted in different ways depending on the platform you are building the game for. If you are making a game for the PC than you use keyboard and mouse as your main input source, whereas, for the mobile phones your input can be mostly touching the screen. The interaction in Gear VR is slightly different as you cannot use mouse or touch to interact with the GameObjects in Virtual Reality [9]. Although there is different input available for interaction in VR like the buttons in HMD itself and also VR controller available, the most common use of interaction is through gazing, we will be using Unity Raycasting feature for this, which we will be discussing in coming topics. Raycasting enables us to look at different kind of objects present in VR and interact with them with gaze. [9]

During gameplay user can touch on different kind of objects in case of mobile phones. And a lot of times this events needs to be interpreted by the program and give the required result. These inputs can be handled by different classes in Unity. For the PC computers Input class is used and in case of mobile phones Touch class can be used to interpret these events and achieve the required results.

```
void Update ()
{
    if(Input.GetMouseButtonDown(0))
    {
        Debug.Log("Left Click is pressed from mouse");
    }

    if(Input.GetKeyDown(KeyCode.A))
    {
        Debug.Log("Key A is pressed from keyboard");
    }

}
```

Listing 2. Getting Input by calling Input Class

As seen from Listing 2, we put the code in the Update () function as we should know the input given by user during runtime. The above code handles the inputs from the keyboard and mouse of PC. The message “Left click is pressed from mouse” is displayed when user presses the left mouse button and message “Key A is pressed from keyboard” is displayed when the user presses the A key from keyboard. Same kind of results can be achieved from the Touch class for the touch events. In case of VR we mostly interact with object when the person gazes at the object so this kind of approach is not suitable.

4.4 RayCasting

Raycasting is a way of drawing the invisible line from certain point of world space. Raycast has a lot of uses in the gaming as it can find which object has been hit by the ray. This ray is not visible for the user and it can receive different information’s of the GameObject. The distance of the ray also can be defined so that the object is detected within the certain range.

```
// Update is called once per frame
void Update ()
{
    if(Input.GetMouseButtonDown(0))
    {
        Ray ray = Camera.main.ScreenPointToRay(Input.mousePosition);
        RaycastHit hit;
        if(Physics.Raycast(ray, out hit , 100))
            {
            GameObject go = hit.collider.gameObject;
            Debug.Log (go.name);
            }
    }

}
```

Listing 3. Example of raycasting.

As demonstrated in Listing 3, an invisible ray is drawn from the mouse pointer when the left mouse button is pressed as represented by GetMouseButtonDown (0). A Ray is represented by the name of ray in small letters or can be any user given name, hit holds the information of the object hence different properties of object can be retrieved from it. In the above example we are casting a ray of one hundred meters in Unity units. If any GameObject is placed in range of 100 meters from the origin and left mouse button is clicked the name of the GameObject is displayed to the user.

In case of VR we mostly cast the ray from the centre of mobile device and interact with object.

4.5 Delegates and Events

Delegates

One of the most common practises of delegates is subscribing and unsubscribing the methods, they can be also considered as the list of functions or some sort of variables. Where variables contain data, delegates contain functions which can be assigned and unassigned during runtime. We can create a delegate type by use of keyword delegate. Delegates can be also multicasted which allows delegate variable to hold multiple functions.

```
public class ClassOne : MonoBehaviour {

    delegate void SimpleDelegate(int numOne, int numTwo);
    SimpleDelegate exampleDelegate;
    void Start ()
    {
        exampleDelegate += Sum;
        exampleDelegate += Multiply;
        if (exampleDelegate != null)
            exampleDelegate(10 , 15);
        Debug.Log("Unassigning Sum");
        exampleDelegate -= Sum;
        if(exampleDelegate!= null)
            exampleDelegate(5, 15);
    }

    private void Sum(int _first,int _second)
    {
        Debug.Log("Sum of numbers is " + (_first + _second));
    }

    private void Multiply(int _first, int _second)
    {
        Debug.Log("Multiplication of numbers is " + (_first * _second));
    }
}
```

Listing 4. Example of delegates

Above is the code snippet that produces the result shown below.

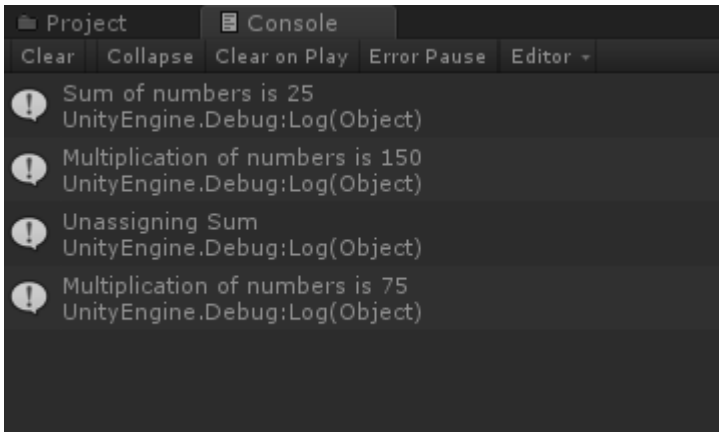


Figure 11. Output of Listing 4

As demonstrated in Listing 4, a delegate of name SimpleDelegate is created which takes two integers as parameter, and then we create an exampleDelegate of type SimpleDelegate, than we assign two functions to SimpleDelegate with plus sign and call the delegate. As the result seen in Figure 11, calling the delegate in return calls the two methods and returns the result; in this case the result is sum and multiplication of

two numbers. It is important to note that the methods should take the same type of parameters as the delegate type. Also, calling the delegate without assigning any methods and gives an error. So, it is always good practice to check for null exception error before calling the delegate. As also can be the Listing 4, we can deassign the methods with minus sign and calling the delegate again does not call the deassigned method.

Events

Events can be thought as the broadcast system. When some event happens other classes can know about that event and complete the necessary task. For example when the button is clicked or something happens than the classes which are interested in that event can listen and do something when that situation occurs.

```
5   public class ClassOne : MonoBehaviour {
6
7
8       public delegate void SimpleDelegate(int numOne, int numTwo);
9       public static event SimpleDelegate exampleDelegate;
10
11      private void Update()
12      {
13          if(Input.GetKeyDown(KeyCode.C))
14          {
15              if (exampleDelegate != null)
16                  exampleDelegate(100, 5);
17          }
18      }
19  }
20
```

Listing 5. Delegate with the event assigned

```
public class ClassTwo : MonoBehaviour {

    private void OnEnable()
    {
        ClassOne.exampleDelegate += Sum;
        ClassOne.exampleDelegate += Multiplication;
    }

    private void OnDisable()
    {
        ClassOne.exampleDelegate -= Sum;
        ClassOne.exampleDelegate -= Multiplication;
    }
    public void Sum(int _first, int _second)
    {
        Debug.Log("Sum of numbers " + (_first + _second));
    }

    public void Multiplication(int _first, int _second)
    {
        Debug.Log("Multiplication of numbers is " + (_first * _second));
    }
}
```

Listing 6. Class responsible for listening to the event

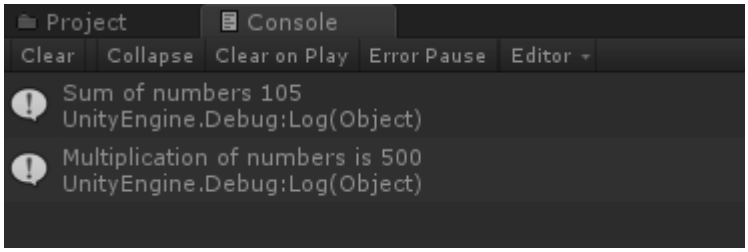


Figure 12. Output of code snippet of Listing 5 and 6

As we see from Figures 12 and listings 5 and 6, ClassOne has an event exampleDelegate of type SimpleDelegate. We check the users input in update function and when the user presses the C key in the keyboard we invoke that event has happened. On the other hand ClassTwo is listening for that event to happen and perform the necessary task. In this case the class prints the sum and multiplication of two numbers. In the above example it is demonstrated that the ClassTwo does not need to know or have reference of ClassOne but can perform the task when certain action takes places in the ClassOne. This idea can be used for multiple classes also. Example when the Key C is pressed in ClassOne there can be different part of classes and codes listening for that event to happen and perform different tasks accordingly.

4.6 Coroutines

Calling the methods is done in one frame. For example, the start function is called in the first frame of the game and update once per frame. This limits the creation of procedural animation, or different kind of events over the time period. For example, if we create a function which has For loop and responsible for moving the three-dimensional object one Unity unit in x axis per frame until 10 units. When we execute the function we will see that that the object moves 10 units directly as the whole function is executed in one frame and the movement that is intermediate is not visible. In this kind of situations the coroutines are effective and useful. A coroutine has capability to pause the execution and continue from the point where it left in Unity.

```
public class SimpleCoroutine : MonoBehaviour {

    public GameObject simpleObject;

    void Start ()
    {
        StartCoroutine(SlowMovement());
    }

    IEnumerator SlowMovement()
    {
        for (int i = 0; i <= 100; i++)
        {
            simpleObject.GetComponent<Transform>().position += new Vector3(0.1f, 0, 0);
            yield return null;
        }

    }
}
```

Listing 7. Slow movement of cube by use of coroutines

As shown in Listing 7, coroutines can be created with name IEnumerator and function name. The execution of program stops at yield return and comes back to that point in the next frame if the return type is null. Furthermore, the coroutines can wait for the certain time and come back to execution at user defined time. In the above example we see that the SlowMovement coroutine is called in the start function. The coroutine is responsible for making one hundred loops and moving the object in x axis at the rate of 0.1 units for frame. Usually when this kind of loop is executed inside the function the whole one hundred loops are made in one frame. Where, in this case one loop is made every frame, so the movement of the GameObject is smooth and visible to user.

Coroutines can be made for any cases and this is useful when we want to get the result gradually.

5 Stay Safe VR application

Humanitarian Security - Stay Safe virtual reality application was based on the idea that the staff members of red cross could analyse and understand the different kind of security measures person might need to take like the public services, disaster areas, crime zones and more. For the 360 environment the team of photographers went to the third world country to shoot the 360 panoramas and videos. The pictures were taken from one of the Red Cross office in a third world country.

First, the pictures from different places of the house were taken. One 360 video from the street of this country was also taken to demonstrate the daily life activity of the street. There were 9 different virtual environments existing in project, each scene in Unity representing each environment. These places were Intro space with video, Briefing room where the person was presented with a map in world space canvas in three dimensional spaces and needed to analyse the suitable house. The person needed to go through the three locations where two locations were in bad place and the third location being suitable place. These three locations were analysed looking the map with different locations of house and answering the quiz questions where person could choose yes or no as an answer as illustrated in the figure below. After answering each question correctly for each location, the person could move to next location and answer the question.

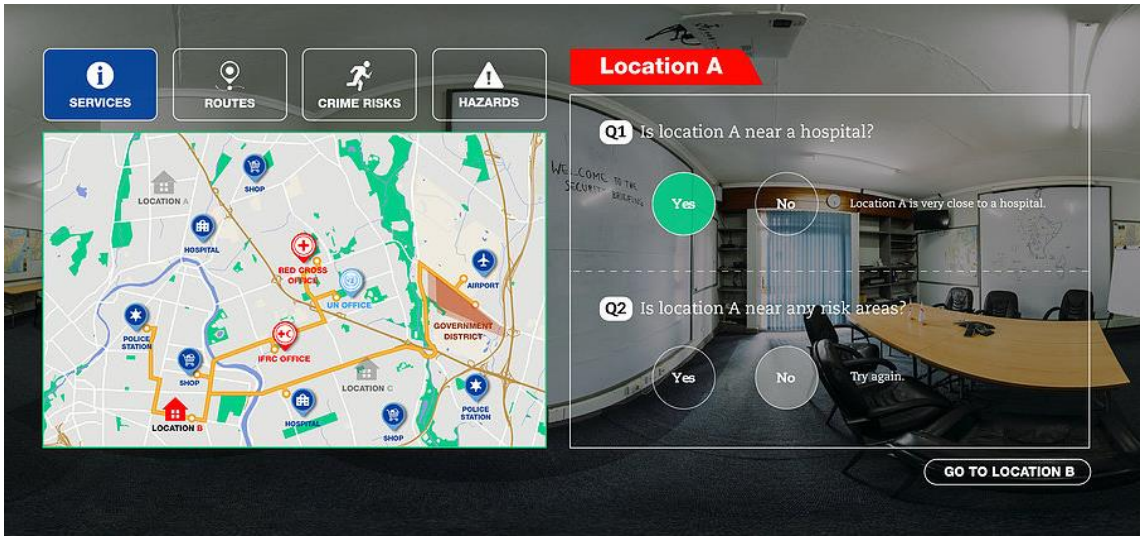


Figure 13. User Interface for analysing the map [Lyfta][1]

After the correct location was found which location C was, the person was taken to different space. This space started from the outside of a house where person needed to find different kinds of security measures to consider before entering home. These security measures could be present on any area of panorama image and could be found by gazing on it. When the person gazed on the area of interest also called as hotspot for half a second that area would light up and reticle representing where person would is looking would be filled. After the reticle was filled the information about the security measure of that area was displayed in world space canvas for the person to read.

Furthermore, the information on how many hotspots were found was visible on the person's field of view. After all the hotspots were found person was taken to other spaces sequentially. Other spaces were Driveway, Hallway, Bedroom, Kitchen, Living Room, Garden and Rooftop. These spaces represented most common areas of house where the person needed to look out for the security measures. For example, in the kitchen environment person needed to look out for measures like if there are any fire extinguisher, or know about where the electric box was present. Each space had different kind of security measures and the person needed to look and think carefully to find out those areas of interests. Besides, there was a menu which was accessible for the user all the time for exploring different scenes and to get hints for finding the hotspots.

5.1 Setup of Scene

As the virtual reality experience in this application was not based on in any movements of users but rather user would stay on same place and analyse the environment, we set up the sphere with the normal inverted and put the camera at the centre of the world space to create the environment. The sphere with inverted normal was created with the application called blender. Blender is software for creating 3D models.

Textures, Shaders and Materials

In Unity the textures, shaders and materials work closely with each other. The textures are the bitmap images. The shaders are used for calculation of the pixels rendered, which in turn are the scripts based on different algorithms and mathematical calculations. The standard shaders are most commonly used in Unity as they are less

performance intensive and also customizable highly. The materials are used for wrapping the 3D models or other GameObjects and define how the surface of the objects should be rendered. A material can have different kind of properties like alpha, emission, height maps, color values and much more depending on what kind of shaders are used. Materials can be created from the Hierarchy Window from the create option.

As the environment surrounding the player or user while in virtual reality was basically a 360 image. A material was created with the 360-panoramic image as a texture. This material used unlit texture shader. As the materials are affected by the lighting of the surrounding, this shader was chosen as we wanted the 360 environment to be as it was photographed. The unlit texture shaders are not affected by lightning hence good option while rendering the real life photographs or images in the environment. This material was applied to the sphere that was created earlier.

Figure 14. Basic setup of VR scene.

When you apply the material to the normal sphere and place the camera in the middle nothing from the sphere is rendered as the material is wrapped around it from outside

but if you invert the normal and apply the material to the sphere than the material is rendered inside as seen in the bottom right corner of Figure 14. Hence, the 360 degree environment was achieved. [13]

5.2 Creating User Interface

UI is one of the most important visual parts for any application. UI known as user interface has the self-explanatory meaning. It is the easiest way for the interaction between human and computers. Also widely known as graphical user interface it is most common use in most of the applications and desktop computers itself. Buttons, Images, Input fields are the basic examples of user interface. Where user clicks the button and it triggers something to happen and the possibilities are much more. [14]

While we use the desktop computers or mobile phones we can see different kinds of buttons for interactions in the screen. These user interfaces are easy to create in Unity for the visualization. When creating the user interface we need to have a Canvas.

Canvas

Canvas is the boundary or an area where all the UI elements should exist. If there is no canvas present and UI element is created than Unity automatically creates the canvas and the UI element resides inside it as a children. A Canvas has different properties like Rect transform which has different information like size, pivot point, anchors. Along with different properties that can be on the canvas there is a property called Canvas itself. Within the Canvas property we can find the Render Mode setting, which we can choose to different options. These settings include Screen Space - Overlay, Screen Space - Camera and World Space. Screen Space - Overlay refers to the interface that is overlaid in top part of the screen of different devices, whereas Screen Space - Camera refers to an interface that is certain distance from the Unity camera which allows a depth in the UI and different kind of appearance that can be achieved by changing the camera properties. Whereas World Space render also known as “diegetic interface” camera can exist within the 3D space itself which allows us to put the UI in the front or back of 3D objects.

As we cannot see our phone or computer screens when we are in virtual reality we use the world space canvas to render the UI elements. In this project all the UI elements

were created in world space canvas. It is also important to note that if the UI elements are too close to your eye it would be difficult to see the elements and also might cause nausea. The position of your head or eye is represented as Unity camera when application is built and all the UI elements should be at least at reasonable distance from camera for the user to read the information.

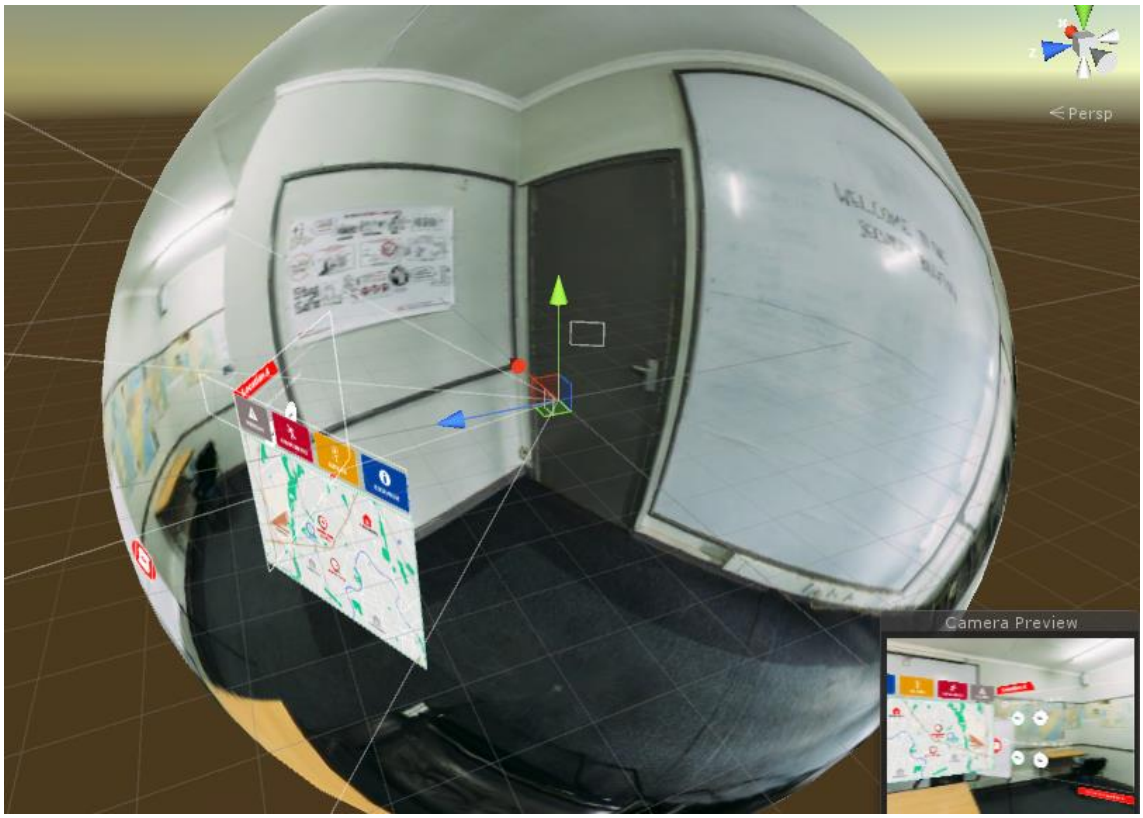


Figure 15. Scene view of user interface for analysing map

As seen in Figure 15. The User Interface for analysing the map is placed in 3D space; however the UI elements cannot be scaled in z axis. All the elements like Hotspots, Reticle for user to know that objects has been interacted, information for users and buttons for interactions were created in world space canvas in this project.

5.3 Interaction with objects

As discussed earlier interacting with objects in VR has different approach than how we interact with our PC or smart devices. For the interaction a script called “VREyeRaycaster” was created which casted a ray from centre of the screen to the world space. We placed this script on the Main camera as we only need one instance

of this script in each scene. The interactive items had the “VRInteractiveItem” script attached to them when these items were hit by ray the “VRInteractiveItem” script was accessed with GetComponent and that script was informed if the ray was still on the object or out from the object. Since this ray was casted from centre of screen and the HMD would change the users rotation of head to the rotation of camera in Unity this insured that the ray was always coming from the centre of person’s field of view.

```
private void EyeRaycast()
{
    // Show the debug ray if required
    if (m_ShowDebugRay)
    {
        Debug.DrawRay(m_Camera.position, m_Camera.forward * m_DebugRayLength, Color.blue, m_DebugRayDuration);
    }

    // Create a ray that points forwards from the camera.
    Ray ray = new Ray(m_Camera.position, m_Camera.forward);
    RaycastHit hit;

    // Do the raycast forweards to see if we hit an interactive item
    if (Physics.Raycast(ray, out hit, m_RayLength, ~m_ExclusionLayers))
    {
        VRInteractiveItem interactible = hit.collider.GetComponent<VRInteractiveItem>(); //attempt to get the VRInteractiveItem on the hit object
        m_CurrentInteractible = interactible;

        // If we hit an interactive item and it's not the same as the last interactive item, then call Over
        if (interactible && interactible != m_LastInteractible)
            interactible.Over();

        // Deactive the last interactive item
        if (interactible != m_LastInteractible)
            DeactiveLastInteractible();

        m_LastInteractible = interactible;

        // Something was hit, set at the hit position.
        if (m_Reticle)
            m_Reticle.SetPosition(hit);

        if (OnRaycasthit != null)
            OnRaycasthit(hit);
    }
    else
    {
        // Nothing was hit, deactive the last interactive item.
        DeactiveLastInteractible();
        m_CurrentInteractible = null;

        // Position the reticle at default distance.
        if (m_Reticle)
            m_Reticle.SetPosition();
    }
}


private void DeactiveLastInteractible()
{
    if (m_LastInteractible == null)
        return;

    m_LastInteractible.Out();
    m_LastInteractible = null;
}
```

Listing 8. Part of VREyeRaycaster script to detect the interactive items

As seen in Listing 8, if the objects had collider and “VRInteractiveItem” scripts attached to them the Over() function was called when ray hit the object and Out() function was called when ray was out from the object.

```
// The below functions are called by the VREyeRaycaster when the appropriate input is detected.
// They in turn call the appropriate events should they have subscribers.
public void Over()
{
    m_IsOver = true;

    if (OnOver != null)
        OnOver();
}


public void Out()
{
    m_IsOver = false;

    if (OnOut != null)
        OnOut();
}
```

Listing 9. VRInteractiveItem script with Over () and Out () functions

As can be seen from listing 9, VRInteractiveItem script has those required functions and when those functions are called they trigger an event. Any other class that is interested on these events can subscribe the methods to perform particular tasks. On the other hand for the object to perform particular task it needs to subscribe to those events.

An simple example for this can be let's say we want to scale an 3D object by 0.3 Unity unit on x axis when we look at it , we would put the "VRInteractiveItem" script on this object and create another script which subscribes to the OnOver() and OnOut() events in this script. We would create the following script.

```
public class OWNINTERACTABLEITEM : MonoBehaviour {

    public  VRInteractiveItem m_InteractiveItem;
    public GameObject ShowandHide;

    private void OnEnable()
    {
        m_InteractiveItem.OnOver += HandleOver;
        m_InteractiveItem.OnOut += HandleOut;

    }

    private void OnDisable()
    {
        m_InteractiveItem.OnOver -= HandleOver;
        m_InteractiveItem.OnOut -= HandleOut;

    }

    //Handle the Over event
    private void HandleOver()
    {
        transform.localScale += new Vector3(0.3F, 0, 0);

    }

    //Handle the Out event
    private void HandleOut()
    {
        transform.localScale += new Vector3(-0.3F, 0, 0);

    }

}
```

Listing 10. Subscribers for events when item is gazed.

As we can see from listing 10, we put the reference of the VRInteractiveItem on this objects and assigned HandleOver function for OnOver () event and HandleOut function for OnOut () event. Inside those functions we scale up and scale down the x values respectively.

5.4 Response to Interaction

As we saw from listing 10, how the object can react when they have been interacted. In this project we had two ways of letting the users know if the object has been interacted or not. There were two different kinds of object that reacted to the gaze. One of them was the hotspots or an area in the panorama and another were different kinds of buttons. We would be talking briefly about this in upcoming topic.

Reticle

First of all the reticle was created in UI to show if the objects have been interacted or not. For this a circle image was created in certain distance from camera which was filled with white. In the image properties the field method was set to radial 360. And the starting value was set to zero in fill amount variable.

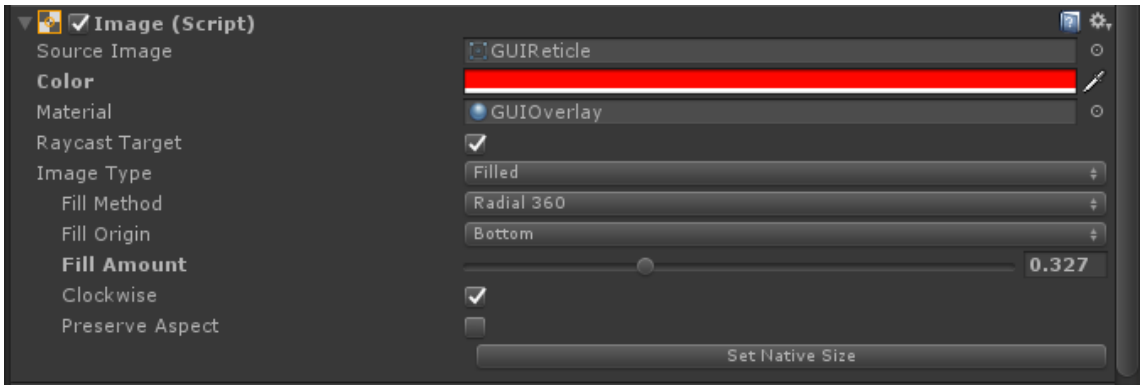


Figure 16. Image property of Reticle

As can be seen from the Figure 16 Image property has Radical 360 fill method, the fill amount can range zero to one and colour for filling was set to red. When the Fill amount is set to 0.32, 0.32 percentage of that image is filled with red as shown in figure 17.

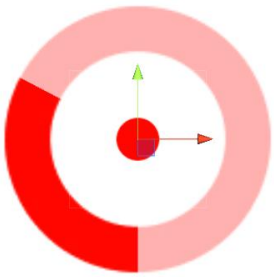


Figure 17. Reticle Image with small fill amount in UI

Now the area for hotspots were created by adding the empty GameObject with box colliders in front of panoramic image to define the boundary and “VRInteractiveItem” script was attached to them, when the user looked at that boundary a small cut off image in the world space canvas was displayed to give an impression that the area was highlighted. See Figure below:

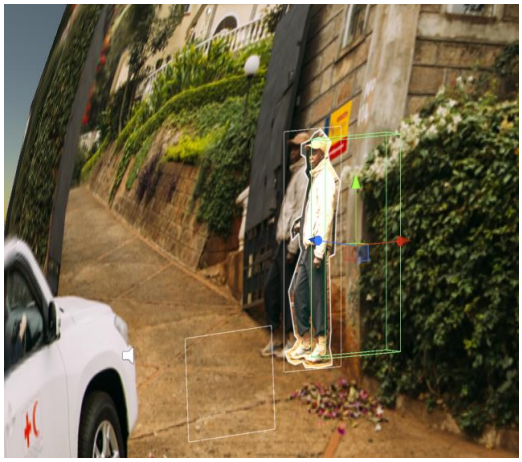


Figure 18. Collider and image cut-off to show user interaction

Besides this, a script was created to handle the filling of the reticle. When a collider was looked by the user, image was highlighted and retile would start filling. This was done by creating the coroutine and after the reticle was filled an event was created to inform that the reticle was filled and filled amount was set back to zero. That particular collider subscribed to that event and handled the visual that the selection of that area has been completed. For displaying and hiding objects Unity's set active method was used which can take bool parameters setting object true or false.

Figure 19. Different stages of Gaze.

As we can see from figure 19, when the person was gazed the area was highlighted by cut-off of that area and when the reticle was filled another image remained displayed to let user to let know that area has already been visited by user. Also the information about that area was displayed to user.

Figure 20. Information about security guard. [Lyfta][1]

The information about that particular area is displayed as shown in figure 20.

Masking

As filling the reticle and highlighting the small area to represent gaze was one of the way of interaction and visualization, the buttons were another type of interactive items. Usually the buttons can be created from Unity directly which are usually used for UI elements for games or other applications. In case of virtual reality we created our own buttons with the filling effect.

To achieve this we used two images and Unity Mask. Common use for using mask is to make visible certain area of an image. The masks do not appear in the user interface but rather reveal the image behind it. For the mask property to work the object to be masked should be child of that parent object which has mask property. We created Canvas and added image on it which acts as a button and is on default image. After that we created a panel object making the filling image the child of the mask.

Figure 21. Button default state and fill state

As shown in Figure 21, right map icon represents the default state and right icon represents the filled state. The width and height of panel object Rect transform were set to zero. As the Rect transform was set to zero the filled state of image was not shown. When the user gazed on the interactive object, in this case being the default image. Method in “MaskEvent” script was called which was attached to default state image and hence ran the coroutine to reveal the image behind the mask.

The script for Masking is demonstrated in Appendix 2, as we can see MaskEvent script has various properties including the RectTransform of the mask. When the desired width and height is achieved in same ratio an event are triggered informing subscribers that the mask has been filled and reset the value of Mask width and height to zero. This script triggers an event to the subscriber who wants to know about the filled stage of image and perform the task. For example, when this button was filled, user interface for analysing the map was displayed as illustrated in Figure 13.

Summary

From the above topics, we understood the basic concepts of interaction and response to the interaction in Virtual Reality. We used this concept of interaction to create the Virtual Reality experience with different buttons and hotspots. Although the environment and the elements in the Virtual Reality were different making it good experience but the main concept used behind their working were concepts such as Raycasting, Events and Delegates, Coroutines.

6 Building and distribution

Being one of the leading platforms for virtual reality technology, Unity supports various ranges of devices you can build your application for. Three different builds of application were done for distribution in different stores. Two versions of android build were made in order to distribute the application in Oculus Store and Google Play, whereas one build was made for Apple Store.

In Unity application can be developed for different devices by going to File and Build Settings Option from the editor. This opens a GUI interface where you can select which device you want to build the application for. Furthermore, different SDK should be selected after switching the platform build in order to make the application run in different stores. The SDK can be included by going to Unity XR settings and are added using plus button.

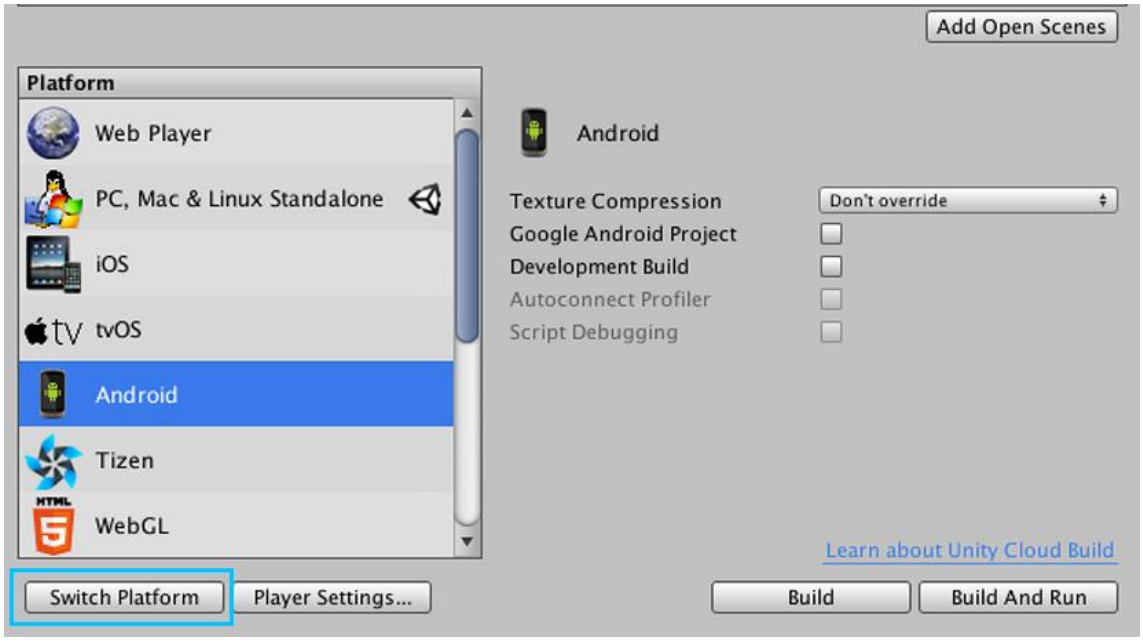


Figure 22. Screenshot of Unity build settings

As we see from figure 22, various range of devices are offered for development of application.

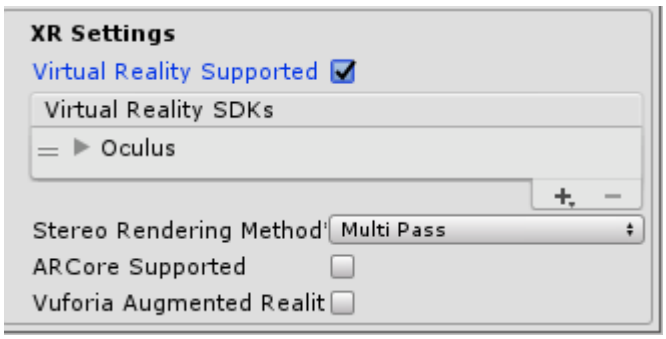


Figure 23. Unity XR settings

As can be seen from figure 23, Oculus SDK is added in order to distribute application in Oculus store and to run the application in GearVR.

Distribution of application in various platforms can be done from the developer console or developer accounts. For distributing application in Google play and oculus store the application was built for android with required distribution settings and app signing. After building the application for android, Unity creates .apk file which can be distributed from the developer account in Google play [17]. Same process is also with developer account in oculus store [18]

For IOS application the Unity creates a project file which can be run through Xcode to generate necessary file for distribution. After creating the account in iTunes the application information can be filled in and uploaded via Xcode. [19]

7 Conclusion

Virtual reality enables us to experience the application in a whole different way. As shown in the project, the development of Virtual Reality requires the knowledge of key points such as how virtual reality works, the different kind of sensors and inputs available for particular devices. It is important to acknowledge and understand these features in order to program correctly and effectively. Furthermore, the limitations that the virtual reality application has on the mobile devices due to the performance of device were discussed. Also, how virtual reality should be optimized to run the project smoothly as investigated.

This application was targeted for different kind of mobile devices and distributed on Google Play, Oculus Store and IOS Store. This provides the basic concept of development of virtual reality application for personal computers also, which enables us to do much more graphically and in the concept of gameplay.

This study suggests that Unity game engine has a good pipeline for Virtual Reality applications. However, the study also found out that different developers might have different opinions and approach for developing VR applications. With the emerging technology in virtual reality we can expect different tools for creating virtual reality applications. Finally, as the devices are getting more powerful in performance each day, we might be able to experience virtual reality such as rendering in real time in near future.

References

1. Lyfta [website]
   https://www.lyfta.com/
   Accessed: March 20 2018

2. Virtual Reality [Online Blogs]
   https://www.vrs.org.uk/virtual-reality/what-is-virtual-reality.html
   Accessed: March 21 2018

3. Virtual Reality [Online Blog]
   https://www.digitaltrends.com/cool-tech/history-of-virtual-reality/
   Accessed: March 21 2018

4. Virtual Reality [Online Blog]
   http://www.realitytechnologies.com/virtual-reality
   Accessed: March 22 2018

5. Unity. Manual [Online]

   https://docs.unity3d.com/Manual/UsingTheEditor.html
   Accessed: October 1 2017

6. Unity Blogs. Different Programming Language for Unity [Online]

   https://blogs.unity3d.com/2014/09/03/documentation-unity-scripting-languages-and-you/
   Accessed: October 5, 2017

7. Microsoft Documents. C# programming
   https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework
   Accessed: October 11, 2017

8. Unity Website [Online]
https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html
Accessed: November 1, 2017

9. YouTube Channel [Online]
https://www.youtube.com/watch?v=hL3ZdNNmMx0
Accessed: December 10, 2017

10. Online Blog
http://www.unitygeek.com/delegates-events-unity/
Accessed: March 24, 2018

11 Unity Website [Online]
https://docs.unity3d.com/Manual/Coroutines.html
Accessed: March 26, 2018

12 Unity manual [Online]
https://docs.unity3d.com/Manual/Shaders.html
Accessed: March 29, 2018

13 Online Video [YouTube]
https://www.youtube.com/watch?v=HEHn4EUUyBk
Accessed: March 29, 2018

14 Wikipedia [Online]
https://en.wikipedia.org/wiki/User_interface
Accessed: March 31, 2018

15 Unity documentation [Online]
https://docs.unity3d.com/Manual/UICanvas.html
Accessed: April 1, 2018

16 Unity documentation [Online]
https://docs.unity3d.com/Manual/script-Mask.html
Accessed: April 9, 2018

17 Google Play [Online]

https://developer.android.com/distribute/index.html
Accessed: April 17, 2018

18 Oculus Store [Online]
https://developer.oculus.com/distribute/
Accessed: April 17, 2018

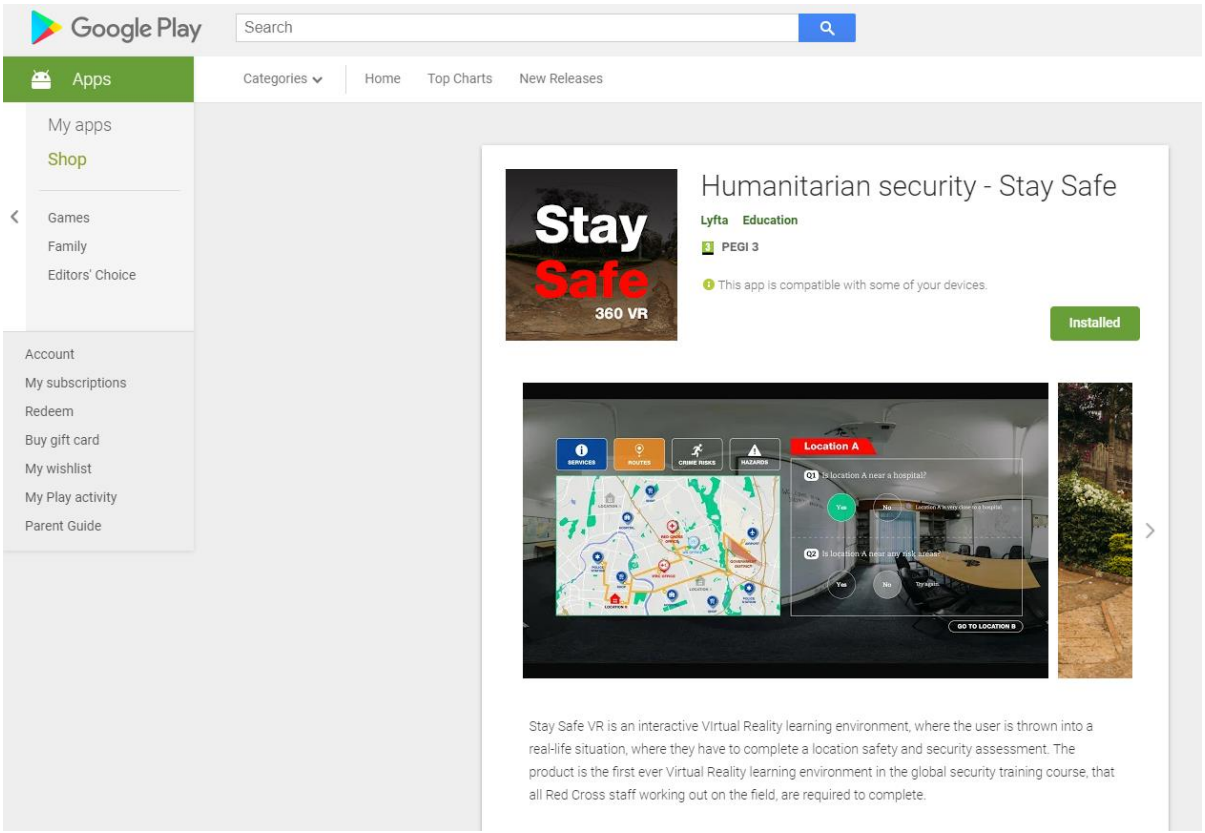
19 Apple distribution [Online]
https://developer.apple.com/distribute/
Accessed: April 18, 2018

Appendix 1

Appendix 1: Screenshot of Application in different stores

Google Play

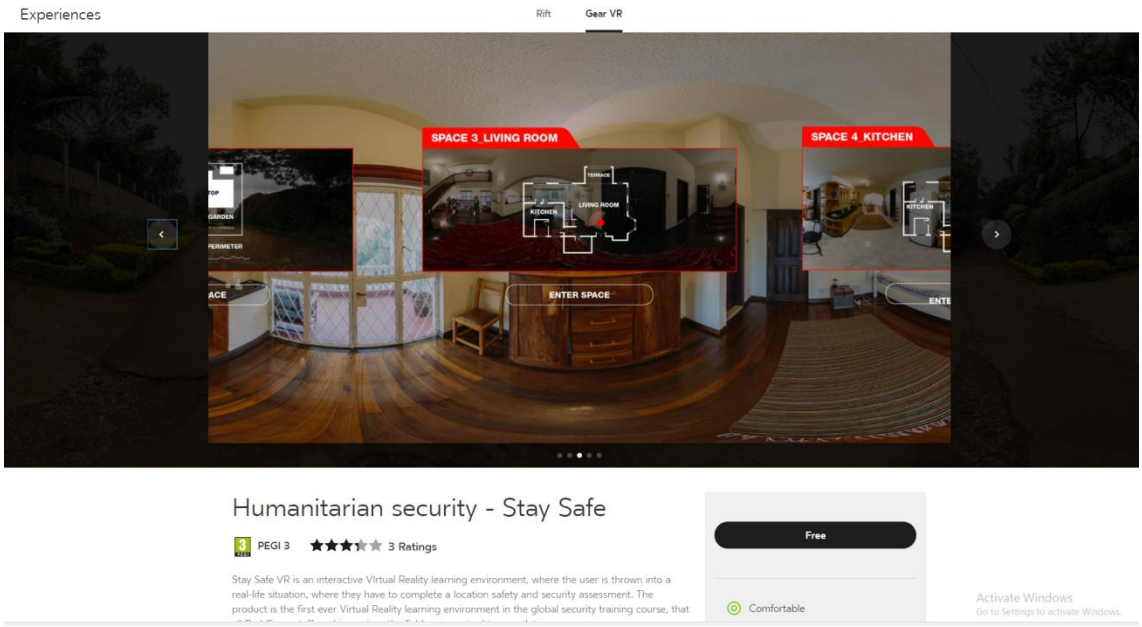
Link to application



.

Oculus Store

Link to application



Appendix 2

Appendix 2: Script for filling image by Masking

Mask Event Script

```
public class MaskEvent : MonoBehaviour {
    public event Action OnMaskFilled;

    public RectTransform masktransform;
    public float maskWidth;
    public float maskHeight;
    public float durationOfFill;
    private float timer = 0.0f;
    private float ratio;
    private bool isFilled = false;
    private Coroutine ExpandRoutine;
    private Coroutine ShrinkRoutine;

    void Start()
    {
        masktransform.sizeDelta = new Vector2 (0.0f, 0.0f);
    }
    public void StartFill()
    {
        ExpandRoutine = StartCoroutine (ButtonExpand ());
        if (ShrinkRoutine != null && masktransform!=null)
        {
            StopCoroutine (ShrinkRoutine);
        }
    }

    public void ReverseFill()
    {
        ShrinkRoutine = StartCoroutine (ButtonShrink());
        if (ExpandRoutine != null && masktransform!=null)
        {
            StopCoroutine (ExpandRoutine);
        }
    }

    private IEnumerator ButtonExpand()
    {
        while (timer < durationOfFill)
        {
            ratio = (timer / durationOfFill);
            masktransform.sizeDelta = new Vector2 (maskWidth, maskHeight) * ratio;
            timer += Time.deltaTime;
            yield return null;
        }
        isFilled = true;
        if (OnMaskFilled != null)
            OnMaskFilled ();
    }

    private IEnumerator ButtonShrink()
    {
        while (timer > 0.0f && isFilled == false)
        {
            timer -= Time.deltaTime;
            ratio = (timer / durationOfFill);
            masktransform.sizeDelta = new Vector2 (maskWidth , maskHeight) * ratio;
            yield return null;
        }
    }

    public void ResetValues()
    {
        masktransform.sizeDelta = new Vector2 (0.0f, 0.0f);
        ratio = 0.0f;
        timer = 0.0f;
        isFilled = false;
    }
}
```