Goldsmiths
UNIVERSITY OF LONDON

FINAL YEAR PROJECT

Virtual Reality and Its Effectiveness for Education

Author:
Butrint TERMKOLLI

Supervisor:
Dr. Daniel BUCHAN

A thesis submitted in fulfillment of the requirements for BSc Computer Science Degree

January 31, 2023

Declaration of Authorship

I, Butrint TERMKOLLI, declare that this thesis titled, "Virtual Reality and Its Effectiveness for Education" and the work presented in it are my own. I confirm that:

- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Signed:

Date: 13th August 2020

UNIVERSITY OF LONDON

Abstract

Computing Department

BSc Computer Science Degree

Virtual Reality and Its Effectiveness for Education

by Butrint TERMKOLLI

This project explores the effectiveness of using Virtual Reality (VR) as a tool for educating young people. The aim is to inform of dangers of drug/substance abuse with the use of an immersive environment which simulates the side effects of narcotics. The project makes use of Unity and modern VR technology to develop and create a virtual environment and incorporates the use of C# to generate the VR Experience. Results from a small focus group show that the overall product is successful, and that the virtual experience is an effective educational tool. Current research has supported the use of VR technology for pedagogic intentions, however this project has demonstrated the prospect of using VR to support drug education specifically.

Acknowledgements

Massive thanks to my supervisor Daniel Buchan who greatly ameliorated the initial sparks of this idea.

Also a huge thanks to Brikena, Gentiana and Tahmid for believing in me and giving me assistance with LaTeX when it was most greatly needed and appreciated.

Contents

Chapter 1

Introduction

1.1 Motivation and Aims

Social media and the internet have undoubtedly made the world appear to be smaller, notably due to widespread accessibility and universal connection [1]. Many businesses now solely operate using online platforms, which also includes businesses associated with licit and illicit substances [2]. The impersonal nature of attaining drugs in this manner has consequently led to the vast consumption and purchase of narcotics, despite a fundamental lack of awareness and education regarding substance abuse [3].

Western societies in particular are characterised by young people who have succumbed to social circumstance and peer pressure to engage in drug use [4]. It has become increasingly important to acknowledge that although it would be deemed ineffective to try and cease drug usage entirely, the education of the young would provide an effective deterrent. This project will seek to create a new tool, a Virtual Reality (VR) experience, which allows for a realistic and immersive educational experience.

Research has considerably indicated that VR is an effective instrument which can be utilised to inform and educate, suggesting it may perhaps prove substantially more successful than the current implemented methods [5]. At present there are no existing VR simulations which act as an educative resource on drug use and misuse.

This project will aim to provide an evaluation on the advantages of educating via Virtual Simulation: a new tool that can aid and benefit the way in which we teach. A new, more interactive form of learning can better educate people on the consequences of licit and illicit substance abuse.

1.2 Overview of Report

The remaining chapters of this report are outlined in the structure below;

Chapter Two - *Background Research* – This chapter will discuss Virtual Reality's usage throughout history, highlighting its current uses within the workplace. Methods of learning will also be explored within this chapter, with an overview of existing prevention programs in place concerning drug abuse.

Chapter Three - *Design* – Chapter Three will focus on the design process, with a detailed synopsis of the main technology and components required to build the project.

Functionality requirements are detailed within this section, including flow-charts to see how unique areas of the project will perform. Prototypes are also presented to show the initial scope of the project.

Chapter Four - *Implementation* – This chapter details how the final project was developed through software implementation with the use of Unity and C#. Code snippets give evidence to the unique functions that helped develop the project.

Chapter Five - *Testing* – This chapter will refer to the functionality requirements previously developed; with the use of different testing methods to ensure that these requirements were met throughout and after the projects development.

Chapter Six - *Conclusion* – The project is concluded within this chapter, detailing the overall successes and failures, with additional insight into further implementations that would benefit the project's effectiveness in the future.

Chapter 2

Background Research

This chapter presents the research examined and conducted before commencing with the development of the project.

2.1 Motivation and Justification

As drug use is more prevalent in the younger population [6], this VR game will therefore be aimed directly at this age group. This tool has the potential to educate within schools and universities to inform young people of the severe consequences of narcotic consumption. The main motivation for devising this project was the insufficient amount of interactive and fun experiences being utilised to inform young adults of the extensive repercussions and effects of different drugs.

2.2 Virtual Reality

Virtual Reality is a computer-generated simulation that provides a user with a sensory, 3-D environment in which they can interact with their surroundings. Digital experience has been transformed by this ability to create engaging illusions in which any desired setting can be created.

2.2.1 History and Current Uses

The earliest conception of VR can be traced back to the 19th century; the notion of immersing ourselves within mirages emanates from historic fascination with stereoscopes [7]. If you produce similar images to each eye, our mind can merge them and find depth in their differences. The very first VR device is believed to be cinematographer Morton Heilig's non-computerised 'Sensorama' in 1957 [8]. His arcade-style booth revealed stereoscopic 3D film blended with audio, aromas and breeze. A spinning chair allowed the customer limited control over the experience. This prototype is highly restrictive in comparison to today's VR technology.

Today, VR is used within the military, in sport, in medical training, as a hobby and most relevantly, to educate [9].

In a teaching environment, students are able to communicate with one another within a three-dimensional digital setting. It has also proven to be especially advantageous for students with Autism [10]. Research has demonstrated that the use of VR provided children with a motivating and safe space to exercise their social abilities [10].

Technology group, Floreo, have designed VR scenes that grant children the opportunity to learn and exercise abilities that include making eye contact and social interaction.

Fictional ideologies

The notion of VR is believed to have entered public consciousness as a result of Stanley G. Weinbaum's 1935 novel, 'Pygmalion's Spectacles' which introduced the possibility and fantasy of "instead of being on a screen, the story is all about you, and you are in it." [11] His vision of VR centred around a goggle-type device that immersed the user into a holographic adventure. Weinbaum's concept of VR has undoubtedly influenced the present uses of the simulation device.

Current Fundamentals

Ivan Sutherland is widely regarded as the first individual to devise a VR display system that is worn as a headpiece. He established the idea of a lifelike computer-generated world that could be viewed through a head-mounted machine, enabling the user to realistically interact with their digital surrounding. He named this concept 'Ultimate Display' with the goal of producing a simulated reality so naturalistic, it could not be distinguished from reality.

Ivan believed it 'would, of course be a room within which the computer can control the existence of matter.' His 'Sword of Damocles' prototype was pivotal in prompting innovation for the age that followed. His research is considered to be the foundation of today's VR technology. [12]

Commercial use

What was once science-fiction fantasy, has remarkably developed into a more affordable, accessible, domestic VR that can be experienced at home. Companies have embraced VR technology and the many possibilities it can provide [13]. Commercial spending on VR is nearing $9.2 billion by 2021 [14]. VR machinery is also widely exercised in training procedures, allowing trainees to fully experience all types of scenarios simulated. It is extremely beneficial for trainees as they are able to practice interaction which enhances confidence and service performance.

The use of VR commercially has additionally spared many companies a fortune [15]. Businesses now have the ability to explore unfamiliar ideas using virtual reality, as opposed to spending excessive amounts on functioning test examples. Aircraft design, for example, utilises computer-generated simulation in order to trial brand new features [16].

2.2.2 Strengths and Limitations

In this section I'll be reviewing Virtual Reality: the strengths and limitations that current hardware has in creating a realistic experience.

Illusions

As VR is a form of perceptual illusion, a limitation is that current home VR Kits are only capable of displaying two senses at most; visual and/or auditory, with an addition of minimal touch/haptics that barely imitate real-life. Increasing the amount

of senses that a VR simulator simulates, will enhance a user's enjoyment and 'sense of presence' [17]. Also, with the current progression of technology, multisensory VR will soon be commercially available for purchase. This creates the question: if all one's senses were connected to the equipment, to what extent could one differentiate between reality and a virtual world? [17]

R.L. Gregory's [18] study on perceptual illusions indicates that by deceiving our brains, one can process information that although resembles reality, does not match the true image. Gregory's demonstrates that pieces of renowned art such as the Hollow-Face illusion and Oscar Reutersvärd's Penrose triangle, are effective illusions that use visual perception to 'trick' the mind. Thus, using two senses in VR can be sufficient in creating a powerful experience.

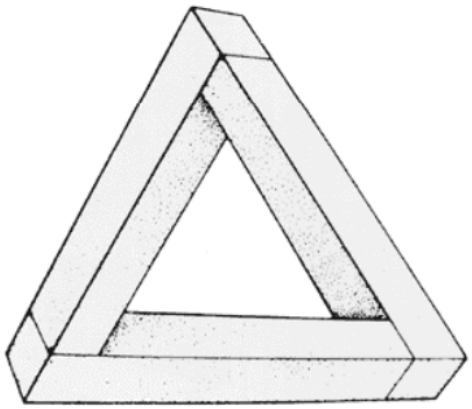

FIGURE 2.1: Oscar Reutersvärd's Penrose Triangle

Learning Styles

Considering a neurological approach to learning styles, VR can be embraced by both hemisphere-dominant learners [19]. The two hemispheres of the brain are associated to different processing functions. This allows for dominance of one area, which varies from person to person. Right hemisphere dominant learners are much more self-centred and creative; their learning proficiency is tied to visual and tactile symbols and require a more abstract and hands-on approach when learning and memorising information. Left hemisphere dominants are more competent with using language and numbers. Left hemisphere dominance is also tied in with logic and critical thinking.

Although education institutes encourage the use of both hemispheres, the current implemented education system is heavily based on using language and numbers for memory, which causes right-hemisphere dominants to struggle [20]. Ideally, both styles of information should be presented in an academic setting [21]. This ensures that a student's preferred style of learning will be attained, while also representing other students that have unique learning styles. This would allow the incorporation and extension of different learning processes for all students, when presented with the same material. Educators could also link different styles together when presented with a different medium. As VR is effective at simultaneously presenting information, both visual and analytical, this theory of processing style has been considered when creating this project.

2.3 Drugs within Youth

In this section, the current issues of drug abuse and specifically available education in schools/higher education in regards to drug use will be highlighted.

2.3.1 Licit drugs

Alcohol is typically consumed during teenage years in Western cultures, despite not reaching the legal drinking age [22]. Surveys have indicated that a significant majority of 15-16 year olds in the UK have been intoxicated from drinking with many regularly doing so [23]. The introduction of prevention programmes in schools was initiated in response to the large proportion of teenagers that drink alcohol. The aim of these is to educate young people in order to minimise health and social risks associated with intoxication and alcohol abuse. Despite this, evidence of success of these intervention strategies has been slight [24].

2.3.2 Current prevention programs

With the 21st Century being renowned as the psychedelic renaissance [25], reviewing current school-based drug prevention programs was necessary to understand the possible effectiveness of the project [26]. Although there is a large expansion of prevention programs, a majority of so are split into three separate categories. Peer teaching and learning has been used widely to introduce a collaborative approach [27]; whereas some teach resilient mechanisms as a preventative measure [28]. Lastly, others take a more passive approach to educating [29]. A large proportion of prevention programs are conducted within the grounds of primary education.

Soole's [26] review highlighted the significance of conducting educational programs targeting legal drug use as well as illegal substances to ensure effective prevention. The research also showed that interactive schemes were most impactful in drug education.

2.4 Existing Software

Within the market there are many games and simulations that deliver similar messages that this project will look to compete with. It is important to research pre-existing software to avoid developing a generic product. Many of the current software designs, although are informative and educational' do not incorporate the use of Virtual Reality. This limits the amount of engagement in these software's to being just behind a screen, which is something this project aims to develop further.

2.4.1 Drug Simulator

Drug simulator V.1.0 is a game developed by ChoppyPine. This simulator has 6 available drugs simulations. Each simulation is unique and made using scientific evidence. Although extremely informative, the game is designed in Unreal Engine and is a simple 3D game. The limited layering of the game does not allow for a realistic experience and thus lacks complexity [30].

2.4.2 3D Organon

3D Organon has proven to be an incredible tool in the medical word, with the simulation of the human body. With over 4,000 structures and organs, its detailed model of a human anatomy allows us to observe the countless uses of VR. One issue however, is the lack of in-depth information one is able to obtain while in the 'hands on' mode. Organs have basic labels, and no information is given about the organs job or possible clinical issues unless one exits into another scene that doesn't include the anatomy. Incorporating all information together is an important design feature, which 3D Organon lacks [31].

2.5 Public Survey

A short survey was conducted on Google Forms to gain an insight into opinions surrounding drug awareness as well as gauging the public's familiarity with Virtual Reality technology. The survey was also supported with face-to-face questioning in order to attain deeper knowledge of drug culture perceptions in the age group. Thus both qualitative and quantitative data was collected to shape and enrich the virtual project.

2.5.1 Results

Question 1: Do you find learning through games/activities useful?

Question 1 looks to fortify Sperry's theory [19], with an almost equal distribution between participants who find activities and games useful for education and those which don't. This instils the idea of unique learning styles; as school curriculum is the largely the same for all students in the UK, their preferred method of retrieving and storing information was not considered.

Question 2: Have you ever used a Virtual reality headset before?

Upon reviewing the results from Question 2, it is clear to see that most people have never used a Virtual Reality headset before. This is an expected result, as the VR kit is not an easily accessible and affordable item for most households.

Question 3: Were you taught about drugs during your time in education?

Question 3 received similar outcomes, with 71.4% of participants not recalling having any drug education in the past. After speaking to a Primary School Teacher, it was highlighted that drug education is not generally taught, as parents fear that they might be glorified.

Question 4: Is there substantial drug use within your age/friendship group?

Question 4 aimed to understand whether drug use within friendship groups was apparent. 71.4% of participants highlighted that there was a substantial use, with an equal 14.3% mentioning that they either didn't know, or drug use wasn't specified within their conversations.

Question 5: Do you think drugs are dangerous?

Lastly, an optional open-ended question was asked to gather some qualitative data. A majority of participants agreed on the dangers of drug use, with many detailing their reasoning behind past experiences, rather than the scarce information that was provided to them through educational facilities.

2.6 Project Aims

The focal aim of this project is to develop a virtual experience in which users are able to educate themselves about the dangers of drugs. This is targeted at a younger audience, with the importance of education particularly stressed during a period whereby 'experimenting' may be customary [32].

The virtual experience will contain two main parts, the environmental development (Created in Unity) as well as the side-effects (Developed with C# in Windows Visual Studio).

The environment will be able to interact with the user's actions, with objects that symbolise drugs that can be pointed at to reveal information, or grabbed and ingested, similarly to reality. The game will be able to recognise when objects are picked up and will use range parameters to identify when an object has been ingested.

Development within Microsoft Visual Studio will explore Post-Processing, creating unique on-screen effects that will mimic real life drugs.

Chapter 3

Design

The formulation process regarding the design of this project was essential in ensuring a successful outcome. This section will provide a deeper insight into the resources and techniques used to advance this project and the reasoning behind the approach and response to the research statement. Crucial criteria were established to develop the basis of the project's performance during the trial stage.

3.1 Functionality Requirements

	Requirements
i	Grabbable feature for object
ii	Grabber feature for controllers
iii	Able to interact with drugs
iv	Ability to experience side effects with Post-processing
v	Different learning styles
vi	Drug-specified sound effects

TABLE 3.1: Table detailing the functionality requirements created for the project to succeed.

(i, ii & iii): In order for this experience to be accessible and function at a basic level, the first three points had to be satisfied fully. At minimum, objects had to be grabbable using the *grabbable* script given via Unity. This was done by setting *rigidbody* and *gravity* to the object to ensure that it reacted correctly with movement. Similarly, with *grabber*; OVRController's hands were required to have a grabber script included. This allows the virtual hands to react with the scene's objects, allowing the user to pick them up and move them. The drug(s) must be able to be picked up as well as interact with the user's face: to mimic swallowing/inhaling. This ensures that the project is as realistic as possible – having a button or a different trigger function would be a false representation of a real scenario and thus less effective.

(iv): This allows the user to experience the side effects of the drug that they interact with. This will greatly enhance the realism of my project; taking drugs with no effect on the user's experience would not achieve the same reception.

(v): In order to satisfy this requirement, the project will include a UI Panel that allows interaction by pressing the drug. In line with background research into unique learning styles, it is important that both a right-hemisphere dominant technique as well as a left-hemisphere dominant technique are included. The right-brain dominants' required creativity learning style will be shown through the post-processing

effects (iv); with a left-brain dominant being able to use UI Panels to assist their methodical approach to learning.

(vi): Sound functionality will ensure that this project retains an adequate amount of realism. Drug-specified effects will allow the user to experience more than one sense (visual), with research into illusion effectiveness showing that using more senses will increase the realism that a VR project can display.

3.2 Unity

Unity is a renowned 3D development platform which allows and supports the creation of a range of experiences and games that are accessible on different devices. Unity's ease of use compared to Unreal Engine 4 ensures the suitability for the scope of this project.

3.2.1 C#

C# is an object-oriented programming language. As Unity only supports the use of C# and Javascript, it is the favourable programming language to use for this project. It allows for the breakdown of a problem into smaller parts; the structured approach simplifies the process of using an object-oriented programme. It also has a rich set of library functions and data files, and its language code does not require header files: code is written inline.

3.2.2 High Definition Render Pipeline (HDRP)

Unity features' HDRP within their VR building kit. HDRP is an effective tool for delivering photorealistic visuals and qualities that other competitors lack. The visual qualities role in developing the experience is incredibly important, as it is imperative that a user must feel like they are having a realistic experience. Poor graphics would thus ruin the immersion.

HDRP takes advantage of a GPUs (graphics processing units) power, something that was previously performed by the less powerful CPU (central processing unit). Using a GPU allows for processes such as tessellation to be performed. The use of a GPU saves a computer's memory and bandwidth, which in turn increases efficiency.

Without this feature, realistic visual effects would require a third-party software – limiting the design features due to time constraints or lack of third part compatibility.

3.2.3 Importance of Inertia

The tendency to resist changes in their state of motion is described as inertia. Einstein described this as:

$$F = m\,a$$

Where 'F' is the Inertia, 'm' is the mass of the object, and 'a' is the acceleration of that object. In 3-D mechanics, rotating an object around an axis will require an amount of force dependent on the object's mass. This can be described as:

$$R = W\,I$$

Where 'R' is the rotational force, 'W' is the angular velocity and 'I' is the moment of inertia. An example of this is: if an object were to spin on the x-axis and a force was applied on the y-axis, the object would not switch axis immediately but will instead use both forces applied, to spin in an appropriate foreign direction.

Unity is most effective at displaying inertia tensor, with its own plugin: *RigidBody*. This creates identical force application to objects in real life – precision that is key in creating an educational platform by adding to the realism.

3.2.4 Oculus Integration Package

The Oculus Integration package, *OculusIntegration.unitypackage*, includes a variety of necessary features to assist developers with building Oculus based Virtual Reality within Unity. A wide variety of software development kits (SDK's) are bundled within this package ensuring flawless integration. This is something that Unity's direct competitor, Unreal Engine 4 (UE4), does not include. UE4 is only capable of generic built in support for all virtual reality hardware, with Oculus hardware requiring further manual integration to be fully supported.

Prefabs

Prefabs are incredibly useful for creating projects with Unity. These are reusable objects, such as shapes, light or camera's that are part of the Oculus Integration Package. These prefabs are quite simple to use, with the ability to drag and drop them into scenes from the *Assets/Oculus/VR/Prefabs/* folder.

OVRCameraRig

The OVRCameraRig is a custom VR camera which is optimised to render stereoscopic display on an Oculus device. An interface is also provided through OVRManager. This is a necessity for the design as the hardware being used is an Oculus Rift S. Therefore, Unity's Oculus package ensures that the software's display is optimised to its best quality. This feature is lacking among other game engines, such as UE4 and GameMaker.

OVRPlayerController

The OVRPlayerController enables the user to navigate in the virtual environment, which provides child objects and appropriate apparatus that are needed for 3D control. This is incredibly useful, as using this prefab means that the Rift S' controllers are already setup for implementation, making it simple to access them in scripts.

3.3 Prototyping

3.3.1 Low-Fidelity Prototypes

The main scene was initially designed on paper wireframes in order to gain an insight and understanding into the creation before it was attempted. There were many wireframe ideas, however, drug use and abuse amongst young people in clubs was

a prominent theme that was raised in the qualitative data. For this reason, a public restroom was the most suitable environment to introduce into this study.

3.3.2 High-Fidelity Prototypes

Before finalising any model design choices, simple cubes and circles were used as objects so as to not increase compiling time while experimenting. This was conducted to test many different versions of an object's interaction, before cloning its components onto the final model.

3.3.3 Asset Store

The asset store is useful tool that will be used in the designing process. Objects will need to simulate and represent real-life items: the prototype forms were unable to achieve this level of realism. For this reason, the asset store will be used to purchase a wide variety of models to use in the experience.

3.3.4 Postprocessing

Postprocessing is an important feature in Unity which will be the basis of creating all the visual side effects and provide the element of realism. This is done by initially rendering in the memory of the video card. Shaders are then used to apply post-processing filters to the image buffer before displaying the image to the device. Through the alteration of the rendering effect and thus its perception, side effects that mirror real-life ones were able to be simulated.

3.4 Script Flow Charts

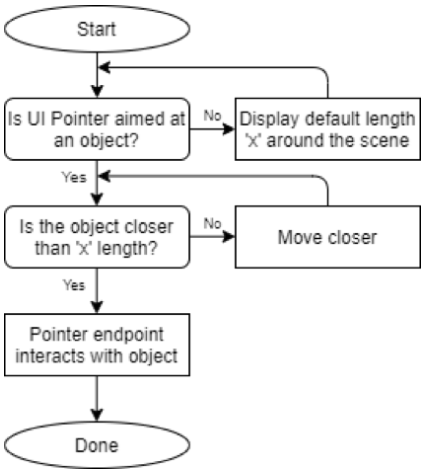

FIGURE 3.1: Flow chart used to better understand the function of the pointer, in order for the project to excel.

This chart shows the interaction capabilities that the physics pointer will include: x is an undetermined length that will be used for the pointer.

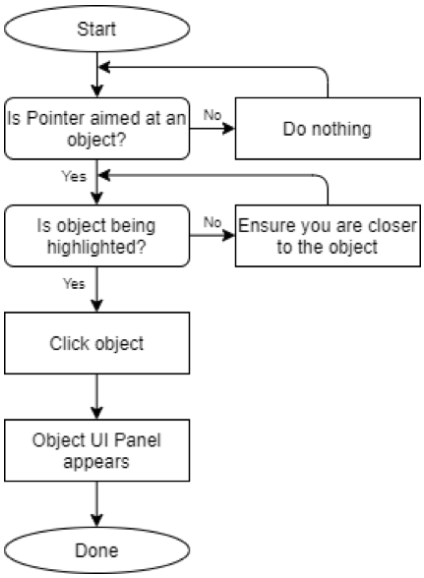

FIGURE 3.2: Flow chart detailing the stages to open a UI Panel in this project.

This diagram above assumes the physics pointer has already been applied, and uses it to point at objects. Button presses made by the user activate a panel which is appropriate to the object that is highlighted. A limitation to this script is the physics pointers' length, however, the scene's size was not large enough to cause a significant issue.

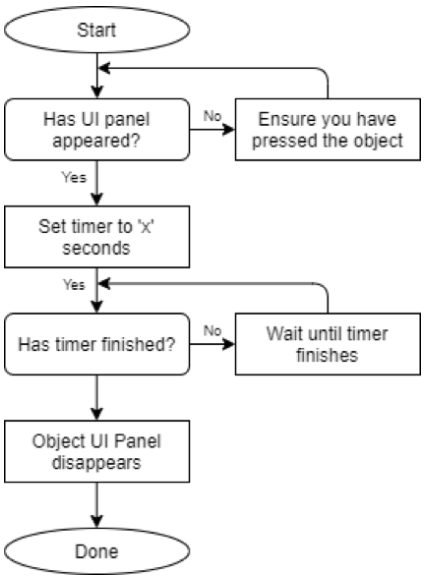

FIGURE 3.3: Flow chart detailing how panel timer will be executed.

Assuming the UI panel can be activated using the physics pointer, this script will be a timer for the panel. The timer allows less 'button-clicking' for the user. This makes it easier to use, and the timer will also be appropriate for how many words there are to read i.e.; x in this flow chart will not be 5 seconds if the text doesn't take 5 seconds to read.

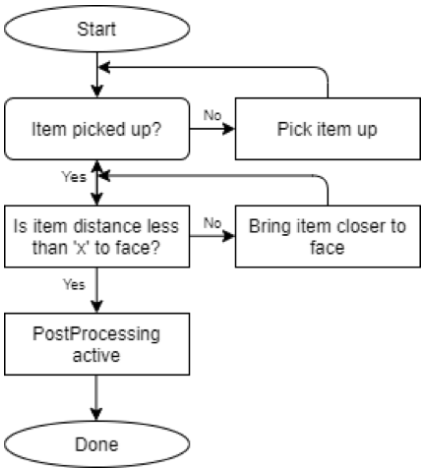

FIGURE 3.4: Flow chart of how item interaction will work.

The Item will be picked up, and when brought 'x' distance close to the face, will activate the object i.e.; drug - will activate the post processing filter.

Chapter 4

Implementation

This chapter describes the implementation of design choices with C# into Unity, particularly focusing on the areas of development that were complex and interesting. The methods used and their overall impact on the wider project will be outlined.

4.1 Distance Script

The distance script is an integral part of the project. It is used to compare the distance between two objects, using an if statement to alter the objects as they get closer to each other. The below listings show the distance script designed for the project, adapted from resources online [33].

```
// public class Distance : MonoBehaviour
{
    // Use GameObject so we can alter which objects to use within Unity
    public GameObject Object1;
    public GameObject Object2;
    public float distance_;
```

CODE SNIPPET 4.1:
Distance Script Snippet 1

GameObject creates a new object, which can be set using the Unity inspector. In this script's case, the two objects *Object1* and *Object2* were named as such to easily distinguish between the two. The objects' models would be changing throughout the implementation process.

```
// Update is called once per frame
void Update() {
// Tracks the distance between object 1 and object 2
        distance_ = Vector3.Distance(Object1.transform.position,
            Object2.transform.position);
// if statement which tells us if the object is less than the given length,
    activate the postprocessing. Distance is altered to mimic putting drug
    to mouth
        if (distance_ < 0.25)
        {
            // Log update to verify if statement works
            Debug.Log("Drug has been taken");
            // Enable postprocessing effect
```

```
            GameObject.Find("CenterEyeAnchor").GetComponent<PostProcessLayer>
            ().enabled = true;
        }
    }
        }
    }
```

CODE SNIPPET 4.2:
Distance Script Snippet 2

The float variable *distance_* is next initialised to track the distance between the two separate objects.

Before adding postprocessing effects, two cube objects were created with the following *if* statement:

```
if (distance_ < 0.25) {

            // Log update to verify if statement works

            Debug.Log("Drug has been taken");
}
```

CODE SNIPPET 4.3:
Distance Script Snippet 3

Debug.log was used to display a message in Unity's log. This is an effective way to acknowledge whether a statement has been implemented correctly. A simple line of code was also incorporated that adds a visual representation to how the distance script would work:

```
// Turn the cube yellow

gameObject.GetComponent<Renderer> ().material.color = Color.yellow;
}
```

CODE SNIPPET 4.4:
Distance Script Snippet 4

This line of code was initially added into the *if* statement and would simply turn one of the cubes yellow when the distance had been shortened to less than 0.25 metres.

The distance script is overall quite simple, however is an integral part of the entire systems design. Debug logs and the use of a shader creates easier opportunities to identify any bugs during testing phases.

4.2 VR Input

The VR Input script creates a pointer to act similar to a 2D cursor - using the Oculus Rift S' controller to be able to move and click with it. Tutorials online assisted me with the creation of this script [34].

```
using UnityEngine.EventSystems;

 public class VRInput : BaseInput
{
    // Dummy Camera attached to the controller, to mimic a 3D Cursor

    public Camera eventCamera = null;
```

CODE SNIPPET 4.5:
VR Input Script Snippet 1

Unity's *EventSystems* revolve around depending on an 'Event Camera' to know where the cursor is on the screen. Due to VR not having a cursor, a dummy camera must be used instead. This camera is attached to the Rift S controller in-game and will have its constant midpoint taken to mimic a cursor.

```
public override Vector2 mousePosition
    {
        get
        {
            // This ensures the cursor comes from the centre of the
                controller, /2 on both axis
            return new Vector2(eventCamera.pixelWidth / 2,
                eventCamera.pixelHeight / 2);
        }
    }
```

CODE SNIPPET 4.6:
VR Input Script Snippet 2

This function shows the mouse position. The pixel width and height are divided by two, as that will ensure that the cursor is in the centre of the *eventCamera*'s screen.

```
    // OVR Inputs for button

    public OVRInput.Button clickButton = OVRInput.Button.PrimaryIndexTrigger;

    // OVR Input to check which controller to use; set to all to choose in
        unity

    public OVRInput.Controller controller = OVRInput.Controller.All;
```

CODE SNIPPET 4.7:
VR Input Script Snippet 3

Using the OVR Library [35], *OVRInput*'s are implemented for both the button and controller. A *clickButton* variable is set to activate with the Primary Index Trigger, as it is the most accessible button to use on the controller. The controller used was set to *All* to accommodate both Left and Right-hand dominant users.

```
  protected override void Awake()
{
    // Script inherits from BaseInput
    GetComponent<BaseInputModule>().inputOverride = this;
}
```

CODE SNIPPET 4.8:
VR Input Script Snippet 4

Upon commencing, *Awake()* is used to initialise any variables or game state. This *Awake()* function inherits from the *BaseInputModule* script found in: *ui / UnityEngine.UI / EventSystem / InputModules / BaseInputModule.cs*.

The *inputOverride* variable within *BaseInputModule.cs* controls the input you want to use e.g. if you want to set this to a key press such as the spacebar, or a mouse click – it is possible. However, in order to replace this input with a button from the controller, it was set to *this*. This is possible as the script inherits from *BaseInput*.

```
public override bool GetMouseButton(int button)
{
    // Gets the value of whatever button you have on your controller
    return OVRInput.Get(clickButton, controller);
}

public override bool GetMouseButtonDown(int button)
{
    // Gets the value of whatever button you have pressed
    return OVRInput.GetDown(clickButton, controller);
}

public override bool GetMouseButtonUp(int button)
{
    // Gets the value of whatever button you have recently lifted
    return OVRInput.GetUp(clickButton, controller);
}
```

CODE SNIPPET 4.9:
VR Input Script Snippet 5

These three functions are used to track a button's action. They all work relatively similarly. *GetMouseButton*: receives the value of whichever button one is using, *GetMouseButtonDown*: receives the value of whatever button has been pressed, and *GetMouseButtonUp*: receives the value of whatever buttons has been lifted.

4.3 Physics Pointer Script

This script was used to create the pointer that would interact with the drugs/objects. Resources online assisted me with the development of the pointer [34].

```
   public class PhysicsPointer : MonoBehaviour
{
    // Default length, set as 3.0f as it was a length that wasn't too long
    public float defaultLength = 3.0f;

    // Takes two points in 3D space, and draws a line between the two
    private LineRenderer lineRenderer = null;
```

CODE SNIPPET 4.10:
Physics Pointer Script Snippet 1

A float value, *defaultLength* was initially created. This was done to create a base for the pointer, so that if it was not being obstructed by any objects or environment, it would retain this original length. The length was tested a few times to ensure that the pointer was not too short so that the user would have to be extremely close to the object to interact with it, and not too far that it would obstruct the users view.

```
public class PhysicsPointer : MonoBehaviour
{
    // Default length, set as 3.0f as it was a length that wasn't too long
    public float defaultLength = 3.0f;

    // Takes two points in 3D space, and draws a line between the two
    private LineRenderer lineRenderer = null;
```

CODE SNIPPET 4.11:
Physics Pointer Script Snippet 2: lineRenderer initialisation for two points

A Line renderer, *lineRenderer* was also created. This was an essential part of the pointer; it takes the x, y and z coordinates from 2 separate points, and draws a line between the two. In this case, the two separate points would be the controller itself and wherever the controllers ray was aiming towards.

```
private Vector3 CalculateEnd()
    {
        // Create two separate final points
        RaycastHit hit = CreateForwardRaycast();

        Vector3 endPosition = DefaultEnd(defaultLength);
```

CODE SNIPPET 4.12:
Physics Pointer Script Snippet 3

The beginning of the *CalculateEnd()* function shows two initialised variables, *hit* and *endPosition*. Raycast: a ray that moves in a particular direction after being emitted from a position in a 3D space, similarly to a laser. The *RaycastHit* function *CreateForwardRaycast()*; used for the hit variable is as follows:

```
private RaycastHit CreateForwardRaycast()
    {
        RaycastHit hit;

        // Create a ray stemming from our controller/hand
        Ray ray = new Ray(transform.position, transform.forward);

        Physics.Raycast(ray, out hit, defaultLength);
        return hit;
    }
```

CODE SNIPPET 4.13:
Physics Pointer Script Snippet 4

The *hit* variable is used when the pointer aims at an object/item; *transform.positon* is used throughout this script, and is the location of our controller/hand. The *out* keyword is used to store the data received from the *ray* into our hit function, which is what will be returned from this function.

```
// Default length for if nothing is hit

    private Vector3 DefaultEnd(float length)
    {
        return transform.position + (transform.forward * length);
    }
```

CODE SNIPPET 4.14:
Physics Pointer Script Snippet 5

This function will assume that if the pointer hasn't made any collisions, it will return a length, with *transform.position*; this is added by *transform.forward*, multiplied by the *length* passed through, which is the length originally set.

Ultimately, the *CalculateEnd()* function ends with the two final point variables being used in the following if statement:

```
    // If something is hit, the ending hitpoint will be the object which is
        then stored
    if (hit.collider)
        endPosition = hit.point;

        // Otherwise, use our defaultLength
        return endPosition;
    }
```

CODE SNIPPET 4.15:
Physics Pointer Script Snippet 6

Update in Unity is a function that refreshes a script every frame, checking for any statements that would cause changes in the program. In this case, it is required that the function updates the *UpdateLength()* function, as it uses a renderer which is constantly checking for collisions.

4.4 Pointer Events

The Pointer Events script was a major part of the object interaction phase. This script controls the visual interaction with objects, using different variables to recognise what form of interaction the user has with the object; ie, hover over, click, release. Tutorials online assisted me with developing the different object interactions [36].

```
public class PointerEvents : MonoBehaviour, IPointerEnterHandler,
    IPointerExitHandler, IPointerDownHandler, IPointerUpHandler,
    IPointerClickHandler
{
    // Initialise basic colours for the different modes an object will enter
    // All default to white, however can all be changed in Unity's Inspector
        mode with the Color. colour wheel
    [SerializeField] private Color normalColor = Color.white;
    [SerializeField] private Color enterColor = Color.white;
    [SerializeField] private Color downColor = Color.white;
    [SerializeField] private UnityEvent OnClick = new UnityEvent();

    private MeshRenderer meshRenderer = null;
    }
```

CODE SNIPPET 4.16:
Pointer Events Script Snippet 1

The *[SerializeField]*'s in this script are used to ensure that the private variables are visible when using the inspector.

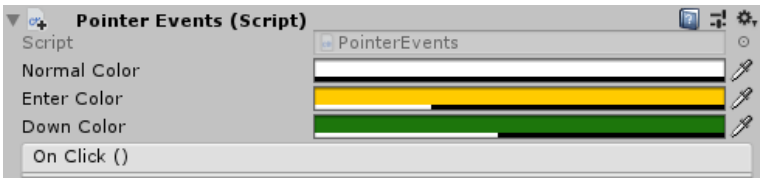

FIGURE 4.1: Image of Inspector within the Unity Application; displaying the PointerEvents function

The *Color* variables are used to initialize unique colours when the pointer interacts with an object. *normalColor* creates a shader over the object when the user has no interaction with it; *enterColor* creates a shader over the object when the user has the pointer hovering over the object, and *downColor* is used for when the object has clicked the object.

The variable, *OnClick*, triggers a Unity Event constructor when a button has been pressed. This allows certain elements to be triggered when the object has been clicked; in this case, a UI Panel to activate.

```
private void Awake()
    {
        meshRenderer = GetComponent<MeshRenderer>();
    }
```

CODE SNIPPET 4.17:
Pointer Events Script Snippet 2

Within this *Awake()* function, new material colours need to be initialised when the pointer aims at the object.

```
public void OnPointerEnter(PointerEventData eventData)
    {
        // Changes object colour when pointer hovers over object
        meshRenderer.material.color = enterColor;

        print("Enter");
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        // Changes object colour when pointer stops hovering over object
        // We will set this to 100% Opacity, as we do not want a colour
        meshRenderer.material.color = normalColor;

        print("Exit");
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        // Colour changes when object is pressed
        meshRenderer.material.color = downColor;

        print("Down");
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        // Changes object colour when pointer hovers over object after click
        meshRenderer.material.color = enterColor;

        print("Up");
    }
```

CODE SNIPPET 4.18:
Pointer Events Script Snippet 3

These functions all work similarly, using the three colours set at the top of the script to change the objects material colour depending on the varying action that is being applied onto the object. As an example, the *enterColor* will be rendered onto the object when the pointer has lifted it's click off the object. This means that the object will retain its shader after being clicked.

Turning Panel On and Off

Using *PointerEvents*' Unity event, *OnClick*, a runtime only function can be set, which will enable the panel. This is done by using the Inspector, and setting the *GameObject* (panel) to *SetActive* (bool). With it being a boolean, whether it remains active or not can be controlled using a script.

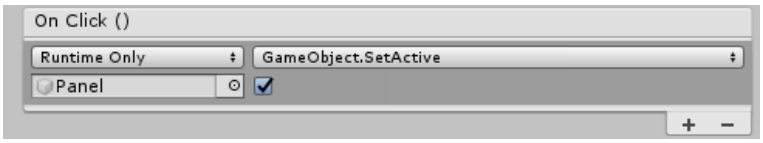

FIGURE 4.2: Image of Inspector within the Unity Application; displaying the *SetActive* feature.

4.5 Timer Script

Below shows the script used to turn a panel off after it is activated by clicking on an object with the pointer. Resources online assisted with the creation of a timer [37].

```
public class TextTimer : MonoBehaviour
{
    public float time = 20; //Seconds to read the text

    void Start()
    {
        Destroy(gameObject, time);
    }
}
```

CODE SNIPPET 4.19:
Timer Script Snippet 1

This is a simple script that sets a variable *time* which determines how long the object will be active for, in our case, that object will be the panel. *Destroy* works such that *Destroy(Object obj, float t = 0.0F)*, with *obj* being the object to be destroyed and *t* being the delay desired before actually destroying the object. As the time is set to 20 seconds, seconds can be used in this function instead of entering a float value.

4.6 Active Script

An issue faced during this process, was that the pointers were constantly on. This was an aspect of design that was not intended, so a script was implemented that could control whether the pointer was active or not. Tutorials online assisted with ensuring that the pointers could activate and deactivate by button press [38].

```
public class Active : MonoBehaviour
{
    // Initialise the hide/show with a button press
    public OVRInput.Button menu = OVRInput.Button.PrimaryHandTrigger;

    private bool activity;
```

CODE SNIPPET 4.20:
Active Script Snippet 1

A menu variable was created that would be the button trigger, which would toggle the objects (in our case, the objects are the pointers). Although it is shown that the button used is the Primary Hand Trigger, it was later changed to the controllers 'A' button via Unity's inspector model, as it clashed with Unity's grabber script, which used the same button.

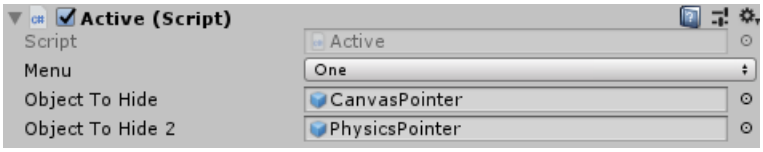

FIGURE 4.3: 'One' resembles the controllers 'A' button.

```
void Start()
    {
        activity = false;
    }
```

CODE SNIPPET 4.21:
Active Script Snippet 2

A variable *activity* was devised, which was set to *false*. This is done so that when the game started, the objects also start as inactive.

```
    // Set the two pointer we want to disable/enable, these can be changed
        in Unity's inspector

    public GameObject ObjectToHide;
    public GameObject ObjectToHide2;
```

CODE SNIPPET 4.22:
Active Script Snippet 3

As there are two pointers, both were labelled as objects to this script, ensuring that both the pointers will be able to activate and deactivate. Later it was understood that the Canvas Pointer is not required for this program, however, it was retained as the code yielded no issues.

```
void Update()
    {
        ObjectToHide.SetActive(activity);
        // Here disables the first object
```

```
        if (OVRInput.Get(menu))
            // If button is pressed, activate the pointer
        {
            activity = true;
        }

        else
        {
            // Else, deactivate it

            activity = false;
        }
    }
```

CODE SNIPPET 4.23:
Active Script Snippet 4

The *Update* function introduces the main section of this script. When the menu button is pressed, it will either activate/deactivate the pointers, depending on its previous state. This is done for both objects.

```
void Update()
    {
       ObjectToHide2.SetActive(activity);
        //Here disables the second object in a similar fashion

        if (OVRInput.Get(menu))
            // If button is pressed, activate the pointer
        {
            activity = true;
        }

        else
        {
            // Else, deactivate it

            activity = false;
        }
    }
```

CODE SNIPPET 4.24:
Active Script Snippet 5: Similar active/non-active function for ObjectToHide2

4.7 Environment

A final viable scene was implemented, as originally a basic plane with a few simple objects was used. Browsing Unity's Asset Store, there was a bathroom scene which looked almost identical to the original low-fidelity prototype sketches.

As the intention was to show the side effects of and provide information about alcohol, a beer bottle was also purchased.

To mimic a public restroom, a script was created to control the scene's lighting. The following code was adapted from resources online. [39]

```
public class Flicker : MonoBehaviour
{
    Light testLight;

    public float minWaitTime;
    public float maxWaitTime;
```

CODE SNIPPET 4.25:
Environment Script Snippet 1

Two variables were generated that controlled the minimum and maximum time before the lighted turned off from on. This randomness is observed in real flickering lights, so it was important to ensure this was reflected in the experience. The *min* and *max WaitTime*'s were customisable within Unity's inspector, so they could be changed appropriately to fit the scene.

```
void Start()
    {
        testLight = GetComponent<Light>();

        StartCoroutine(Flashing());
    }
```

CODE SNIPPET 4.26:
Environment Script Snippet: 2

Coroutines are especially useful for modelling behaviour over consecutive frames, a reason it was used in this script. A *yield* function was not required as light flickering was aimed to be continuous throughout the frames; and a *yield* function suspends a *Coroutine* execution.

```
IEnumerator Flashing()
    {
        while (true)
        {
            yield return new WaitForSeconds(Random.Range(minWaitTime,
                maxWaitTime));

            testLight.enabled = !testLight.enabled;
        }
    }
```

CODE SNIPPET 4.27:
Environment Script Snippet: 3

As there are no *yield*'s (as previously mentioned), the *while (true)* statement will essentially loop forever. With this in mind, *WaitForSeconds* will select a random time between the minimum and maximum wait time set within the unity inspector. As a result, the lights sporadically flicker.

4.8 Post-Processing

Postprocessing is an important feature in Unity which will be the basis of creating all the visual side effects and provide the element of realism. This is done by initially rendering in the memory of the video card. Shaders are then used to apply post-processing filters to the image buffer before displaying the image to the device. Through the alteration of the rendering effect and thus its perception, side effects that mirror real-life ones were able to be simulated.

For side effects, effects were required to be fully immersive. Post processing is extremely effective at doing this, as Unity allows for the creation of effects that require the entire image to change in a certain way. Tutorials and templates were used to assist my work within this section. [40] [41]

```
public sealed class Drunk_PPP : PostProcessEffectSettings
{
    // Create two new floats to change the screens distortion
    [Range(0f, 1f), Tooltip("Drunk Distortion effect intensity")]
    public FloatParameter amplitude = new FloatParameter { value = 0.5f };

    [Range(0f, 100f), Tooltip("Drunk Distortion effect frequency")]
    public FloatParameter frequency = new FloatParameter { value = 0.5f };
```

CODE SNIPPET 4.28:
Post-processing Script Snippet: 1

For the Drunk Post Processing render, two float parameters were created first, *amplitude* and *frequency*. These were devised so the severity of distortion could be customised; although both values are set to *.5*, it could be changed through Unity's Inspector.

```
public sealed class Drunk_PPPRenderer : PostProcessEffectRenderer<Drunk_PPP>
{
    public override void Render(PostProcessRenderContext context)
    {
        var sheet =
            context.propertySheets.Get(Shader.Find("Custom/Drunk_PPP"));

        // Set float parameters
        sheet.properties.SetFloat("_Amplitude", settings.amplitude);
        sheet.properties.SetFloat("_Frequency", settings.frequency);
```

CODE SNIPPET 4.29:
Post-processing Script Snippet: 2

The two parameters were set as *_Amplitude* and *_Frequency*, to be used in the Post Processing shader.

```
#include
    "Packages/com.unity.postprocessing/PostProcessing/Shaders/StdLib.hlsl"

    TEXTURE2D_SAMPLER2D(_MainTex, sampler_MainTex);
    float _Amplitude;
    float _Frequency;
```

```
    // Float2 stores the x and y of each pixel
    float2 distortion;
```

CODE SNIPPET 4.30:
Post-processing Script Snippet: 3

Within the .shader file, two parameters were initialised into floats. A distortion *float2* was also included (which stores the x and y movement of each pixel). During the development of distortions, there was a lack of control for the speed and size of distortion that was then corrected by the implementation of *float4* vectors:

```
// Create two new vectors to control the speed and size of the distortion

    [Tooltip("Speed of the Distortion only x and y used")]
    public Vector4Parameter speed = new Vector4Parameter { value = new
        Vector4(1, 1, 0, 0)};

    [Tooltip("Size of the Distortion only x and y used")]
    public Vector4Parameter size = new Vector4Parameter { value = new
        Vector4(1, 1, 0, 0) };
```

CODE SNIPPET 4.31:
Post-processing Script Snippet: 4

The snippet above shows the two new implementations to the post-processing script. Although introduced as *Vector4*, z and w axis' were kept the same, and only x and y were changed.

```
float4 _Speed;
float4 _Size;
```

CODE SNIPPET 4.32:
Post-processing Script Snippet: 5

With full control of the frequency, amplitude, speed and size of the distortions, a full x and y distortion effect was created:

```
// Fragment shader loops over every picture, using x, y, z and w

    float4 Frag(VaryingsDefault i) : SV_Target
    {
        // Float2 stores the x and y of each pixel
        float2 uv = i.texcoord;

        // Different graphs types used to create a more unique pixel movement
        distortion.x = (sin((_Speed.x * _Time.y) + (uv.x * _Size.x) *
            _Frequency)) * _Amplitude;
        distortion.y = (cos((_Speed.y * _Time.y) + (uv.y * _Size.y) *
            _Frequency)) * _Amplitude;
```

CODE SNIPPET 4.33:
Post-processing Script Snippet: 6

Sine and Cosine were used to change the distortion structure. The diagram below presents a Sine, Cosine and Tangent graph. Using a Tangent created major issues

with the way the distortion would move and did not draw any favourable similarities to the side effects of being drunk. Sine and Cosine waves were thus used to create an accurate distortion that mimics the feeling of being drunk or lightheaded.

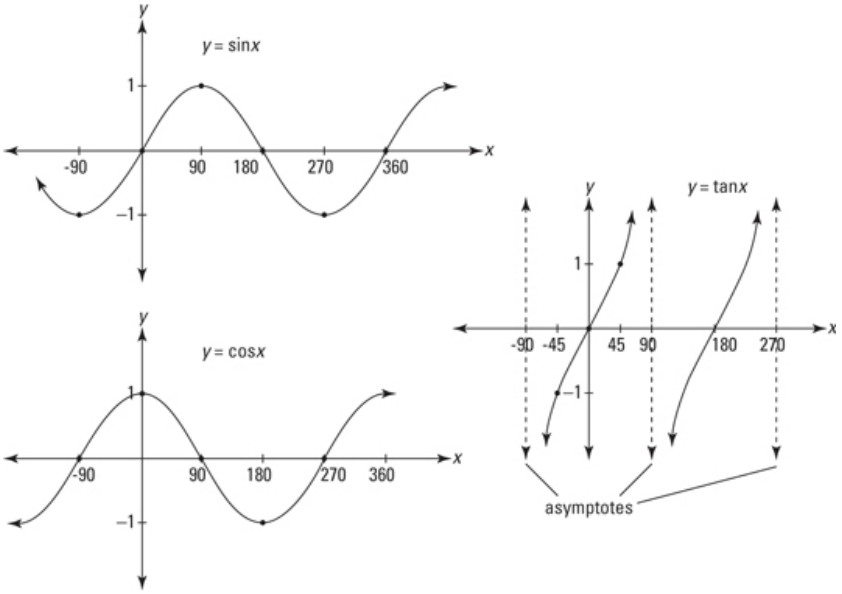

FIGURE 4.4: Sine, Cosine and Tangent graphs depicting how distortion works within Unity's post-processing shaders.

Chapter 5

Testing

In light of the COVID-19 pandemic, user testing became extremely limited for the project. Volunteers consisted of those within an isolated group as well as a handful of people online. Discord was used to communicate with volunteers online: a digital platform used to communicate with a range of people from gamers to educators.

Due to the cost of current home VR Kits, there was difficulty in finding volunteers who had Virtual Reality setups within their homes, which also had compatibility with the project. An Oculus hardware was necessary; therefore, the testing group was limited to 10 people (8 online and 2 in person). All users had reported having previous experience with alcohol effects and therefore could directly compare both experiences. Testing was implemented both during the investigation process as well as upon completion of the VR experience.

5.1 Regression Testing

The most important form of testing that occurred throughout the creation of the project were manual regression tests. As objects and scripts constantly interact with each other, any new introductions could cause previously implemented software to perform in ways that were not intended, or cause additional errors and bugs.

Regression tests were administered after each new script or object. This was mostly done through playtesting: a method of quality control in which the user plays with incomplete versions of the experience. This encouraged feedback throughout the process and allowed for changes to be made to the level design, gameplay as well as resolve any issues that arose. As this was carried out after each implementation, regression testing was made simpler. Through separating the work, resolutions for any issues were easily identified.

5.2 Usability Testing

Usability testing was conducted with the 10 gathered participants after the completion of the VR experience. These participants fit into the aimed stakeholder group, which are people between the ages of 16 and 23. As a majority of participation's were conducted through an online application, a think-aloud technique was used to understand how the participant was reacting to certain aspects of the virtual experience in real-time.

The 8 online participants also used screen share, a feature that is embedded within the Discord application. These features grant the ability to simultaneously view

what the participant is doing within the virtual reality, as well as listen and see them in the real-world. The 2 interactions conducted in person had a computer monitor nearby which allowed for a similar viewing into the virtual world.

5.3 Provided Tasks

Participants were given 5 set instructions to ensure that the project was adequate. These tasks were spoken to all participants analogously to guarantee that there were no extraneous variables in the tasks' results.

	Task	Task Aim
1	Move around your room and move your arms about	Ensure that mobility is fluid in relation to real life. If a user's movement in real life is almost identical to movement in-game, ie; walking and moving hands.
2	Use the grab and point controls on your controller	Virtual hands can grab and point. If a user's actions on their controllers allows them to make a grab and point gesture in game.
3	Read the provided information out loud on the drug display	User is able to use the pointer successfully to bring up a panel which describes the drug. Task is considered as completed if they are able to read the panel to the end within the panels set timer.
4	Take the drug how you would in real life	User can grab the bottle, and will only cause the side effects to occur when held up near the mouth to mimic drinking.
5	Walk around the room, how do you feel?	User must walk around the room and explain how they are feeling.

TABLE 5.1: Table detailing the tasks and aims to be deemed successful.

5.3.1 Results & Findings

Task 1:
8/10 Users succeeded with this task. The 2 unsuccessful users had limited movement due to their environment, so could not imitate walking around.

Task 2:
All users succeeded with this task. There were no issues with the in-game hands making a grab gesture or activating the pointer with the controller's 'A' button.

Task 3:
6/10 users were successful with completing this task. 3 unsuccessful users reported that the beer bottle would not react to the pointer, however, a restart of the game later fixed the recurring issue. The remaining unsuccessful user reported that the panel's timer was too fast, and that they were unable to complete reading before it disappeared.

Task 4:
9/10 users found success with this task. An unsuccessful user reported that the side effects weren't working. Again, a restart of the game hastily fixed the issue, so a user's hardware is assumed to be the issue for this task.

Task 5:
Similar results to Task 1, the same users found difficulty navigating through their tight real-life environment. Although the game allows movement with the controller's analogue stick, this does not represent real life movement so was therefore recorded as unsuccessful.

Discussion

All users commented on the usefulness that the project has for educating a young audience in regards to the dangers of drugs; highlighting the success within this focal group.

The chances of extraneous variables to occur were increased due to the limitations of the pandemic's lock down; hence some task failures occurred due to a user's environmental limits; ie, their VR space was too confined to test mobility.

With these test results, slight changes were also made to how the game would work. Panel time was extended to 20 seconds, up 5 seconds from the previous 15 second timer. This was done so that user's with various levels of reading skills were all capable of reading the panel before it would disappear.

```
public class TextTimer : MonoBehaviour
{
    public float time = 20; //Seconds to read the text
}
```

CODE SNIPPET 5.1:
Timer Script Change: Float time changed to 20 instead of 15

A larger hit detection on the bottle was also implemented. Although reopening the software gave a temporary fix for the users, increasing the hit detection zone of the bottle meant that the pointer was more likely to cause an interaction. This was done in case there were any issues with the way the pointer had been angled towards the bottle.

Usability testing was invaluable, as issues were discovered that were previously deemed as an acceptable feature.

Chapter 6

Conclusion

This chapter aims to analyse the final project, delving into the successes and failures of the project itself; future works detailing the next steps to further develop the project, and what was learnt throughout the creation of this VR Experience.

6.1 System Successes

Overall, the project has been successful and would be a good tool in educating users about the effects of drugs through Virtual Reality. Functionality requirements have been met, with an object having the ability to be picked up, moved, and used to display side effects. With this in mind, the experience successfully resembles that of a real-life scenario, with palpable objects and an ideal scene.

In regards to the consideration of different learning methods, various learning strategies have been incorporated. A panel has been implemented to effectuate better learning for users that require language educating, with an inclusion to side-effects aiming to accommodate a visual learner.

6.2 System Failures

One limitation of the project is the lack of sensory information that it presents. The virtual experience is only capable of emitting visual data, an issue which Gallace's research highlighted [17]; a lack of senses could affect the realism that the project displays.

Another drawback is the inability to test the difference between sober and non-sober in game. With all the drugs' side effects being displayed on the panel, there is no way to directly compare how the drugs affect physical performance in a real-world setting.

6.3 Future Works

The project created is only an initial steppingstone for what could become a more advanced and useful tool to be used within an educational system. This section will look at the future implications that will allow for this virtual experience to progress to that stage.

6.3.1 Mini-games

To combat the system failures mentioned, mini-games will be implemented in the near future. These mini-games will be able to monitor the difference between being under the influence, with examples such as: an aiming game to target hand-eye coordination side effects and a reaction time mini-game which will compare user's times while under the influence.

This will give the VR experience more depth as well as allow for the user to understand the dangers that drug abuse could bring about.

6.3.2 Increased senses

To target another limitation, the addition of more senses will greatly enhance the user experience [17]. Tactile feedback through the VR controllers as well as an introduction to sound would allow the implementation of up to three senses.

6.3.3 Variety

Another application that could be added, is the addition of a variety of drugs. This experience only represented one drug, however it would be more effective when this could be compared and combined with other narcotic education.

6.3.4 Support other VR Devices

As the VR Experience only supports Oculus headsets, another useful future implementation would be creating compatibility with other large VR Headset companies, such as HTC, Valve or Windows. By doing this, there will be a higher chance of consumption expenditure via educational facilities such as Secondary Schools and Universities, the age bracket in which this project is targeted at.

6.4 Self-evaluation

Throughout the entire development of this project, I've delved deep into systems and software that I had previously not attempted, such as Unity and VR. I have also been developing the project with a new, unfamiliar language (C#) which despite difficulty, was a welcomed challenge. The project provided a steep learning curve, with available resources scarce due to Virtual Reality games being a relatively new gap within the market. However, despite adversity, the development process was extremely rewarding and enjoyable.

My original scope of the project was to develop a VR experience in which young users can educate themselves of the dangers of drug abuse. Although basic, I am proud of what I have created. With the future implementations mentioned I believe that it could take this project to a level that is revered among educational facilities, such as schools, colleges, or centres.

6.5 Final Comments

As mentioned beforehand, the development of the project has been invaluable. Thank you to my supervisor Daniel Buchan for providing support and guidance throughout.

Bibliography

[1] D. Miller, J. Sinanan, and X. Wang. "How the World changed Social Media". In: (2016). URL: https://library.oapen.org/viewer/web/viewer.html?file=/bitstream/handle/20.500.12657/32834/604151.pdf?sequence=1&isAllowed=y.

[2] M. A. Bachhuber. "Buying Drugs Online in the Age of Social Media". In: (2017). URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5678407/.

[3] M. A. Pentz. In: *Prevention of Adolescent Substance Abuse through Social Skill Development* (1983), pp. 193–197. URL: https://books.google.co.uk/books?hl=en&lr=&id=2RKmvE4HjAMC&oi=fnd&pg=PA195&dq=M.+A.+Pentz.+âĂœPrevention+of+Adolescent+Substance+Abuse+through+Social+Skill+DevelopmentâĂİ+&ots=X-bP3VIye7&sig=zWTfi5B9yPc4wX5YCNp_7HmAel8&redir_esc=y#v=onepage&q&f=false.

[4] H. Pilkington. "Beyond 'peer pressure': Rethinking drug use and 'youth culture'". In: (2006). URL: https://pubmed.ncbi.nlm.nih.gov/17689368/.

[5] J. McComas, M. MacKay, and J. Pivik. "Effectiveness of virtual reality for teaching pedestrian safety". In: (2004). URL: https://www.liebertpub.com/doi/abs/10.1089/109493102760147150.

[6] D. B. Kandel and J. A. Logan. "Patterns of drug use from adolescence to young adulthood". In: (2011). URL: https://ajph.aphapublications.org/doi/abs/10.2105/AJPH.74.7.660.

[7] E. Bierstadt. "Improvement in Stereoscopes". In: (1876). URL: https://patentimages.storage.googleapis.com/1b/2d/ce/9fc3bcdd7e2f66/US174893.pdf.

[8] M. L. Heilig. "Sensorama Simulator". In: (1962). URL: https://patentimages.storage.googleapis.com/90/34/2f/24615bb97ad68e/US3050870.pdf.

[9] R. Coulter, L Saland, and T Caudell. In: *Medicine Meets Virtual Reality* 15 (2012), pp. 155 –176. URL: https://link.springer.com/article/10.1007/s10803\-012\-1544\-6.

[10] M. R. Kandalaft et al. "Virtual reality social cognition training for young adults with high-functioning autism". In: *Medicine Meets Virtual Reality* (2007). URL: https://books.google.co.uk/books?hl=en&lr=&id=fg3vAgAAQBAJ&oi=fnd&pg=PR1&dq=Medicine+Meets+Virtual+reality+15.+Page+155.+R+Coulter,+L+Saland,+T+Caudell.+2007&ots=eOp0Qg-ZjV&sig=8q0sZLrgjXuM4xtpWKfjxPBr4MI&redir_esc=y#v=onepage&q&f=false.

[11] S. G. Weinbaum. In: *Pygmalion's Spectacles* (1935). URL: https://books.google.co.uk/books?hl=en&lr=&id=r5wHDAAAQBAJ&oi=fnd&pg=PT3&dq=Pygmalion\%E2\%80\%99s+Spectacles&ots=Br7UHGTKOG&sig=enDKdob6TD9eWQQXdFaUvvRLnIE&redir_esc=y#v=onepage&q=Pygmalion\%E2\%80\%99s\%20Spectacles&f=false.

[12] P. Baudisch. "Virtual reality in your living room: technical perspective". In: (2015). URL: https://dl.acm.org/doi/fullHtml/10.1145/2754393.

[13] L. P. Berg and J. M. Vance. "Industry use of virtual reality in product design and manufacturing: a survey". In: (2016). URL: https://link.springer.com/article/10.1007/s10055-016-0293-9?&utm_medium=other&utm_source=other&utm_content=4172018&utm_campaign=10_dann_ctw2018_4_econ_24.

[14] Trastica. "Virtual Reality for Consumer Markets". In: (2016). URL: https://tractica.omdia.com/research/virtual-reality-for-consumer-markets/.

[15] P. M. G. Emmelkampa et al. "Virtual reality treatment versus exposure in vivo: a comparative evaluation in acrophobia". In: *Behaviour Research and Therapy* 40, 5 (2002), pp. 509 –516. URL: https://www.sciencedirect.com/science/article/abs/pii/S0005796701000237.

[16] D. W. Mizell. "Virtual Reality and Augmented Reality In Aircraft Design and Manufacturing". In: (1996). URL: http://library.um.edu.mo/ebooks/b11505011.pdf#page=40.

[17] A. Gallace. "Multisensory Presence in Virtual Reality: Possibilities & Limitations". In: (2012). URL: https://www.igi-global.com/chapter/multisensory-presence-virtual-reality/55937.

[18] R. L. Gregory. "Perceptual illusions and brain models". In: (1968). URL: https://royalsocietypublishing.org/doi/abs/10.1098/rspb.1968.0071.

[19] R. Sperry. "Split-Brain". In: (1968). URL: https://embryo.asu.edu/pages/roger-sperrys-split-brain-experiments-1959-1968.

[20] J. T. Bruer. "In search of brain-based education". In: *The Jossey-Bass Reader on The Brain and Learning* (2007), pp. 51 –69. URL: https://books.google.co.uk/books?hl=en&lr=&id=ZZmyoWI90ckC&oi=fnd&pg=PA51&dq=left+hemisphere+brain+education&ots=l4iDm1MWEU&sig=GxOnMKLrx88m7GIAD2DU23-trkU&redir_esc=y#v=onepage&q=left\%20hemisphere\%20brain\%20education&f=false.

[21] D. B. Browne. "Learning Styles and Native Americans". In: (1986). URL: https://eric.ed.gov/?id=ED297906.

[22] L. D. Johnston, P. M. O'Malley, and J. G. Bachman. "Attitudes and Beliefs About Drugs Among Seniors". In: *Drug Use, Drinking, and Smoking: National Survey Results From High School College, and Young Adults Populations* (1988), pp. 127 –145. URL: https://books.google.co.uk/books?hl=en&lr=&id=GxBol7sII4MC&oi=fnd&pg=PR1&dq=johnston+1989+drugs&ots=6VPWqSGioE&sig=dwWjL6O9nBMdEdlu-aka5zuQ_i4&redir_esc=y#v=onepage&q=johnston\%201989\%20drugs&f=false.

[23] P. M. C. Miller and M. Plant. "Drinking, smoking, and illicit drug use among 15 and 16 year olds in the United Kingdom". In: (1996). URL: https://www.bmj.com/content/313/7054/394.short.

[24] N. Heather and T. Stockwell. "Brief Interventions". In: *The Essential Handbook of Treatment and Prevention of Alcohol Problems* (2004), pp. 117 –138. URL: https://books.google.co.uk/books?hl=en&lr=&id=54yUzyDiTJ0C&oi=fnd&pg=PR5&dq=(Heather.+N+\%26+Stockwell.+T.,+2004).&ots=pHwNKE3qDy&sig=I7CxJ7EoklYQLmrU5vydDl2HVUc&redir_esc=y#v=onepage&q=(Heather.\%20N\%20\%26\%20Stockwell.\%20T.\%2C\%202004).&f=false.

[25] B. Sessa. "The psychedelic renaissance: Reassessing the role of psychedelic drugs in 21st century psychiatry and society". In: (2012). URL: https://psycnet.apa.org/record/2012-23435-000.

[26] D. W. Soole. "School-Based Drug Prevention Programs: A Review of What Works". In: (2008). URL: https://journals.sagepub.com/doi/abs/10.1375/acri.41.2.259.

[27] D. R. Black, N. S. Tobler, and J P. Sciacca. "Peer Helping/Involvement: An Efficacious Way to Meet the Challenge of Reducing Alcohol, Tobacco, and Other Drug Use Among Youth". In: *Journal of School Health* 68, 3 (2009). URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1746-1561.1998.tb03488.x.

[28] J. Hurry, C. Lloyd, and H. McGurk. "Long-Term Effects of Drugs Education in Primary School". In: *Addiction Research* 8, 2 (2000), pp. 183 –202. URL: https://www.tandfonline.com/doi/abs/10.3109/16066350009004419.

[29] S. G. Forman, J. A. Linney, and M. J. Brondino. "Effects of coping skills training on adolescents at risk for substance use". In: *Psychology of Addictive Behavior* 4, 2 (1990), pp. 67 –76.

[30] ChoppyPine. "Drug Simulator V.1.O". In: (2019). URL: https://choppypine.itch.io/drug-simulator-v10.

[31] Medis Media. "3D Organon VR Anatomy". In: (2016). URL: https://www.3dorganon.com/.

[32] J. Arnett. "The Young and the Reckless: Adolescent Reckless Behavior". In: (1995). URL: https://journals.sagepub.com/doi/abs/10.1111/1467-8721.ep10772304?journalCode=cdpa.

[33] BeepBoopIndie. "Unity 5.6 Tutorial | Distance Between Objects (Vector3)". In: (2017). URL: https://www.youtube.com/watch?v=OMPV-duv25Q.

[34] VR with Andrew. "[01] [Unity] Simple VR Pointer". In: (2019). URL: https://www.youtube.com/watch?v=rgTshoVCZVQ&t=435s.

[35] Oculus. "Unity Integration". In: (2020). URL: https://developer.oculus.com/downloads/package/unity-integration/.

[36] VR with Andrew. "VR-SimplePointer". In: (2019). URL: https://github.com/C-Through/VR-SimplePointer/blob/master/Assets/_SimplePointer/Scripts/PointerEvents.cs.

[37] K. Jones. "How to make disappear a GUI text after an amount of time?" In: (2015). URL: https://answers.unity.com/questions/970467/how-to-make-disappear-a-gui-text-after-an-amount-o.html.

[38] Programmer. "Unity 3d VR - Hide and Show Model on Controller Button Click". In: (2018). URL: https://stackoverflow.com/questions/53566557/unity-3d-vr-hide-and-show-model-on-controller-button-click.

[39] Ashkan$_g$c. "Blinking Lights". In: (2010). URL: https://answers.unity.com/questions/11510/blinking-lights.html.

[40] Unity. "Writing Custom Effects". In: (2018). URL: https://github.com/Unity-Technologies/PostProcessing/wiki/Writing-Custom-Effects.

[41] A. Trotter. "Drunk Post Processing Shader - Unity Basics". In: (2018). URL: https://www.youtube.com/watch?v=AcCDqH8LA9M&t=.

List of Figures

Listings

Appendix A

Background Research - Survey Results

Conducting a short survey on Google Forms, 5 Quantitative questions were asked based on participant's previous knowledge into Virtual Reality and drug education.

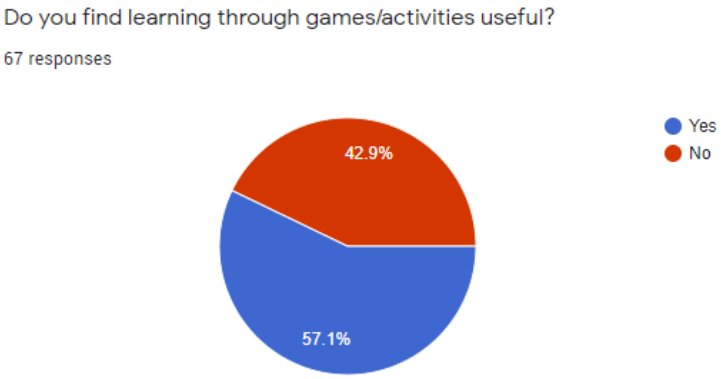

FIGURE A.1: Question 1 of the form

57.1% of responses stated that they do find learning through games useful, with 42.9% stating that they don't. Although there is a slight difference, it is most safe to support the idea that there is a split between the learning styles of people.

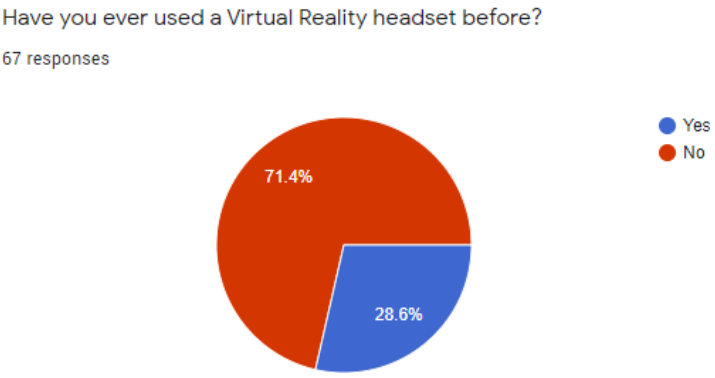

FIGURE A.2: Question 2 of the form

71.4% of responses have never used a Virtual Reality headset before. This is a substantial amount, which is not surprising due to Virtual Reality still being a novelty item within the household or schools; especially due to costs.

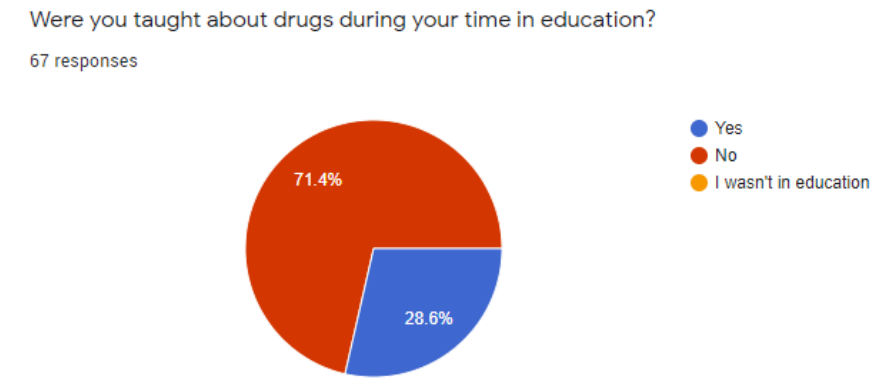

FIGURE A.3: Question 3 of the form

Question 3 favoured similar results, with 71.4% of responses detailing that they have not learnt about drugs through their time in education.

This is alarming due to the ever exceeding rate of drug abuse occurring within the UK, with the 21st century being hailed as the psychedelic renaissance [25].

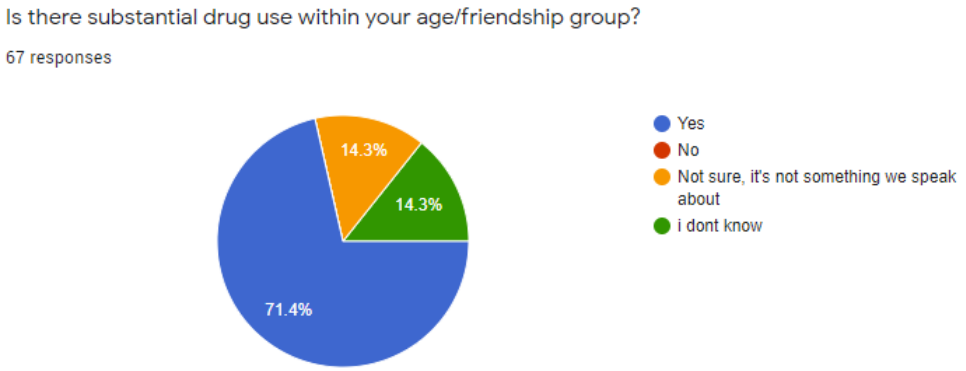

FIGURE A.4: Question 4 of the form

An additional question was asked to understand whether drugs were an issue recurring within their friendship group.

71.4% said yes; a large percentage of the participants. This could be due to drug use being such a common occurrence within the age group, that discussions revolving such are no longer taboo.

One-to-one questions were asked with respondents who optionally offered to give extra information.

Question 5:

Do you think drugs are dangerous?

To an extent, I think smoking and alcohol minimally is not dangerous, but drug abuse definitely is.
Taken in moderation I don't think so but there's always a risk.
Yes I think they're dangerous even alcohol and smoking too.
Yes, seen and heard people die from overdoses.
Yes people die.
Not really no, people come into class with red eyes from weed but it seems funny, not dangerous

TABLE A.1: Table detailing the responses receive from optional Question 5.

These responses show that a majority of people agree that drugs are quite dangerous, yet, the education system is not educating young adults.

Appendix B

Testing

B.1 Usability Testing: Individual Reports

User	Task 1	Task 2	Task 3	Task 4	Task 5
1	✓	✓	✗	✓	✓
2	✗	✓	✓	✓	✗
3	✓	✓	✓	✓	✓
4	✓	✓	✓	✓	✓
5	✓	✓	✓	✗	✓
6	✗	✓	✗	✓	✗
7	✓	✓	✓	✓	✓
8	✓	✓	✓	✓	✓
9*	✓	✓	✗	✓	✓
10*	✓	✓	✗	✓	✓

TABLE B.1: Table detailing whether tasks given were completed or not. '*' signifies users who participated in-person rather than online

B.1.1 Unsuccessful task reasoning's

User 1: Task 3

Beer bottle failing to react to the pointer.

User 2: Task 1

Room too small; unable to walk around properly to test out the fluidity of movement.

User 2: Task 5

Similarly, room is small to properly test how movement is impaired with the side effect.

User 5: Task 4

Side effect reported as not working. Restarting the game has fixed the issue

User 6: Task 1

Room too small; unable to walk around properly to test out the fluidity of movement.

User 6: Task 3

Unable to read the panel to the end without it disappearing.

User 6: Task 5

Similarly, room is small to properly test how movement is impaired with the side effect.

User 9: Task 3

Beer bottle failing to react to the pointer.

User 10: Task 3

Beer bottle failing to react to the pointer.

B.2 Regression Testing

Implementation	Tick for no issues
Grabber and Grabbable	✗
Distance script	✓
Pointer	✓
Object	✗
Panel	✓
Post Processing	✗

TABLE B.2: Table detailing whether new tasks being completed yielded any new issues while playtesting

Issue with: Grabber and Grabbable

Issue detected with the collision being way too large. This caused the character model to be pushed into the air when grabbing an object. This occurred because collision on objects is on, so as it collides with an invisible hitbox, it will try to repel out of the hitbox and in turn, pushes the player away/in the air.

This was fixed by minimising the collision size drastically on the controllers/hands.

Issue with: Object

Pointer would not react with the object. This meant that a later implementation of a panel would not yield successful.

The issue was resolved by realising that the pointer used was set for canvas' only, rather than objects. PhysicsPointer was implemented alongside CanvasPointer to ensure that the object would be recognised by the pointer and would work.

Issue with: Post Processing

Originally used a Tangent graph to control the distortion. This caused huge amounts of distortion that would cause Unity to crash as soon as the side-effects were active.

This was resolved by toning the intensity down, however, a cosine wave was later introduced to replace the tan wave. This created a more mellow distortion that resembled being drunk.

Note on Regression Testing

Due to regression testing, issues have been rectified accordingly. Without the use of this form of testing, there would have been great difficulty attempting to debug the entirety of the project at it's final form.

Appendix C

Prototyping

C.1 Low-Fidelity

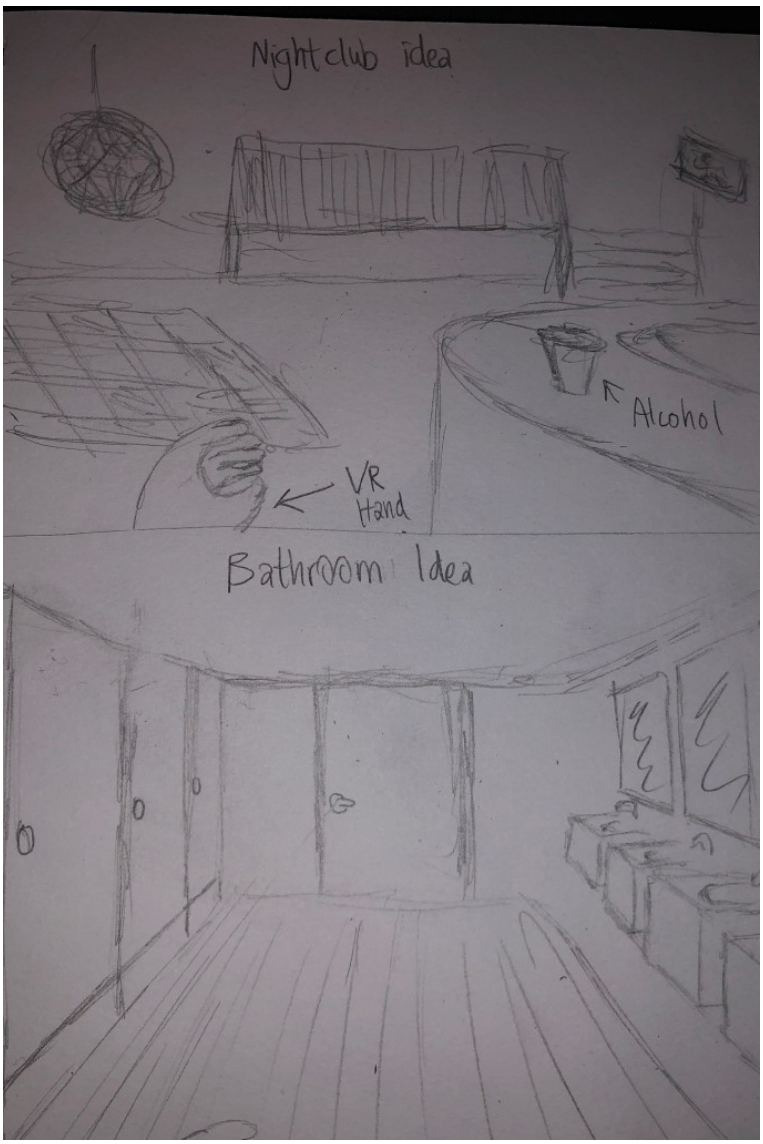

FIGURE C.1: Initial Wireframes.

2 Environments were created to suit the young adult partying lifestyle; a nightclub and a clubs' toilet.

C.2 High-Fidelity

Images below show the high-fidelity prototyping. Scripts were added to these object to check whether everything implemented was working before implementing finalised objects/effects.

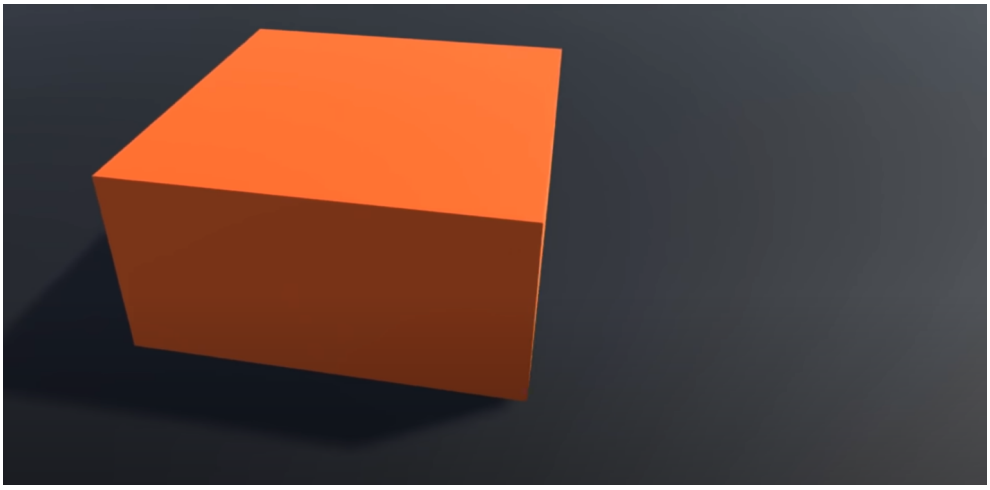

FIGURE C.2: Base

This image shows the initial base which was used as a table during the high-fidelity prototyping phase.

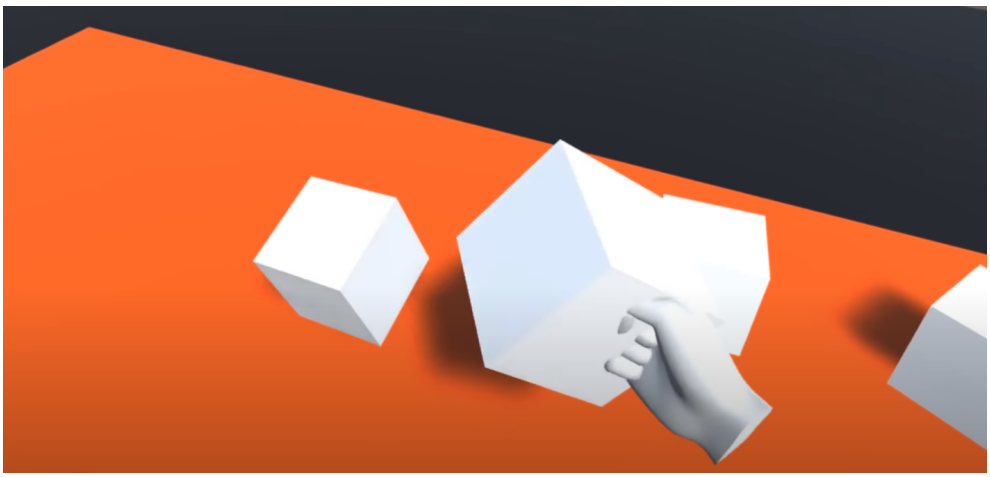

FIGURE C.3: Grabber and Grabbable

This image shows the grabber and grabbable functions being applied to simple objects, in this case, basic white cubes.

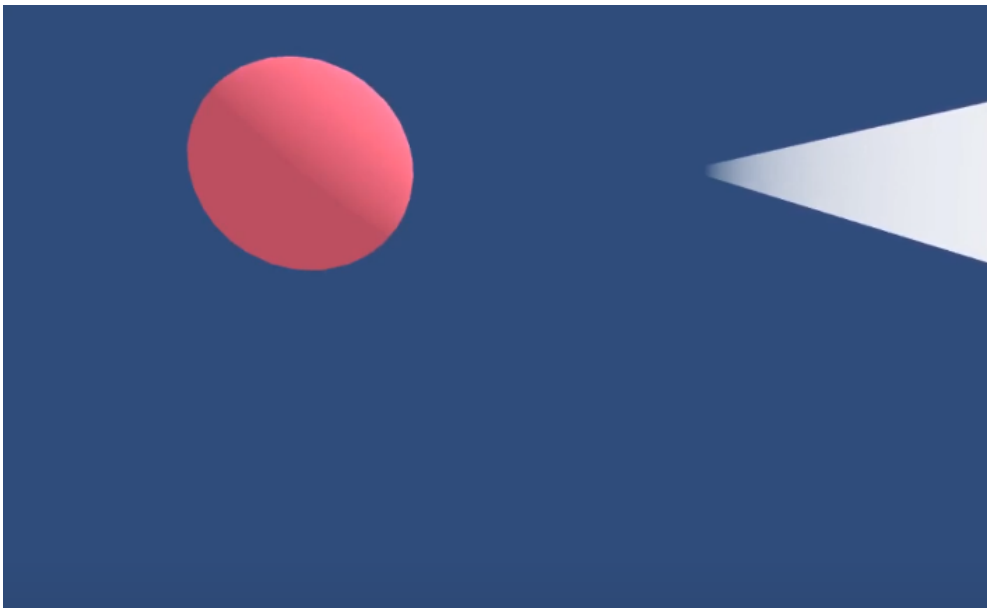

FIGURE C.4: Pointer

This image shows the pointer initially being used with an object.

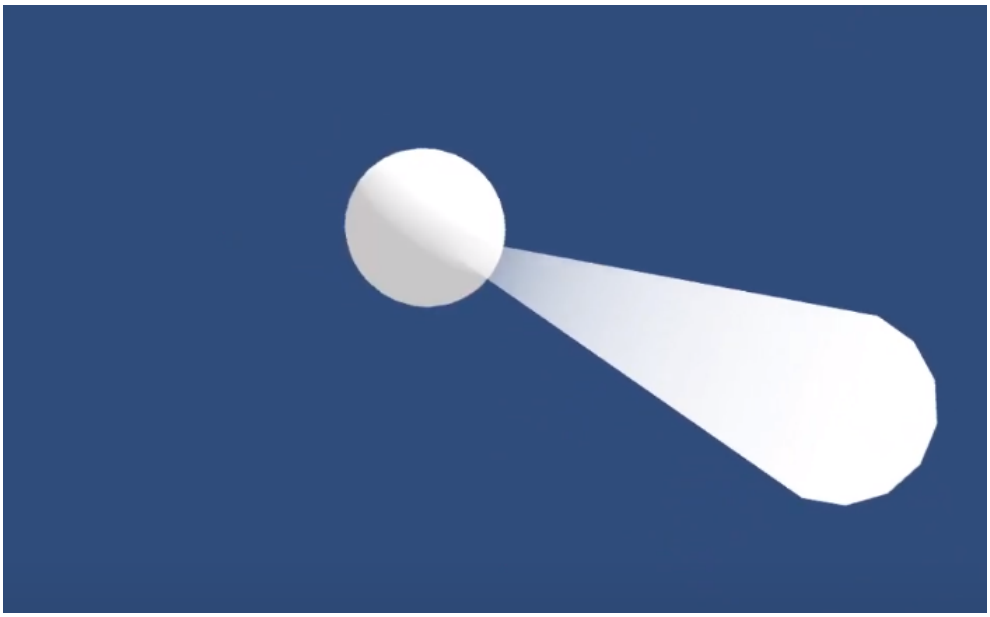

FIGURE C.5: Pointed

This image shows how the object changes when aimed at by the pointer.

Appendix D

Final Design

FIGURE D.1: Hand gestures are effective.

FIGURE D.2: Light is off due to flicker script.

FIGURE D.3: Realistic public toilets.

FIGURE D.4: Realistic public toilets 2.

FIGURE D.5: Pointer in action.

FIGURE D.6: Pointer when aiming at bottle.

FIGURE D.7: Pointer is clicked on object, therefore highlights a unique colour until released.

FIGURE D.8: Information panel.

FIGURE D.9: Beer bottle.

FIGURE D.10: Gesture required to drink.

FIGURE D.11: Post-processing active, side-effects display loss of balance and lack of distance awareness.

Appendix E

Code

The project folders are currently available on Github within this link: https://github.com/butrinto/ComputingProjectFinal.git. Navigate to the README.md file for information on where to locate the project's code.

It is also possible to try the project out yourself via this DropBox link: https://www.dropbox.com/sh/4h6qdwq7lclfhx7/AAChuq9yS4s4m83LDrq--Zrla?dl=0. Download all the files and run the *project2* application. **Please note** that an Oculus Rift headset is required, otherwise the project will fail to run.

If there are any issues with accessing the code via GitHub, all the necessary scripts are written within this Appendix (E).

ChangeUIText.cs

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

public class ChangeUIText : MonoBehaviour
{
    public Text uiText;
    public string setText;

    void Start()
    {
        uiText.text = "";
    }

    void Update()
    {
        uiText.text = setText;
    }
}
```

CODE SNIPPET E.1:
ChangeUIText.cs full code

PanelOpener.cs

```
/* This code was adapted from the following source by Jayanam;
    https://www.youtube.com/watch?time_continue=91&v=
LziIlLB2Kt4&feature=emb_title */
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PanelOpener : MonoBehaviour
{
    // Initialises the GameObject, to assign a Panel in Unity's inspector
    public GameObject Panel;

    public void OpenPanel()
    {
        // Check if Panel is assigned, null
        if (Panel != null)
        {
            // Toggles the active state
            bool isActive = Panel.activeSelf;

            Panel.SetActive(!true);
        }
    }
}
```

CODE SNIPPET E.2:
PanelOpener.cs full code

TextTimer.cs

```
/* This code was adapted from the following source:
    https://answers.unity.com/questions/970467/how-to-make-disappear
-a-gui-text-after-an-amount-o.html */
using UnityEngine;
using System.Collections;
using UnityEngine;
using System.Collections;

public class TextTimer : MonoBehaviour
{
    public float time = 20; //Seconds to read the text

    void Start()
    {
        Destroy(gameObject, time);
    }
}
```

CODE SNIPPET E.3:
TextTimer.cs full code

Active.cs

```
/* This code was adapted from the following source on stackoverflow;
    https://stackoverflow.com/questions/53566557/unity-3d-vr-hide-
and-show-model-on-controller-button-click */

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

// This script will allow us to hide the pointer when they are not in use
public class Active : MonoBehaviour
{
    // Initialise the hide/show with a button press
    public OVRInput.Button menu = OVRInput.Button.PrimaryHandTrigger;
    private bool activity;

    // Set the two pointer we want to disable/enable, these can be changed
        in Unity's inspector
    public GameObject ObjectToHide;
    public GameObject ObjectToHide2;

    void Start()
    {
        activity = false;
    }

    void Update()
    {
        ObjectToHide.SetActive(activity);

        // Here disables the first object

        if (OVRInput.Get(menu))
```

```
            // If button is pressed, activate the pointer
        {
            activity = true;
        }
        else
        {
            // Else, deactivate it

            activity = false;
        }

        ObjectToHide2.SetActive(activity);

        //Here disables the second object in a similar fashion

        if (OVRInput.Get(menu))
            // If button is pressed, activate the pointer
        {
            activity = true;
        }
        else
        {
            // Else, deactivate it

            activity = false;
        }
    }
}
```

CODE SNIPPET E.4:
Active.cs full code

CanvasPointer.cs

```
// This code was adapted from the following source by VR with Andrew;
    https://www.youtube.com/watch?v=i0QIqHHpvKw&t=263s

using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

public class CanvasPointer : MonoBehaviour
{
    // Default length, set as 3.0f as it was a length that wasn't too long
    public float defaultLength = 3.0f;

    // Assigns functionality to specific, overrideable components in the
        Unity Event
    public EventSystem eventSystem = null;
    // Module to assist with generic controller functions; dragging,
        pressing etc
    public StandaloneInputModule InputModule = null;

    // Takes two points in 3D space, and draws a line between the two
    private LineRenderer lineRenderer = null;

    private void Awake()
```

```
{
    lineRenderer = GetComponent<LineRenderer>();
}

// Uses the UpdateLength function to ensure the lineRenderer changes
    every frame
private void Update()
{
    UpdateLength();
}

private void UpdateLength()
{
    // Two positional vectors for the lineRenderer to work
    lineRenderer.SetPosition(0, transform.position);
    lineRenderer.SetPosition(1, GetEnd());
}

private Vector3 GetEnd()
{
    // This float checks for the Canvas' distance away from the pointer
    float distance = GetCanvasDistance();

    Vector3 endPosition = CalculateEnd(defaultLength);

    // if statement; if there is no object to hit, the distance will be 0
    if (distance != 0.0f)
    {
        endPosition = CalculateEnd(distance);
    }

    return endPosition;
}

private float GetCanvasDistance()
{
    // Get data
    PointerEventData eventData = new PointerEventData(eventSystem);
    eventData.position = InputModule.inputOverride.mousePosition;

    // Raycast using data
    List<RaycastResult> results = new List<RaycastResult>();
    eventSystem.RaycastAll(eventData, results);

    // Get closest
    RaycastResult closestResult = FindFirstRaycast(results);
    float distance = closestResult.distance;

    // Clamps the value; returns the given value if it is between the
        minimum and maximum range
    distance = Mathf.Clamp(distance, 0.0f, defaultLength);
    return distance;
}

// EventSystem.cs, a source code from Unity themselves
// BaseInputModule.cs, a source code from Unity themselves
private RaycastResult FindFirstRaycast(List<RaycastResult> results)
```

```
    {
        foreach (RaycastResult result in results)
        {
            // Go through each gameObject to detect if an object is being hit
                or not for every frame update
            if (!result.gameObject)
                continue;

            return result;
        }

        return new RaycastResult();
    }

    private Vector3 CalculateEnd(float length)
    {
        return transform.position + (transform.forward * length);
    }

}
```

CODE SNIPPET E.5:
CanvasPointer.cs full code

PhysicsPointer.cs

```
// This code was adapted from the following source by VR with Andrew;
    https://www.youtube.com/watch?v=rgTshoVCZVQ&t=435s

using System;
using UnityEngine;

public class PhysicsPointer : MonoBehaviour
{
    // Default length, set as 3.0f as it was a length that wasn't too long
    public float defaultLength = 3.0f;

    // Takes two points in 3D space, and draws a line between the two
    private LineRenderer lineRenderer = null;

    private void Awake()
    {
        lineRenderer = GetComponent<LineRenderer>();
    }

    private void Update()
    {
        // Uses the UpdateLength function to ensure the lineRenderer changes
            every frame
        UpdateLength();
    }

    private void UpdateLength()
    {
        // Two positional vectors for the lineRenderer to work
        lineRenderer.SetPosition(0, transform.position);
        lineRenderer.SetPosition(1, CalculateEnd());
```

```
    }

    // Vector3 used for direction and magnitude, but no position
    private Vector3 CalculateEnd()
    {
        // Create two separate final points
        RaycastHit hit = CreateForwardRaycast();
        Vector3 endPosition = DefaultEnd(defaultLength);

        // If something is hit, the ending hitpoint will be the object which
            is then stored
        if (hit.collider)
            endPosition = hit.point;

        // Otherwise, use our defaultLength

        return endPosition;
    }

    private RaycastHit CreateForwardRaycast()
    {
        RaycastHit hit;

        // Create a ray stemming from our controller/hand
        Ray ray = new Ray(transform.position, transform.forward);

        Physics.Raycast(ray, out hit, defaultLength);
        return hit;
    }

    // Default length for if nothing is hit
    private Vector3 DefaultEnd(float length)
    {
        return transform.position + (transform.forward * length);
    }
}
```

CODE SNIPPET E.6:
PhysicsPointer.cs full code

PointerEvents.cs

```
/* This code was adapted from the following source by VR with Andrew;
    https://github.com/C-Through/VR-SimplePointer/blob/master/Assets/
_SimplePointer/Scripts/PointerEvents.cs */
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Events;
using UnityEngine.EventSystems;

public class PointerEvents : MonoBehaviour, IPointerEnterHandler,
    IPointerExitHandler, IPointerDownHandler, IPointerUpHandler,
    IPointerClickHandler
{
    // Initialise basic colours for the different modes an object will enter
```

```
    // All default to white, however can all be changed in Unity's Inspector
        mode with the Color. colour wheel
    [SerializeField] private Color normalColor = Color.white;
    [SerializeField] private Color enterColor = Color.white;
    [SerializeField] private Color downColor = Color.white;
    [SerializeField] private UnityEvent OnClick = new UnityEvent();

    private MeshRenderer meshRenderer = null;

    private void Awake()
    {
        meshRenderer = GetComponent<MeshRenderer>();
    }

    public void OnPointerEnter(PointerEventData eventData)
    {
        // Changes object colour when pointer hovers over object
        meshRenderer.material.color = enterColor;
        print("Enter");
    }

    public void OnPointerExit(PointerEventData eventData)
    {
        // Changes object colour when pointer stops hovering over object
        // We will set this to 100% Opacity, as we do not want a colour
        meshRenderer.material.color = normalColor;
        print("Exit");
    }

    public void OnPointerDown(PointerEventData eventData)
    {
        // Colour changes when object is pressed
        meshRenderer.material.color = downColor;
        print("Down");
    }

    public void OnPointerUp(PointerEventData eventData)
    {
        // Changes object colour when pointer hovers over object after click
        meshRenderer.material.color = enterColor;
        print("Up");
    }

    public void OnPointerClick(PointerEventData eventData)
    {
        // Function will allow us to call a method later
        OnClick.Invoke();
        print("Click");
    }
}
```

CODE SNIPPET E.7:
PointerEvents.cs full code

VRInput.cs

```
// This code was adapted from the following source by VR with Andrew;
    https://www.youtube.com/watch?v=rgTshoVCZVQ&t=435s

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.EventSystems;

public class VRInput : BaseInput
{
    // Dummy Camera attached to the controller, to mimic a 3D Cursor
    public Camera eventCamera = null;

    // OVR Inputs for button
    public OVRInput.Button clickButton = OVRInput.Button.PrimaryIndexTrigger;
    // OVR Input to check which controller to use; set to all to choose in
        unity
    public OVRInput.Controller controller = OVRInput.Controller.All;


    protected override void Awake()
    {
        // Script inherits from BaseInput
        GetComponent<BaseInputModule>().inputOverride = this;
    }

    public override bool GetMouseButton(int button)
    {
        // Gets the value of whatever button you have on your controller
        return OVRInput.Get(clickButton, controller);
    }

    public override bool GetMouseButtonDown(int button)
    {
        // Gets the value of whatever button you have pressed
        return OVRInput.GetDown(clickButton, controller);
    }

    public override bool GetMouseButtonUp(int button)
    {
        // Gets the value of whatever button you have recently lifted
        return OVRInput.GetUp(clickButton, controller);
    }

    // This creates the position of the pointer
    public override Vector2 mousePosition
    {
        get
        {
            // This ensures the cursor comes from the center of the
                controller, /2 on both axis
            return new Vector2(eventCamera.pixelWidth / 2,
                eventCamera.pixelHeight / 2);
        }
    }
```

```
}
```

CODE SNIPPET E.8:
VRInput.cs full code

Drunk_PPP.cs

```
/* This code was adapted from the following sources:
    https://github.com/Unity-Technologies/PostProcessing/wiki/Writing-Custom
-Effects & https://www.youtube.com/watch?v=AcCDqH8LA9M&t= */

using System;
using UnityEngine;
using UnityEngine.Rendering.PostProcessing;

[Serializable]
[PostProcess(typeof(Drunk_PPPRenderer), PostProcessEvent.AfterStack,
    "Custom/Drunk_PPP")]
public sealed class Drunk_PPP : PostProcessEffectSettings
{
    // Create two new floats to change the screens distortion
    [Range(0f, 1f), Tooltip("Drunk Distortion effect intensity")]
    public FloatParameter amplitude = new FloatParameter { value = 0.5f };

    [Range(0f, 100f), Tooltip("Drunk Distortion effect frequency")]
    public FloatParameter frequency = new FloatParameter { value = 0.5f };

    // Create two new vectors to control the speed and size of the distortion
    [Tooltip("Speed of the Distortion only x and y used")]
    public Vector4Parameter speed = new Vector4Parameter { value = new
        Vector4(1, 1, 0, 0)};

    [Tooltip("Size of the Distortion only x and y used")]
    public Vector4Parameter size = new Vector4Parameter { value = new
        Vector4(1, 1, 0, 0) };
}

public sealed class Drunk_PPPRenderer : PostProcessEffectRenderer<Drunk_PPP>
{
    public override void Render(PostProcessRenderContext context)
    {
        var sheet =
            context.propertySheets.Get(Shader.Find("Custom/Drunk_PPP"));
        // Set float parameters
        sheet.properties.SetFloat("_Amplitude", settings.amplitude);
        sheet.properties.SetFloat("_Frequency", settings.frequency);

        // Set vector parameters
        sheet.properties.SetVector("_Speed", settings.speed);
        sheet.properties.SetVector("_Size", settings.size);

        context.command.BlitFullscreenTriangle(context.source,
            context.destination, sheet, 0);
    }
```

```
}
```

CODE SNIPPET E.9:
Drunk_PPP.cs full code

Drunk_PPP.shader

```
/* This code was adapted from the following sources:
    https://github.com/Unity-Technologies/PostProcessing/wiki/Writing
-Custom-Effects & https://www.youtube.com/watch?v=AcCDqH8LA9M&t= */

Shader "Custom/Drunk_PPP"
{
    HLSLINCLUDE

#include
    "Packages/com.unity.postprocessing/PostProcessing/Shaders/StdLib.hlsl"

    TEXTURE2D_SAMPLER2D(_MainTex, sampler_MainTex);
    float _Amplitude;
    float _Frequency;

    // Float2 stores the x and y of each pixel
    float2 distortion;

    float4 _Speed;
    float4 _Size;

    // Fragment shader loops over every picture, using x, y, z and w
    float4 Frag(VaryingsDefault i) : SV_Target
    {
        // Float2 stores the x and y of each pixel
        float2 uv = i.texcoord;

        // Different graphs types used to create a more unique pixel movement
        distortion.x = (sin((_Speed.x * _Time.y) + (uv.x * _Size.x) *
            _Frequency)) * _Amplitude;
        distortion.y = (cos((_Speed.y * _Time.y) + (uv.y * _Size.y) *
            _Frequency)) * _Amplitude;

        float4 color = SAMPLE_TEXTURE2D(_MainTex, sampler_MainTex, uv +
            distortion);

        //float luminance = dot(color.rgb, float3(0.2126729, 0.7151522,
            0.0721750));
        //color.rgb = lerp(color.rgb, luminance.xxx, _Blend.xxx);
        return color;
    }

        ENDHLSL

        SubShader
    {
        Cull Off ZWrite Off ZTest Always

            Pass
        {
```

```
            HLSLPROGRAM

                #pragma vertex VertDefault
                #pragma fragment Frag

            ENDHLSL
        }
    }
}
```

CODE SNIPPET E.10:
Drunk_PPP.shader full code

Distance.cs

```
// This code was adapted from the following source by BeepBoopIndie;
    https://www.youtube.com/watch?v=OMPV-duv25Q

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.Rendering.PostProcessing;

public class Distance : MonoBehaviour
{
    // Use GameObject so we can alter which objects to use within Unity
    public GameObject Object1;
    public GameObject Object2;
    public float distance_;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        // Tracks the distance between object 1 and object 2
        distance_ = Vector3.Distance(Object1.transform.position,
            Object2.transform.position);

        // if statement which tells us if the object is less than the given
            length, activate the postprocessing. Distance is altered to
            mimic putting drug to mouth
        if (distance_ < 0.25)
        {
            // Log update to verify if statement works
            Debug.Log("Drug has been taken");

            // Enable postprocessing effect
            GameObject.Find("CenterEyeAnchor").GetComponent<PostProcessLayer>
            ().enabled = true;
        }
    }
```

```
}
```

CODE SNIPPET E.11:
Distance.cs full code

Flicker.cs

```
// This code was adapted from the following sources:
    https://answers.unity.com/questions/11510/blinking-lights.html
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Flicker : MonoBehaviour
{

    Light testLight;
    public float minWaitTime;
    public float maxWaitTime;
    // Start is called before the first frame update
    void Start()
    {
        testLight = GetComponent<Light>();
        StartCoroutine(Flashing());
    }

    IEnumerator Flashing()
    {
        while (true)
        {
            yield return new WaitForSeconds(Random.Range(minWaitTime,
                maxWaitTime));
            testLight.enabled = !testLight.enabled;
        }
    }
}
```

CODE SNIPPET E.12:
Flicker.cs full code

HandlerButton.cs

```
// This code was adapted from the following source:
    https://answers.unity.com/questions/1735049/reset-text.html
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class HandlerButton : MonoBehaviour
{
    public void SetText(string text)
    {
        Text txt = transform.Find("Text").GetComponent<Text>();
        txt.text = text;
    }
```

```
}
```

CODE SNIPPET E.13:
Flicker.cs full code