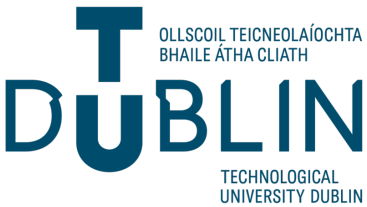
VR Transmission Simulation



Tadhg O'Rourke

Technological University Dublin

This report is submitted for the degree of
Bachelor of Computer Science

April 2021

Declaration

I hereby declare that the work described in this report is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other University.

Tadhg O'Rourke
April 2021

Acknowledgements

I would like to express my deepest thanks to my supervisor Paul Laird for giving me guidance throughout the development of this project, Bryan Duggan for giving me advice on the Unity game engine, my friends for their constant support, and my family for putting up with my constant complaining.

Abstract

With growing concern for viruses rising among the general population, methods to demonstrate the effects transmission are needed. This project attempts to show the potential use of extended reality for the education and research of viruses. Specifically, it investigates whether extended reality hardware can provide a base for the simulation of viruses.

To test this hypothesis, a proof of concept project to simulate viruses and viral transmission in virtual reality has been created. This simulation attempts to demonstrate the process of virus emission, spread, and transmission, as well as some precautions that can be taken to minimise the effects of transmission. To simulate virus particles, the project utilised the Unity game engine's approach to data-oriented design game development, the Data-Oriented Technology Stack. The results confirmed that it is possible to use XR technology to create such a simulation, however some technical limitations may apply.

The resulting simulation suggests that it is possible can be used XR technology to create simulations for education. However, with the current level of XR hardware and XR development suites, using performance first programming practices should be taken into consideration for projects development.

Table of contents

List of figures

List of tables

Listings

Chapter 1

Introduction

1.1 Project Overview

Accompanying the rise of the COVID-19 pandemic around the world is a rise in fear and anxiety (Rodríguez-Hidalgo *et al.*, 2020). This fear and anxiety can be related to misunderstanding how viruses spread (Roy *et al.*, 2020). As a result, the scientific and medical communities have been attempting to raise awareness of methods to prevent the further spread of dangerous viruses. One method being experimented with is virtual town halls (Fletcher *et al.*, 2020).

With the continuing development of extend reality technology, a new approach to education is being discovered. Andrews *et al.* (2019) explores how XR can be used for education in medical practices without compromising sterility. Additionally, as extended reality becomes more advanced, the possibility for use in research has become apparent. One such use for extended reality in research is the ability to conduct remote research, Ratcliffe *et al.* (2021) explains the potential benefits and drawbacks for such use.

Therefore, this project proposes the use of extended reality hardware as a method for both the raising of awareness and the research of transmission of viruses. To test this premise, a XR application to simulate the process and effects of viral transmission is defined. Additionally, some methods to prevent the spread of viruses and virus particles are also included. This project uses XR technology to allow for easy visualisation and interaction with virus particles in a real-time simulated environment. This VR transmission simulation project enables viral transmission to be demonstrated without extensive infrastructure, and with reduced risks to the user.

1.2 Project Objectives

To create this project, a set of objectives needs to be established to determine the effectiveness of the final product. These objectives will be based on achieving a product suitable for this projects topic, as well as testing the technology that was used during development. The following are the goals that this project aims to achieve at the end of development and testing processes:

- Create a extended reality simulation to show the effects of transmission
- Show how interacting with contaminated objects can spread can spread a virus
- Show how taking precautions can reduce the effects transmission
- Create a XR simulation that is comfortable for a user to experience
- Explore of the use of extended reality as a visualisation/simulation tool
- Explore a technology that could be used to increase performance on extended reality devices

1.3 Project Scope

This project aims to demonstrate how transmission of viruses. However, this project does not aim to simulate the effects of any specific virus, instead, this project hopes to show the generic effects and process of transmission. As such, the medical accuracy of the resulting simulation may not high, yet the final application can be further developed to have higher accuracy.

Initially, this project aimed to show how extended reality could be used as a method to demonstrate the effects of viruses, however, over time the resulting application became more of a proof of concept that this type of simulation could be used in experimentation and education in the future, due to the technology stack used still being a highly experimental technology.

Finally, as a combined result of the final product being a proof of concept, and the lack of time available for additional development, a Large environment with custom 3D model assets will not be included. The reasoning will be further detailed in Section 4.2.

1.4 Project Roadmap

Literature Review

This chapter examines the research that was conducted into the themes and technologies that were reviewed. Firstly, a background overview of viruses and viral transmission will be explored. After this, research into extended reality and the hardware that implement it will be outlined. Lastly, an analysis of the technologies and methodologies that could be used to develop this project.

Design

This chapter defines the design for the simulation that will be created over the course of this project. This chapter will show the original illustrations and diagrams that were created to define the functionality, as well as other images showing how this simulation will operate. The various technologies and design methodologies will also be outlined with the reasonings for their selection.

Development and Implementation

This chapter encompasses any development that was conducted to produce a final application. Firstly, the experimentation that was conducted with the technologies being used for development, and the outcomes of that experimentation. After this, some prototypes of the core features defined in the Design chapter. Finally, The development process of the final application will be defined.

System Validation

This chapter describes the testing the simulation underwent over the course of the development of this project. Any unit testing, play testing, user testing, and performance testings will be outlined, as well as the results of that testing.

Project Evaluation

This chapter explores the results of this project, by summarising the achieved goals, issues that were encountered as well as a comparison of the development timeline that was originally established. This chapter will also describe some of the future work that is planned for the simulation, and some conclusions that were drawn over the course of the project's development.

Chapter 2

Literature Review

2.1 Introduction

This chapter focuses on the research conducted in preparation for the design and development of this project. First, a delve into the background topic of this project. This research involves investigations into viruses, the transmission of pathogens, how respiratory mechanics perform transmission, and methods to limit transmission. Following this will be an examination of the technology that is extended reality, as well as its uses and issues. An analysis of extended reality hardware will also be conducted. penultimately, there will be an inspection of the development technologies that could be utilised for this project. Finally, an examination of game development techniques as they relate to extended reality.

2.2 Viruses, Transmission, and Particle Motion

2.2.1 Viruses

A virus is an infectious agent that lives inside the cells of an organism. This type of pathogen infects all forms of life, from microscopic bacteria to animals and plant life. Viruses are transferred via the process of transmission. Over time viruses have proved to be one of the most dangerous existences against life on this planet, killing thousands of people every year (Oldstone, 2020). In an attempt to reduce the dangers of these infectious foes, life on this planet has been coming ways to resist, through evolved methods such as natural immunities (Loebenstein and Carr, 2006) and manufactured methods such as drugs.

Transmission

Transmission is the medical term to denote the passing of a pathogen between individuals. As these pathogens enter the body, the individual can become infected and become a host to that pathogen. Depending on the scale of this transmission, pathogens can become epidemic level events (Noël *et al.*, 2009). Transmission can be performed in several ways and has therefore been broken into different categories based on their transmission method:

- Airborne transmission - transmission via small airborne virus particles
- Droplet transmission - transmission via larger airborne virus particles
- Direct physical transmission - transmission via direct physical contact with infected individuals
- Indirect physical transmission - transmission via contact with contaminated objects or surfaces
- Fecal-oral transmission - transmission via ingestion of contaminated food/water, or by unsanitary hygiene

To learn about how different pathogens perform transmission operations on different scales, many simulation experiments have been conducted (Chen *et al.*, 2020). These simulations can be used to predict certain factors or behaviours a virus could have. In addition to simulations about viruses themselves, transmission simulations can be used to predict the potential needs to fight a virus (Weissman *et al.*, 2020). Therefore, a real-time simulation, without the dangers of possible infection, has the potential to become a resource in the fight against transmission.

2.2.2 Respiratory Mechanisms

Breathing

Breathing is the process of inhalation and exhalation of air into and out of the lungs. The process of breathing provides the lungs with air enabling gas exchange to occur. As a result of gas exchange, the air exhaled by humans has a higher percentage of carbon dioxide, water, and aerosolized particles. This additional water and aerosolized particles can contain viruses (Xu *et al.*, 2012) and other pathogens that can spread around an environment (Scheuch, 2020). As a result of breathing expelling virus particles, the sampling of breath particulates has the potential to be used as a method of detection for respiratory viruses (Ladhani *et al.*, 2020).

Coughing and Sneezing

Coughing is the sudden expulsion of air from the lungs in an attempt to clear them. Coughing is used to clear the lungs and trachea of irritants and foreign objects. In this expulsion, viruses and bacteria are also discharged. Similarly, sneezing is a bodies reaction to irritation of the nasal mucosa. This irritation causes the body to emit a forcibly expel air through the nose and mouth to dislodge the irritant and remove it. Both coughing and sneezing create a large amount of force to remove irritants. A study by Kwon *et al.* (2012) show that the rate at which coughing and sneezing can expel particles can exceed a velocity of 10m/s. Bourouiba *et al.* (2014) details how coughing and sneezing can create clouds of droplets that can contain respiratory pathogens, once expelled. Figure 2.1 shows the cloud that these mechanisms can create.

2.2.3 Particle Motion

Droplet Dispersion and Evaporation

Once emitted from the body, water droplets and aerosol particles form a cloud. Over time these particles begin to disperse and evaporate. Liu *et al.* (2016) examines how these clouds can disperse and evaporate at different rates depending on humidity. The rate at which these clouds evaporate and disperse can also be influenced by the temperate and space available (Yan *et al.*, 2019). Liu *et al.* (2017) expands to discuss how these clouds can move in rooms with different levels of ventilation. The resulting simulation's respiratory mechanics will need enough virus particles to create a cloud of particles. The created particles will also need to have the ability to evaporate and disappear over time.

Brownian Motion

Brownian motion, also called Brownian movement, describes the random movement of microscopic particles that are suspended in a liquid or gas. This motion was first described by Robert Brown in 1828. This randomised motion is due to, as described by Einstein and Furth (1956), "the irregular thermal movements of the molecules of the liquid", such that liquids and gasses are constantly in motion and any particles that are light enough can be seen to move randomly as they flow within this medium. Brownian motion can describe the motion of individual particles in the air once emitted via this simulations respiratory mechanics.

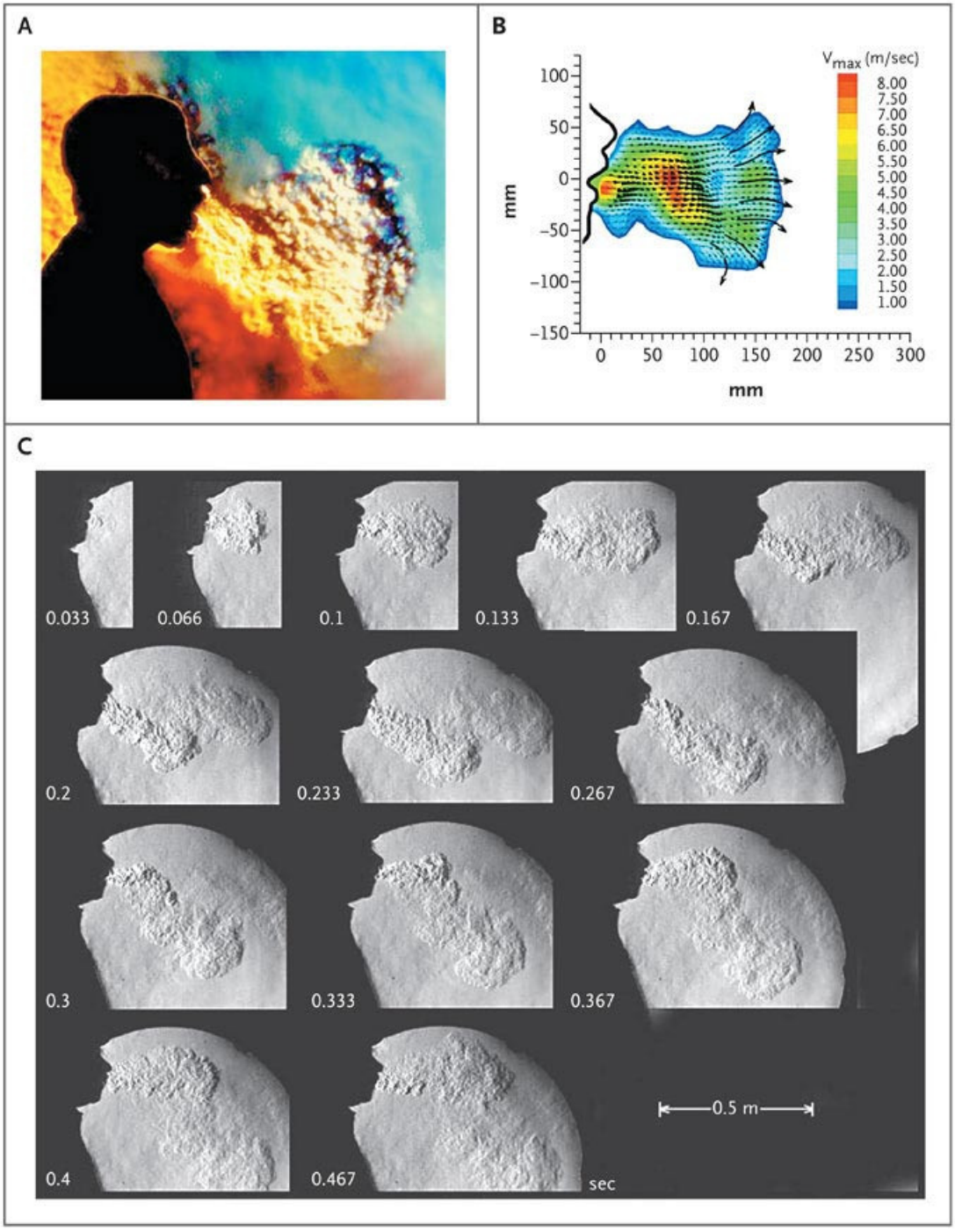


Fig. 2.1 Discharge caused by coughing (Tang and Settles, 2008)

Random Walk

A random walk is a stochastic process, where there is an equal probability for movement to occur in any direction in the available dimensions (Sinai, 1983). Brownian motion can be modelled via random walk algorithms (Harrison and Shepp, 1981). Most established random walk algorithms are limited by the number of steps and range in which they operate. As this project is attempting to simulate particles at runtime, the application of force will be done every frame. Additionally, the implemented random walk will be able to double-back on itself as non-simulated particles could.

2.2.4 Transmission Preventative Measures

In an attempt to reduce the transmission of viruses, several methods have been developed to either destroy dangerous microorganisms or lessen their effects. Depending on the situations different levels of measures can be taken to prevent this transmission. The level of measures will often rise with the level of threat a virus may have.

Disinfectant

Disinfectant, a chemical agent, is designed to destroy microorganisms or render them inactive. Although the process of disinfecting is not as effective as sterilisation, it is easier to perform. Disinfectant should be used to clean surfaces that are not critical to everyday use where virus particles may accumulate Rutala and Weber (2001). Disinfectant is already heavily integrated into healthcare facilities (Rutala and Weber, 2008).

Social Distancing

The practice of social distance is “designed to reduce interactions between people in a broader community” Wilder-Smith and Freedman (2020) in an attempt to counteract the spread of potentially epidemic level viruses.Matrajt and Leung (2020)’s model suggests social distancing measures could help “provide crucial time to increase healthcare capacity but must occur in conjunction with testing and contact tracing”. Additional time is essential in the fight against wide-spreading viruses. Social distancing has shown an estimated 65% reduction in new COVID-19 cases (McGrail *et al.*, 2020).

Face Masks

Face coverings for use in the medical environment, especially when performing surgery, first began to be used during the late 19^{th} Century. The rise in face masks was due to the

thought that "germs" were the cause of disease. In modern surgery, the use of face masks is ubiquitous. they provide protection to "theatre staff from patient-derived blood/bodily fluid splashes" (Da Zhou *et al.*, 2015) that could be harmful. Studies by Lindsley *et al.* (2014) and Wendling *et al.* (2020) have shown that face masks can the amount particles detected at a distances of less than 50cm by up to 96%.

2.3 Extended Reality

Extended Reality (XR) is a term used to describe the extension of a visual reality by way of intractable and wearable computer technology. XR is typically used as a super term for technology-based around the fusion of sensor networks and virtual worlds. The most often seen types of XR are:

- Augmented Reality (AR)
- Virtual Reality (VR)
- Mixed Reality (MR)

The use of XR is still in its infancy, however, the possibilities this technology possesses is already being explored. Hui-Wen Chuah (2018) explores some of the current research XR devices are being used for in the Information System (IS) world. This project aims to be part of that exploration into the potential use cases of XR technologies and to show how even simple simulations can be used to expand past human limitations.

2.3.1 Augmented Reality

"We define Augmented Reality as a real-time direct or indirect view of a physical real-world environment that has been enhanced/augmented by adding virtual computer-generated information" (Furht, 2011). AR can expand past the physical world adding in virtual objects that can be seen through AR-enabled devices. As the theme of this project is to make seen what cannot be naturally seen by the human eye, Augmented Reality would allow any space to become part of the simulation, and allow viral contamination to seen in normal environments.

2.3.2 Virtual Reality

As described by Burdea and Coiffet (2003), Virtual Reality "is a simulation in which computer graphics are used to create a realistic-looking world". These generated worlds provide the

ability to create real-time interactive, non-static, environments suitable for use in training, education, entertainment, and scientific study. VR provides the capabilities to create a real-time intractable world and allows for the interaction of those custom environments without much additional infrastructure. This ability is perfect for this project needs, as simulated microscopic virus particles can be created, and seen, in a safe manner, without the dangers of a virus spreading.

2.3.3 Mixed Reality

Mixed Reality is the combination of the virtual and real worlds to create settings, with physical and digital objects, that can be interacted with in real-time. These types of simulations are often for research and educational settings to expand upon the experience of existing scenarios, as stated by Hughes *et al.* (2005). This type of reality would allow for the expansion of senses past that of Augmented Reality and Virtual Reality, however, the scale required for this type of implementation would go beyond the scope of this project.

2.3.4 Head-Mounted Displays

Extended Reality most recognisable technology is the Head-Mounted Display (HMD). Depending on the HMD, small display optics allow the user to see the computer-generated reality in front of their eyes. AR HMDs use optical head-mounted displays (OHMD) to reflect projected images to allow the user to see them. VR HMDs additionally use inertial measurement units (IMU) to track any movements the user performs and translates them into motions inside the virtual simulation. HMD are often accompanied by a set of controllers which allow for interaction with the virtual world. The controllers track hand movement and hand orientation, enabling locomotion activities in the world. Some HMDs also have hand-tracking capabilities, where the headset can track the movements of hands without controllers by way of external cameras on the device, or through base stations. Under hand-tracking, gestures are used to manipulate the system in the same way buttons on a controller would.

2.3.5 Simulations

Visuals

As visuals are the fundamental theme of any XR technology, the more detailed the computer graphics are, the more immersive the simulation can become. “The goal of immersive virtual environments (VEs) was to let the user experience a computer-generated world as if it were real - producing a sense of presence, or ’being there,’ in the users mind” (Bowman and

McMahan, 2007). In following the point of this project, any visuals that will be created need to be accurate enough that the user will be able to associate what happens in the created simulation with what is happening outside the perception of the human eye.

Education

Visual learning, the learning of material through visual aids, is of one the primary learning styles a student may learn through. As virtual reality is often experienced through visuals, studies have been conducted to test the effectiveness of virtual reality as a learning aid. Helsel (1992) examines the possibility that "Virtual Reality has the potential to move education from its reliance on textbook abstractions", while Christou (1970) talks about how virtual reality "allows the student to experience scenarios and situations rather than imagining them". As more research has been done on using virtual reality in education, its use as a supplementary tool to allow students to experience what they are learning is only becoming more reasonable. By its use in education, this project has the potential to be used as an educational tool to demonstrate how taking steps can reduce the likelihood of viral transmission.

Industry

In industry, training a person on specialised hardware, under specific conditions, or in a strange environment, can be very expensive to achieve some semblance of what the real experience would feel like. By applying virtual reality in this area companies have been able to reduce the needed apparatus in favour of devices such as HMDs, and other XR technology, to allow trainees to visualise what they are expected to see. This conversion to XR related tools has begun already in the manufacturing industry (Mujber *et al.*, 2004). This type of training should soon be tested in the medical field according to Mantovani *et al.* (2003). The ability to adapt Virtual Reality to exotic environments, those not easily accessible is a core to the theme of this project.

Entertainment

As a form of entertainment, Extended Reality, by way of Virtual Reality is most often associated with video games. virtual reality has become a major part of the games industry, where it allows custom worlds to be created, with effects, not possible in the normal world, can be achieved and interacted with. According to Grandviewresearch.com (2020), virtual reality in the gaming market is expected to grow at a compound annual growth rate (CAGR) of 30.2% from 2020 to 2027. This expected expanse in virtual reality will only fuel the development of HMD devices, and other XR technologies, allowing more sophisticated

Fig. 2.2 VR omnidirectional treadmill (Virtuix Inc., 2017)

projects to be developed in both the games and entertainment industries. This expansion is telling that the technology could be applied in more situations than it is currently being used for.

2.3.6 Locomotion

Locomotion in the context of XR related technology refers to the ability monitor and move around in virtual world space. XR locomotion is most often encountered as a teleportation/blink feature (Bozgeyikli *et al.*, 2016), added to allow the user to traverse the world, however, locomotion extends beyond that. Many XR applications will include continuous movement and continuous turn features giving the user the ability to travel and turn via joystick. These additional movement actions can be disorientating leading to motion sickness or loss of balance. For expanded locomotion, there are omnidirectional treadmills that secure the user in place, but allow for locomotion movement in the virtual world. These types of machines are more accurate in tracking user movement but require expensive setups for use. For the scale of this project, simple locomotion functionality should be added.

2.3.7 Issues

With new technology comes challenges, Extended Reality is no exception to this. Extended Reality has been hit on multiple fronts by challenges ranging from the physical limitations of modern hardware, ethical issues surrounding its use in society, and the short and long effects on the human body. These issues will have to be closely monitored over the course of this project's development to keep in suggested practices to help limit adverse reactions.

Physical Limitations

As Extended Reality devices have been developed, the hardware necessary for tracking the user and performing the tasks for simulations has often been split between a HMDs and a linked computer (Gandhi and Patel, 2018). The HMD has hardware for displaying the simulation and tracking the user, while the linked computer handles processing through its CPU and GPU. As a result, a desire for more portable HMDs has risen, requiring hardware needed for the operation to be embedded into the device. However, the computational power of portable HMDs is gets limited by the size of the device, as a balance is needed between having the weight low enough to be usable and having the computational power necessary for operation. As portable HMDs also require the inclusion of a battery, an additional weight parameter is added to the device. The decision between a computer linked HMD, and a portable HMD will need to be made as this project will require high computational power output to track and manipulate particles during the simulations, but enough portability to be easily setup and used.

Ethical Concerns

With how XR technology functions, some ethical concerns have arisen over its use in society and its effects on the body. With little research into the short and long term effects, the extended use of XR devices has on the human body. When using XR HMDs it is not uncommon to experience effects of motion sickness or disorientation (Nichols and Patel, 2002). Having to deal with such effects, by using an XR device, can be considered ethically wrong. Additionally, with the rise in popularity of XR devices concerning society, the "Potential ethical implications of VR include physiological and cognitive impacts and behavioural and social dynamics" (Kenwright, 2018). The resulting simulation produced should try to have a short completion time, to limit ethical issues.

Medical concerns

Due to how Extended Reality HMDs mimic the illusion of reality by placing it close to the user's visual space, concern over a potential rise in eye strain and myopia. Turnbull and Phillips (2017) has found that in short term, no adverse effects on the binocular status of eyes, and Guo *et al.* (2017) found that visual fatigue rose with long-term focusing during the use of HMDs. Another common effect of ER reality devices is motion sickness and nausea. Motion sickness with XR devices is caused due to the conflicting signals the brain receives about self-movement in digital reality (Patrão *et al.*, 2020). The physical load of HMDs has also been a cause for concern among researchers. By having a large weight pulling the

head forward, stress can be caused on the neck of the user. Ito *et al.* (2019) has suggested a new design to help alleviate joint torque at the neck. As this projects themes associate with medical issues, great care will have to be taken the choosing of a HMD for the project to help mitigate the medical problems described above.

2.4 Extended Reality Head Mounted Displays

2.4.1 Valve Index

The Valve Index, released by the Valve Corporation (2020) in June 2019, is a computer connection requiring VR HMD. The Index boasts the highest display refresh rate, of 144Hz, for Virtual Reality Head Mounted Displays on the market. The Index functions by having the connected computer performs all the processing required for the device to operate, letting the HMD handle the display, audio, and tracking. By having the computer perform all required processing tasks the Index can outperform most other VR HMDs on the market in terms of performance. To track the movements of the user, the Index uses a system of base stations. These base stations are generally placed at opposing angles allowing the device to track user movement even the device controllers are outside the viewing area of the HMDs external cameras.

The Valve Index would provide the necessary processing power needed for this project to come to fruition, however, the infrastructure required will put a strain on the resources available. Additionally, the weight of the index, being 809g is on the higher end of HMDs, this could be burdensome on the user as explained in Section 2.3.7.

Fig. 2.3 Valve Index (Valve Corporation, 2020)

Fig. 2.4 Oculus Quest 2 (Facebook Technologies, LLC, 2020)

2.4.2 Oculus Quest 2

Facebook Technologies, LLC (2020)'s Oculus Quest 2, the successor to the very successful Oculus Quest, is a lightweight, and portable, all-in-one Virtual Reality HMD. The Quest 2 has a high max resolution of 1832x1920 pixels per eye, this allows the display to show very detailed images, which helps with eye fatigue. This device performs all required processing tasks on the device itself, segregating some of CPUs cores to the tracking of the user, leaving 3 cores to the developer for their applications. However, for additional processing power, the HMD can be connected to a computer for support. The Quest 2 uses a system of Infrared sensors on the side of the headset, as opposed base stations, to detect user movement when the device is active. The Quest 2 also supports Hand tracking. This gives greater control over the actions seen in the digital simulation, but this technology is still in development.

As the Oculus Quest 2 does not require additional infrastructure to function, the setup, and use, of the resulting simulation will be easier to test and evaluate. However, by this, any program developed for the device must be heavily optimised, as to not reduce the performance of the device. With this project, where a large number of particles are to be generated, the device's performance could struggle.

2.4.3 Microsoft HoloLens 2

The Microsoft Corporation (2019) HoloLens 2, is a Mixed Reality smart glasses device built to add digital graphics to the visible world. With an upgraded Holographic Processing Unit (HPU), the HoloLens 2 offers improved hologram stability over its predecessor. This HMD

Fig. 2.5 HoloLens 2 (Microsoft Corporation, 2019)

has a clear visor to see the real world, where holographic displays can be added. These holographic displays can be given commands via gestures the user performs.

The Hololens 2 has a limited field of view (FOV) of 52 degrees which would cause strain on the eyes, due to the limited possible movement available. By using a mixed reality HMD over VR, the project would not require additional scene building, this would greatly increase resources to work on other aspects of the project. However, the tracking of emitted particles would be more resource-intensive.

2.5 Development Technology

2.5.1 Unity

Unity is a games engine, developed by Unity Technologies (2005), that has the goal of making game development universally accessible. Unity has a simplistic user interface designed to help beginner developers quickly learn how to use the software while providing powerful features needed for professionals developers. Unity uses Microsoft Corporation (2000)'s C# as its main scripting language, enabling developers to create scripts for custom functionality. Unity supports a wide range of extended reality devices, providing APIs and asset packages providing features that can be built upon. The wide array of useful assets, APIs, and features Unity grants could supply many tools for development, which would make the development of this project much simpler.

2.5.2 Unreal Engine

The Unreal Engine, developed by Epic Games (1998), is another game engine designed with high-quality rendering and visuals in mind. The Unreal Engine also used in film making due to its ability to model virtual sets and special effects. As Unreal has a large emphasis on visuals, there are a wide array of powerful tools to help develop stunning graphics. The tools

would be very helpful in the creation of any visuals and particle systems this simulation will utilise. The Unreal Engine uses a system of 'Blueprint' system of visual scripting allowing non-programmers to implement the functionality. Functionality can also be extended through the use of C++. The Unreal Engine is also optimised for extended reality, providing extensive documentation and case studies on the platform. The combination of C++ and the optimised support for XR devices would enable greater control over the performance of the resulting simulation.

2.5.3 C++

C++, an object-oriented programming language based on the C programming language, originally conceived by Bjarne Stroustrup in 1979, with standards set out in Ellis and Stroustrup (1994), is a staple language in the games development industry. C++ enables developers to manage memory and bug prevention techniques to provide a major advantage in games development when performance is a necessity. Extended reality applications can be developed with no additional game engine by using C++ and Software Development Kits (SDK) provided by the HMDs. However, if this project was developed this way, the resulting simulation would be built for only one HMD, unlike the Unity Engine and the Unreal Engine which can be built to different HMDs. Additionally, Much more of the underlying functionality would have to be built which would take extra development resources.

2.5.4 Blender

Blender is a free open-source 3D asset creation suite created by the Blender Foundation (1998), from which 3D models can be designed, created, and animated. With a large amount of tutorial series available, Blender has become a staple software of indie game developers looking to create custom 3D assets for their games. Blender has integration features that permit for easy switching of model formats allowing the models to be imported into software such as Unity and the Unreal Engine. This fact would allow for the easy creation of any custom assets this project may demand.

2.5.5 Autodesk 3ds Max

Autodesk 3ds Max, by Autodesk Inc. (1988), is an industry-standard 3D computer graphics programming for making 3D animations, models, games, and images. Autodesk 3ds Max provides a professional toolset with procedural modelling techniques, fluid animation simulations, and character rigging. This software uses a subscription model, however, a

thirty-day free trial is possible for students, but the time scale of this project may prove that is insignificant to meet requirements.

2.6 Game Development

2.6.1 VR Game Design

The development of XR applications is often synonymous with the development of games due to it having similar resulting products. As a result of this, many game design principles can be applied to the development of extended reality simulations. Game design has been expanded on, especially in the area of virtual reality, as VR HMDs have become popular in recent years. VR game design gets expanded into an area such as simulation comfort, immersion, movement, and explores how to deal with some of the underlying issues of extended reality devices (See Section 2.3). Desurvire and Kreminski (2018) propose a set of guidelines that can be used to help design virtual reality games with better experiences.

2.6.2 Game Testing

Game testing is a software testing process that is used as part of games development to ensure video game quality. This style of testing aims to remove software bugs, and test products user experience. During the development of a product a Software Development Engineer in Test (SDET), or technical tester, build automated tests cases, using test frameworks, to ensure any code written performs its task correctly. For this project, the following software testing techniques are available for building automated test cases to test any written code:

- Unit Testing
- Integration Testing

Unit Testing

Unit testing is a method of building test cases to test the functionality of small separate code units. These unit tests guarantee the units fitness for use. These unit tests are executed with every execution of a program allowing developers to determine if functionality has changed. If the unit tests don't pass, any changes to the code have therefore changed the functionality. This method has however a disadvantage for each unit, many unit tests may be required to test all edge cases in the function. Additionally, writing unit tests, and fixing issues can be time-consuming (Daka and Fraser, 2014). The use of unit tests within this project may be

time-consuming but could help with ensuring each system operates as expected, especially with the major changes the simulation will have when new systems are added.

Integration Testing

Integration testing is the testing process of combining several modules to test them as a group. This stage of testing generally occurs after unit tests have been each modules unit tests have been certified. The well-standardised method of incorporating integration testing can however prove difficult (Leung and White, 1990). However with an effective strategy, and the reuse of previous tests, integration testing can become straightforward. With this project where large scales of particles, where many systems will have to work together, the process of integration testing would prove advantageous in ensuring these systems cooperate effectively without suffering many faults.

Play Testing

Another type of game test is a playtest. "Playtesting, or using play to guide game design, gives designers feedback about whether their game is meeting their goals and the player's expectations" (Choi *et al.*, 2016). With extended reality, this is especially important as some simulations can quickly problematic for the user playing (See section 6.2.3). This type of testing can be done from an empirical standpoint, as features being playtested should work as expected without bugs arising. Playtesting should be done over the course of a project, to build up a baseline for when new features are added.

2.6.3 Data-Oriented Design

Data-Oriented Design (DOD) is a program optimisation method around the efficient use of CPU caching. This optimisation approach, often used in video game development, manipulates a CPU's cache memory to only load the required data into cache memory rather than the entire object. This change in approach to memory management increases memory efficiency, by cache localisation, enabling quicker processing time by having operations performed on the data rather than the objects holding the data as Figure 2.6 shows.

This change in the programming paradigm would be highly beneficial to the performance of this project. The changing of programming style could prove difficult, as some XR devices may not support the change to Data-Oriented Design. Straume (2019) explores how DOD can be used in virtual reality applications, as would be done in this project.

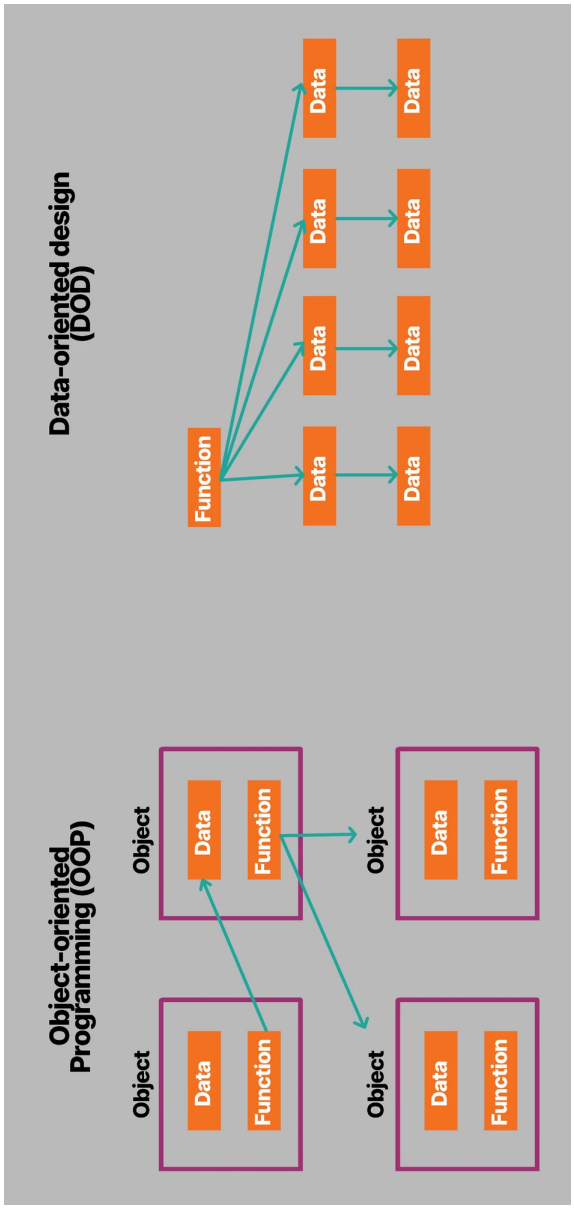


Fig. 2.6 Object-oriented programming versus data-oriented design (Unity Technologies, 2019)

Chapter 3

Design

3.1 Introduction

This chapter will make clear the design aspects of the application to be developed, this will be done through diagrams and discussion. First, as a result of the research conducted in the literature review chapter, the chosen XR hardware and development technologies will be stated, as well as the reasoning for their selection. Then, some of the original illustrations will be shown and discussed before a full outline of the features the simulation will have. Next, some system diagrams will be explored, detailing the underlying structure and operation of the application. Finally, The software development methodology chosen for the development phase of this project will be defined.

3.2 System Architecture

3.2.1 Oculus Quest 2

After a review of extended reality head-mounted displays, and the issues associated with them, a virtual reality HMD, the Oculus Quest 2 (see Section 2.4.2) was chosen as this projects core development platform. The choice of virtual reality, over other extended reality forms, was due to the more powerful hardware available VR HMDs. Additionally, the cost of VR HMDs is much less than mixed reality and augmented reality devices, which will make the resulting application more accessible to more people. The selection of the Quest 2 was made due to its portability. Having a portable HMD, that does not require additional infrastructure, it will make it easier for this project to be evaluated. As the Quest 2 is less powerful, additional steps will have to be taken for this simulation to perform at an acceptable level, but this will in the evaluation of modern XR devices.

3.2.2 Unity

The Unity games engine (see Section 2.5.1) has been selected as the underlying framework that this simulation will be built upon. Unity was chosen because of its extensive support for XR devices, as well as its extensive set of packages containing developer tools. The following tool packages will be used in this project to streamline the development of certain features:

- XR Plugin Management
- Oculus XR Plugin
- Input System
- Universal RP
- Test Framework
- Mathematics

The XR Plugin Management package provides the capability to develop onto XR devices, as well as manages the XR devices that a project can be built to, such as the Quest 2 through the Oculus XR Plugin package. The Universal RP (Render Pipeline) to a lightweight scriptable render pipeline, such allows for the creation of optimised graphics across a range of platforms. The Test Framework package provides the ability for the running of NUnit (Poole and Prouse, 2019) tests in the Edit and Play modes in Unity. The Input System and Mathematics packages provide additional tools for input defining and mathematical operations. These packages will be extremely helpful in the development of this simulation.

Data Oriented Technology Stack

The Unity game engine also provides a data-oriented design approach in their Data-Oriented Technology Stack (DOTS). DOTS is an experimental technology that intends to provide "Performance By Default" (Unity Technologies, 2019) through highly performant C# code. DOTS will allow for the instantiation, simulation, and manipulation of a large number of physics particles, that could not otherwise be done with vanilla Unity.

DOTS consists of three main components, the Entities package (Preview) providing the Entity Component System (ECS), the C# Job System providing easy to write multithreaded code, and the Burst Compiler which takes C# jobs to highly optimised machine code through a new LLVM-based backend compiler. ECS splits standard Unity into three principle parts, Entities, Components, and Systems. Entities are "things" that populate the scene/game, Components store any data attribute associated with an entity, and Systems which perform

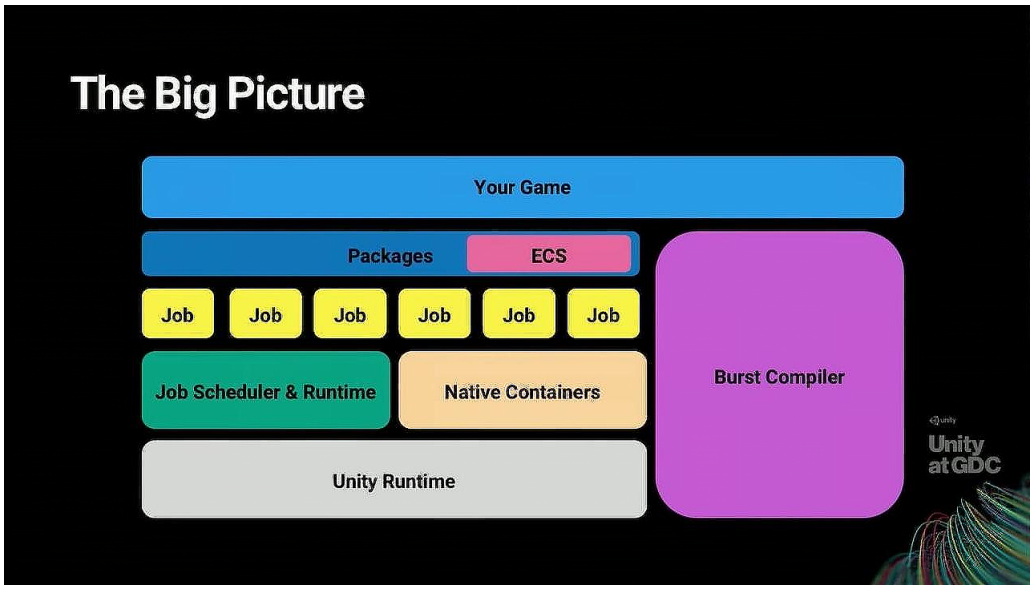


Fig. 3.1 Unity Data Oriented Technology Stack architecture

logical operations on components associated with certain entities. In addition to the core DOTS components, The following other DOTS packages will be used to expand functionality:

- DOTS Editor (Preview)
- DOTS Physics (Preview)
- DOTS Animation (Preview)

The DOTS Editor provides additional analysis and visualisation tools to view entities with their components and the systems which act upon them. DOTS Physics is Unity's custom physics engine built upon the DOTS framework. DOTS physics attempts to provide the base physics systems required for any project needs. Similarly, DOTS Animation is an animation system built upon the DOTS framework. Through the DOTS framework, this project will be able to simulate a large number of virus particles, in a highly performant manner.

3.2.3 Blender

Blender has been chosen as the 3D modelling tool for this project. Any custom assets that this project requires that are not available will be created through this 3D asset creation suite. The expected assets to be created are the face mask, the non-player actor, and the virus particle models. All other models can be acquired through the Unity Asset store or other markets.

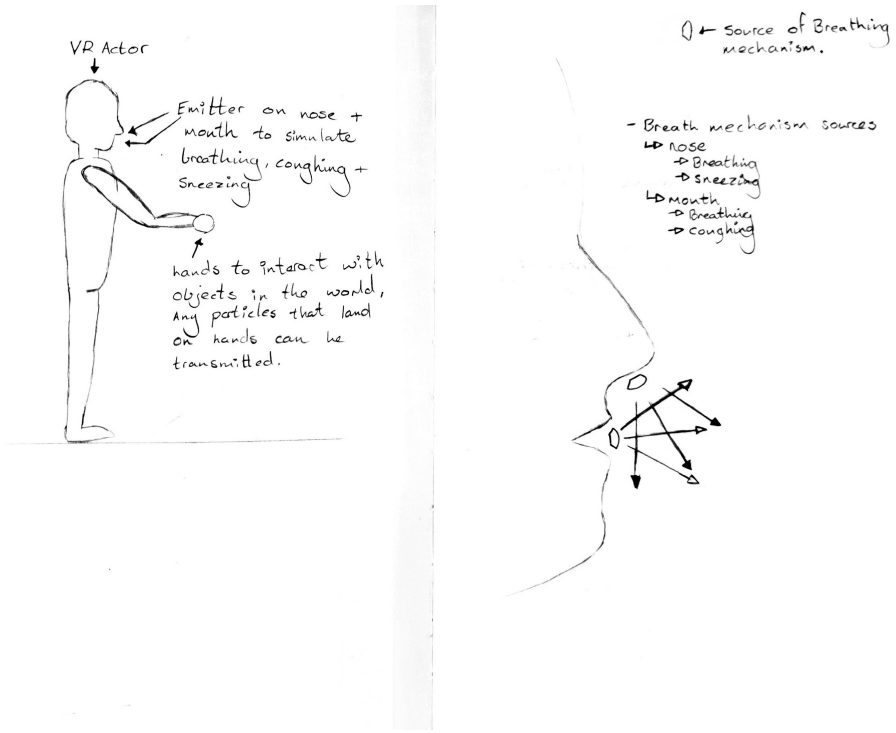


Fig. 3.2 VR actor and emissions source design

3.3 System Design

3.3.1 Low Fidelity Designs

During the initial stages of the development of this project, some low-fidelity illustrations were created to show the basic features this project will have. These illustrations will give a basic overview of how the core simulation mechanics will operate, as well as some of the potential visual effects that will be seen during use.

Figure 3.2 depicts both the XR actor and what locations some of the emissions sources. The user will have simulated hands to be able to grasp and lift an object in the environment. For the emission sources, the nose discharge point will provide the source point for the sneezing, while the mouth discharge point will act as a source for coughing and breathing respiratory mechanics.

Figure 3.3 show several different particles that will be created when by the respiratory mechanics. Three same virus particles are depicted. A virus particle will have the ability to float around in the air until it eventually dissipates. Additionally, there will be a visual effect that will help the user see when the respiratory mechanics are active. The figure also shows how the masks will work, by covering the nose and mouth the particles will be unable to flow through the air.

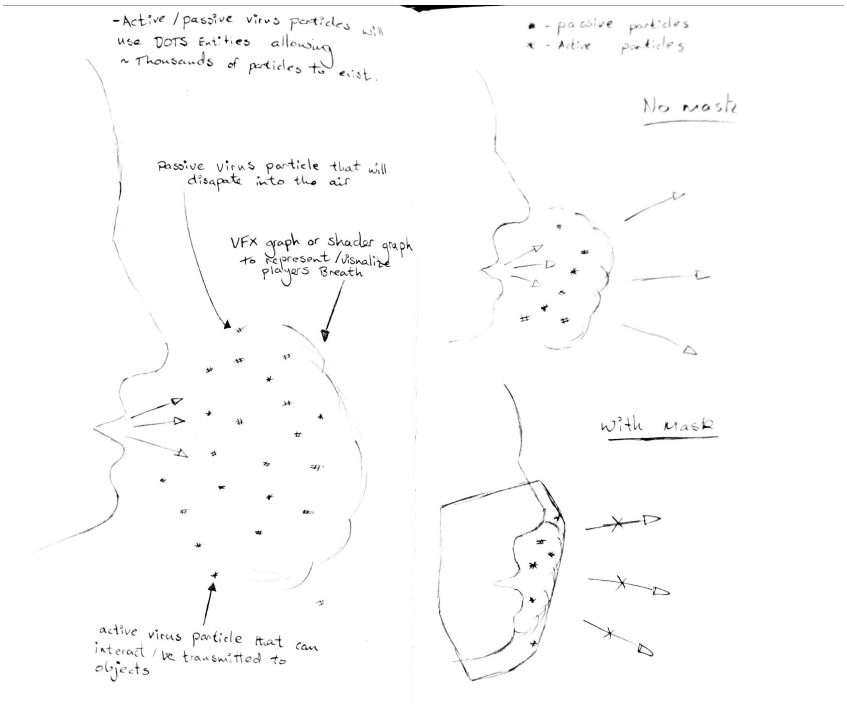


Fig. 3.3 Particles and mask interactions

Figure 3.4 further expands on how the masks will operate. If the mask does not cover the nose, any virus particles emitted will still be able to be released into the air, spreading the virus. The figure also demonstrates how when particles can stick to an object, and become stuck to it. Virus particles can be transferred if another object comes in contact with a stuck particle.

3.3.2 High Fidelity Designs

From the initial illustrations, some higher-fidelity images have been created to define a more concrete design of the simulation. These designs show a more accurate depiction of how features will operate, as well as how the simulation will be laid out.

Similar to Figure 3.3, Figure 3.5 shows the particles that will be created during respiratory mechanics spawning cycles. This diagram shows that particles can spawn from two places, the user's mouth and the user's nose. The mouth spawner will emit a cloud of particles horizontally, while the nose will send particles downwards. Figure 3.6 shows some example types that could be emitted via the respiratory mechanics.

The behaviour of face masks are shown in Figures 3.7 and 3.8. These diagrams show the options the user will have to cover the respiratory mechanic virus particle spawners. Once

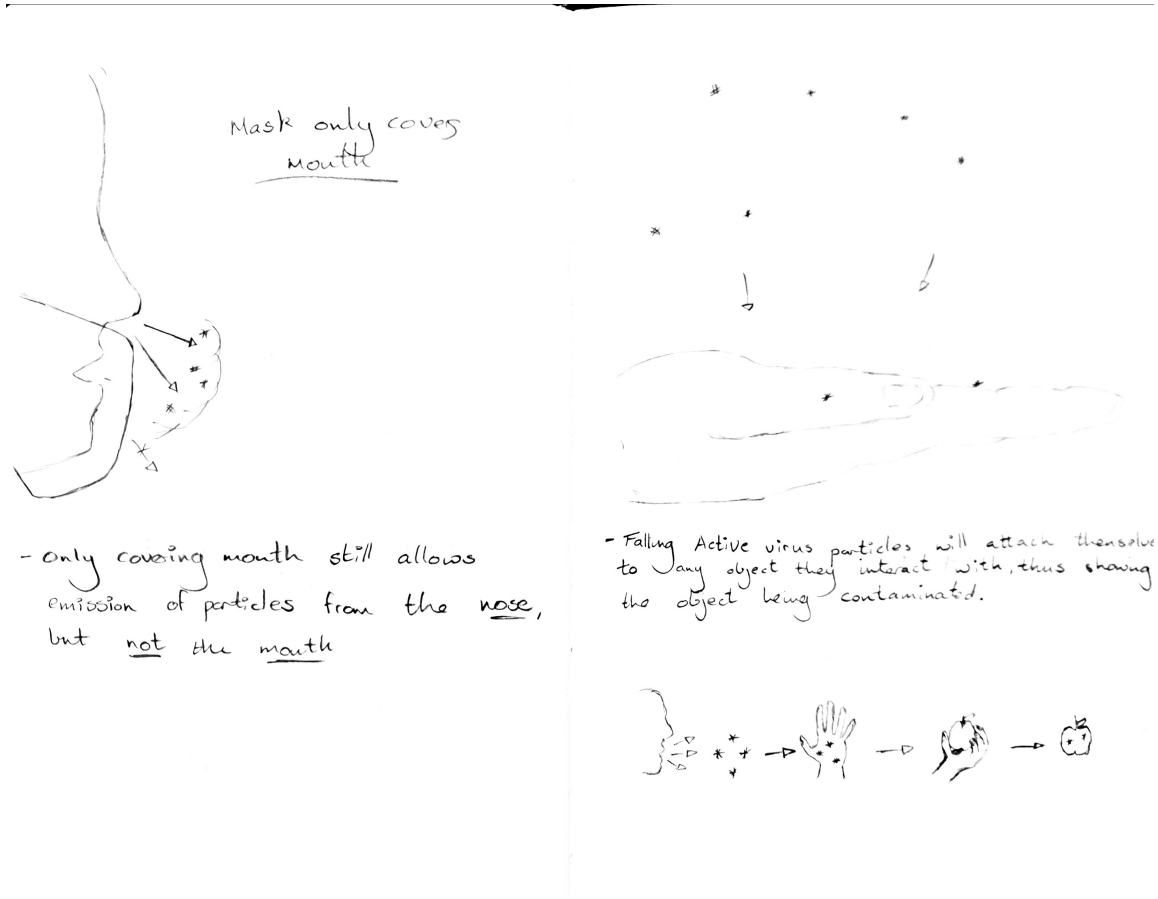


Fig. 3.4 Mask and particle interactions

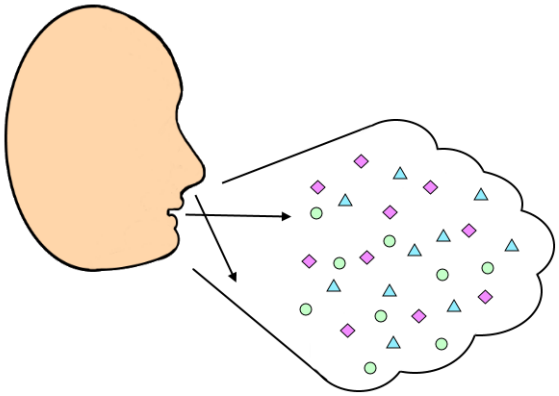


Fig. 3.5 Particle spawning

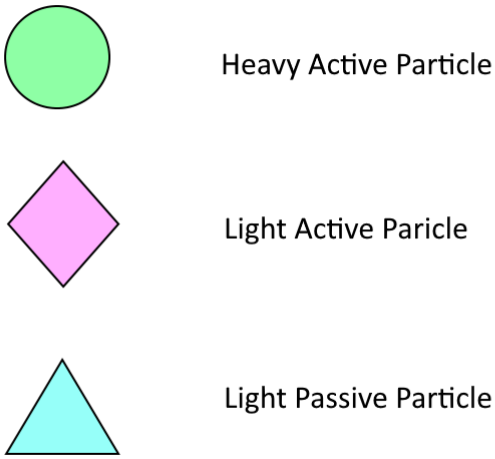


Fig. 3.6 Particle legend

the nose and mouth are fully covered, any virus particles will be contained onto the inside of the face mask.

The user will have the ability to interact with, and pickup, several objects throughout the environment. For any object user picks up, they need to hold their XR controller close enough to the object then activate the pickup button on the controller. The object will then be put attached to the hand model following the XR controller. This can be seen in Figure 3.9. If the user has any particles stuck to the hand model, on pickup the particles will transfer to the object. This transfer shows the effect of indirect-physical transmission (See Section 2.2.1).

Figure 3.11 shows the proposed room layout of the scene for the simulation. The user will be initially placed in the centre of the room. The player will be able to move around the scene using locomotion systems. There will be an NPC that will be moving around the room to show transmission in motion. There will also be a second NPC, this NPC will be able to interact with the user. Against two of the walls, there will be surfaces with objects the user can pick and interact with. On one of these surfaces will be the face mask the user can wear.

3.3.3 Design Clarifications

Extended Reality

As this project will utilise a virtual reality head-mounted display, several utility features will be included to make the simulation more usable and comfortable. As described in Section 2.3.6, some locomotion functionality will be included so the user may traverse the scene

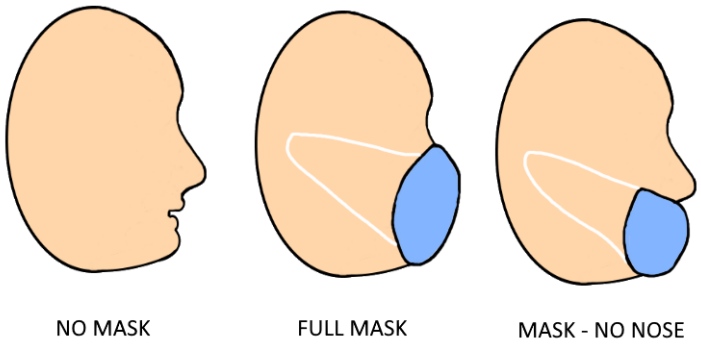


Fig. 3.7 Face covering options

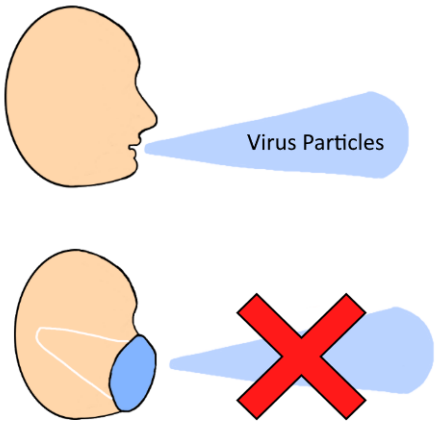


Fig. 3.8 Face mask behaviour

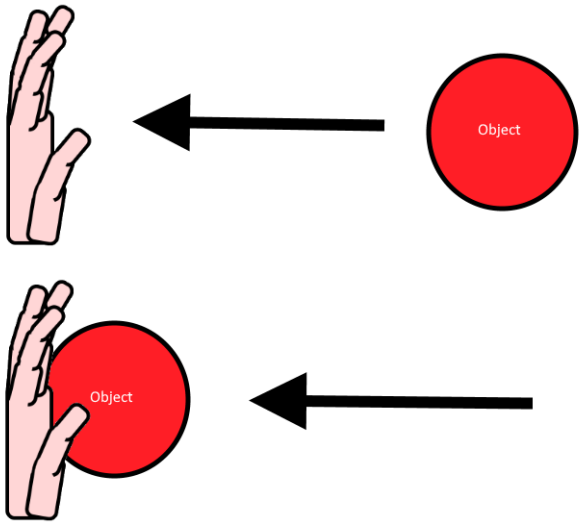


Fig. 3.9 Object pickup

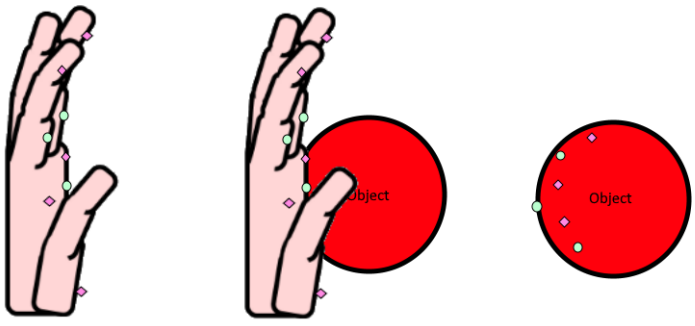


Fig. 3.10 Particle transmission

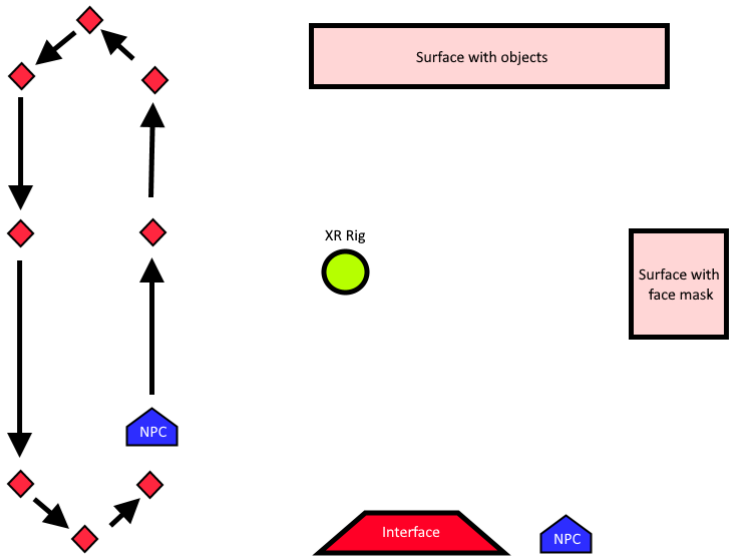


Fig. 3.11 Scene setup

to interact with objects, as a person would be able to do in a non-virtual environment. The locomotion features chosen are snap turning, continuous movement, and teleportation. This simulation will support the ability to pick up select objects in the scene. This is to show how virus particles that come in contact with objects can be transported, and further the transmission of the virus. The behaviours described will be accessed through the buttons and joysticks on the Oculus Quest 2 controllers. Figure 3.12, is the proposed layout of the behaviours described.

Respiratory Mechanics

To emit virus particles similar to authentic respiratory mechanics, spawners will be placed on the user. These spawners will be able to simulate the effects of coughing, sneezing, and breathing. Breathing spawners will be placed at both the nose and mouth. These respiratory mechanic spawners will be highly configurable to make it easier to adjust how certain particles are emitted. An example of this would be the ability to adjust the number of particles that are instantiated during spawning. A cloud visual effect will be played during the spawning cycle of a respiratory mechanic. This cloud is to demonstrate both the volume of mechanic and to show the other gasses and particles that this simulation is not displaying.

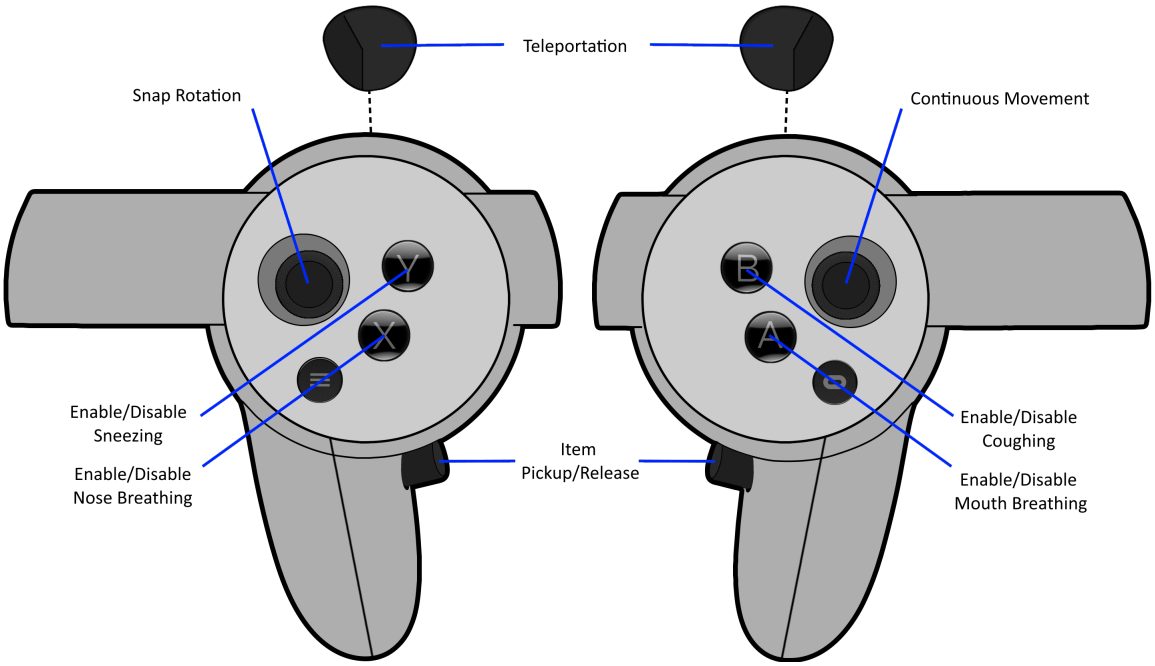


Fig. 3.12 Control layout

Virus Particles

Assorted virus particles will be simulated during play. The various virus particle types will be used to demonstrate how different particles operate depending on size and weight. Particles will come under various categories, each with behaviour distinct to that category. *Active* particles will have the ability to become stuck to objects. *Passive* particles can interact with the environment, but not become stuck. *Light* particles are those that will be under particle motion, having the ability to float throughout the scene. *Heavy* particles, will not be under particle motion, this is to demonstrate larger water particles in a breath.

Particle Motion

For virus particles in the *Light* category, particle motion will be applied to them. As Section 2.2.3 describes, this motion can be randomised in any direction. However, due to the scale of this project, a simplistic random walk algorithm will be used to apply the stochastic motion. Any *light* particle that becomes stuck to an object will no longer have the motion applied to it. The following random walk algorithm will be used to apply Brownian motion to the *light* virus particles:

Algorithm 1 Random walk algorithm

```
1:  procedure RANDOMWALK(P, s)                    ▷ Where P - particle array, s - step
2:      len ← length(P)                                          ▷ Get length of P
3:      loop
4:          for i ← 0 to len do
5:              pos ← P[i].position                              ▷ Get particle position
6:              step ← getRandom(s)          ▷ Get random step distance between 0 and s
7:              dir ← getPointOnUnitSphere()      ▷ Get random point on a unit sphere
8:              P[i].position ← pos + (dir * step)
9:          end for
10:     end loop
11: end procedure
```

Face mask

To show how precautions can be taken to reduce the likelihood of transmission the player will be able to wear a face mask. When worn, the face mask will stop any virus particles that are emitted from the respiratory mechanisms. The user will also have the ability to change how much of the face the mask covers. The face mask will have the options to be not worn, cover only the mouth, and cover the mouth and nose. By covering these areas, the respiratory mechanisms are blocked.

Non-Player Character Actor

Within the scene, there will be two Non-player Characters (NPC) who will have some basic artificial intelligence logic. These NPCs will have respiratory mechanic abilities, where the user can enable/disable the mechanic at an interface in the scene. One of these NPCs will be moving around in the scene, showing how particles spread as a person moves, the other NPC will be stationary, demonstrating the distance at which virus particles can travel and how having distance between people can stop viral transmission via social distancing. At the terminal for the stationary NPC, the user will have the ability to put a mask on the NPC. The user will be able to set how the NPC wears the mask, displaying how not covering the nose and mouth will let virus particles be emitted while breathing/sneezing/coughing. If close enough, the user will be able to shake hands and high-five with the NPC. This will show viral transmission between people.

3.4 System Diagrams

3.4.1 Use Case Diagrams

Figure 3.13 shows a use case diagram of the operations the user will be able to perform during the simulation.

- Enable/Disable respiratory mechanic
  - Breathing by mouth
  - Breathing by nose
  - Sneezing
  - Coughing
- Locomotion
  - Snap Turn
  - Continuous movement
  - Teleportation
- Interact with NPC
  - Put a mask on the NPC
  - Change mask positioning on NPC
  - Enable/disable NPC breathing mechanics
  - shake hands with the NPC
  - high-five with the NPC
- Interact with objects
  - pickup objects in the scene
  - put on mask
  - move mask to cover/uncover nose

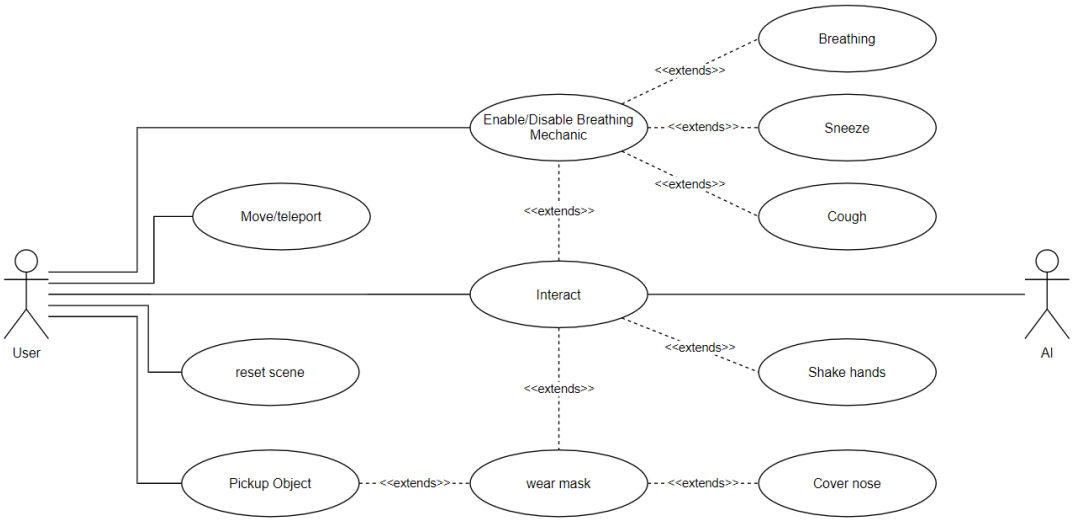


Fig. 3.13 Use case diagram

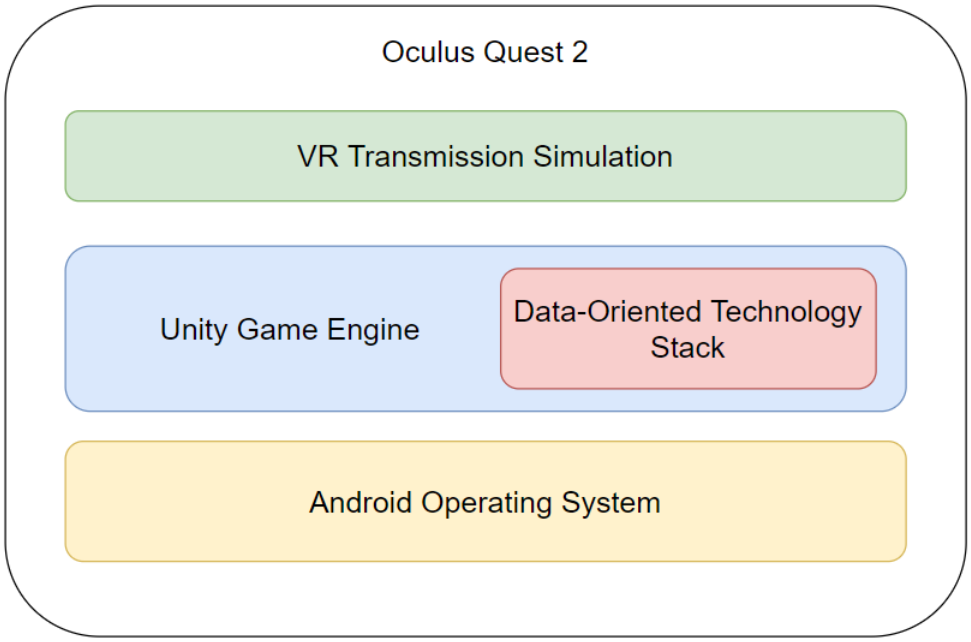


Fig. 3.14 Technical architecture diagram

3.4.2 Architecture Diagram

Figure 3.14 depicts the underlying architecture this simulation will have once built to the Oculus Quest 2 HMD. As the Quest 2 runs an Android based operating system, Unity build settings will have to be changed for Android based applications. The build settings will also be changed to help accommodate less powerful devices like a VR HMD. This diagram also shows the Unity Game Engine incorporating the DOTS framework for the simulation to be built upon.

3.4.3 Class Diagram

The class diagram shown in Figure 3.15 is a representation of how the planned virus particles will be related under Unity's Entity Component System. This diagram shows how the different types of virus particles will be constructed through the composition of several component tags. These component tags will help the system differentiate the different types of virus particles during simulation. Also shown is the BrownianMotionData component, which when connected to an entity will allow Brownian motion to be applied via the Brownian motion random walk system. Finally, this diagram shows a DecayingParticleData component. This component will be used to allow virus particles to dissipate into the environment, but also clean up the environment over time.

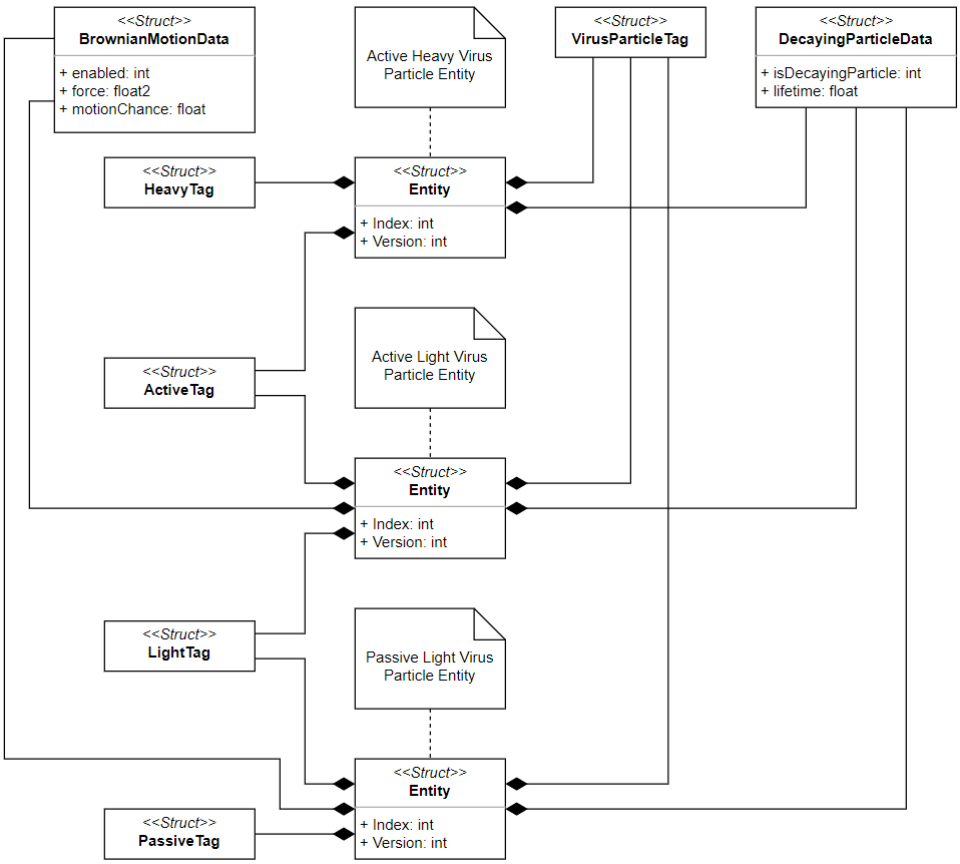


Fig. 3.15 Virus particle class diagram

3.5 Feature-Driven Development

The agile software development methodology of Feature Driven Development (FDD), originally described by Coad *et al.* (1999) and Palmer and Felsing (2001), is an incremental and iterative development process consisting of five basic activities:

- Develop overall model
- Build feature list
- Plan by feature
- Design by feature
- Build by feature

As the development of the overall model has already been discussed in Section 1.1, and Section 1.2, this step of the Feature Driven Development process will be skipped over.

3.5.1 Feature List

The following feature list depicts all planned systems within the application. features are split into separate categories depending on the priority of the component. For the needs of this project, the features will be divided into 3 tiers, primary, secondary, and tertiary features. Primary features are core features, fundamental to the operation of this simulation. Secondary features are behaviours required for this simulation. The tertiary tier is for additional features which would improve user experience during play.

Primary Features :

- Integrate Unity DOTS with XR
- Add controller interaction
- Particle spawning

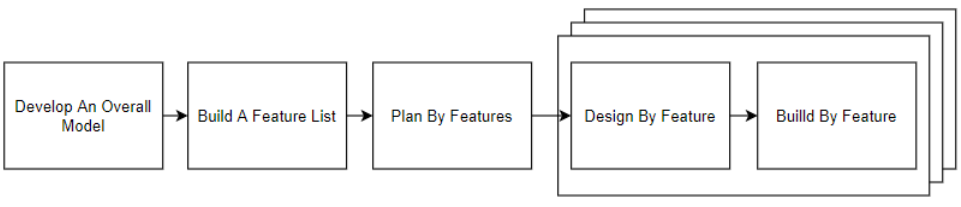


Fig. 3.16 Feature Driven Development model

Secondary Features :

- Particle sticking
- Particle motion
- Locomotion
- Face mask
- NPC respiratory mechanics
- NPC interaction
- Setup scene

Tertiary Features :

- Respiratory mechanic visual effects
- Animations
- Sound effects

3.5.2 Feature Plan

The feature plan takes the previously created Feature List (See Section 3.5.1) and assigns planning around each feature. For this project, the planning will be setting a priority within each tier, as well as estimating the length of time to implement the feature, as seen in Table 3.1.

Table 3.1 Feature plan

ID	Tier	Priority	Feature	Time Estimate - day(s)
1	Primary	High	Integrate Unity DOTS with XR	3
2	Primary	High	Add controller interaction	3
3	Primary	High	Particle spawning	4
4	Secondary	High	Particle sticking	4
5	Secondary	High	Particle Motion	3
6	Secondary	Medium	Locomotion	2
7	Secondary	Medium	Face mask	2
8	Secondary	Medium	NPC respiratory mechanics	2
9	Secondary	Medium	NPC interaction	3
10	Secondary	low	Setup Scene	1
11	Tertiary	low	Respiratory mechanic visual effects	2
12	Tertiary	low	Animations	2
13	Tertiary	low	Sound effects	1

3.6 Conclusion

This chapter explored the design of the proposed simulation. First, the technologies chosen for development were chosen and justified. Next, some diagrams describing the operation and some clarifications about the features this project will have were discussed. Following this, system diagrams explaining the structure of the project and the interactions the user can perform were examined. Finally, the chosen software development method of FDD, and an outline of how it will be applied, was detailed.

Chapter 4

Development and Implementation

4.1 Introduction

This chapter will deal with any development performed to create the resulting simulation. First off, some experiments that were conducted to learn about the development process of the technologies being used will be explored, including their results. Next, any prototype systems of the simulations core features will be analysed to ensure the resulting system will operate as expected. Finally, a delve into the development of the systems created for the final simulation will be defined, as well as the issues encountered with those systems.

4.2 Experimentation

4.2.1 DOTS

As Unity's Entity Component System is based around data-oriented design (see Section 2.6.3), the approach to writing code is different. Due to this, a simple test project was created to learn about the intricacies of the architectural pattern Unity was created. This simple project implemented a simple roll-a-ball game to examine how to set up a DOTS project, how to construct a DOTS scene, and how to create simple gameplay using entities, components, and systems. This simple project showed how to create basic functionality using the ECS pattern, as well.

Performance Comparison

To gauge the full extent to which the Data-Oriented Technology Stack can perform, several experiments were carried out. Firstly, Unity's DOTS Sample project (Unity Technologies,

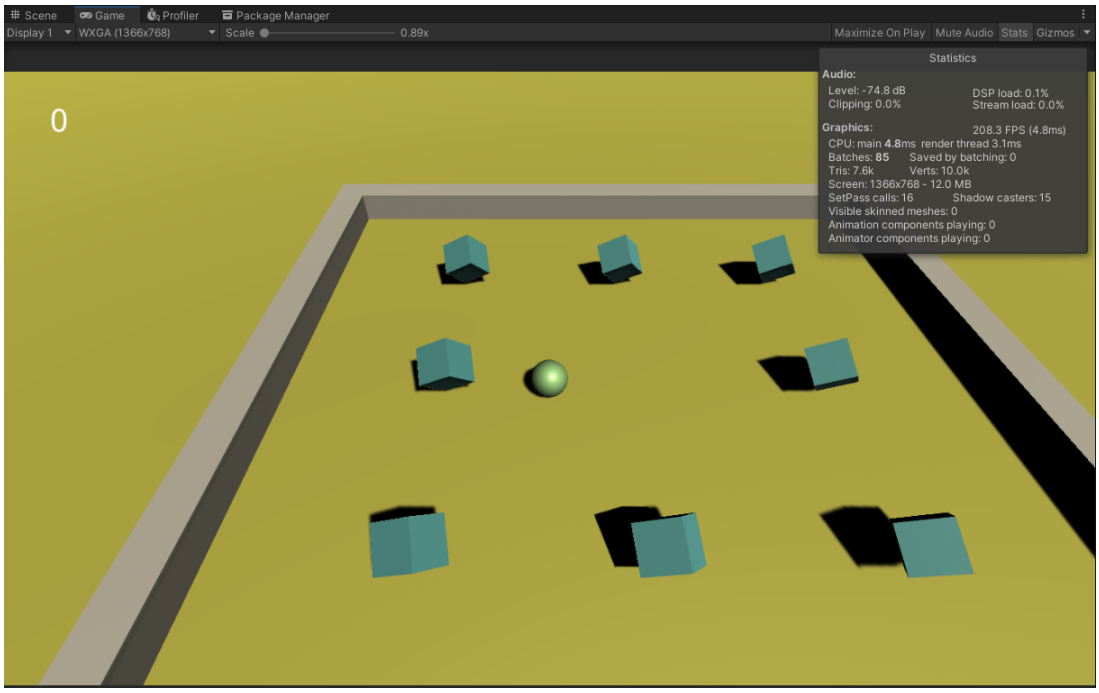


Fig. 4.1 Simple DOTS test project

2021) contains several examples of how to work with each component of the technology stack as well as some stress tests to demonstrate performance. With one example, Unity was able to simulate over 20,000 entities, with ECS systems activating on the entities, at a stable 50 frames per second (FPS) when scheduled, 55 FPS when scheduled in parallel, and 340 FPS when burst compiled. This example was altered to test the limits of DOTS. Under the altered example, DOTS could simulate 100000 entities at 65 FPS, 250000 entities at 32 FPS, and 9 FPS at 1000000 entities. This level of performance should provide the ability to simulate thousands of virus particles entities, under physics, on hardware with limited performance capabilities.

4.2.2 DOTS Physics

As Unity DOTS provides a separate physics engine to work with entities, an examination of how to work with the new components and systems was required. This was done by examining the DOTS sample projects Unity provides for developers (Unity Technologies, 2021). These samples demonstrated the setup of basic physics bodies on entities, how to handle collision events, and how to create custom authoring components for physics joints.

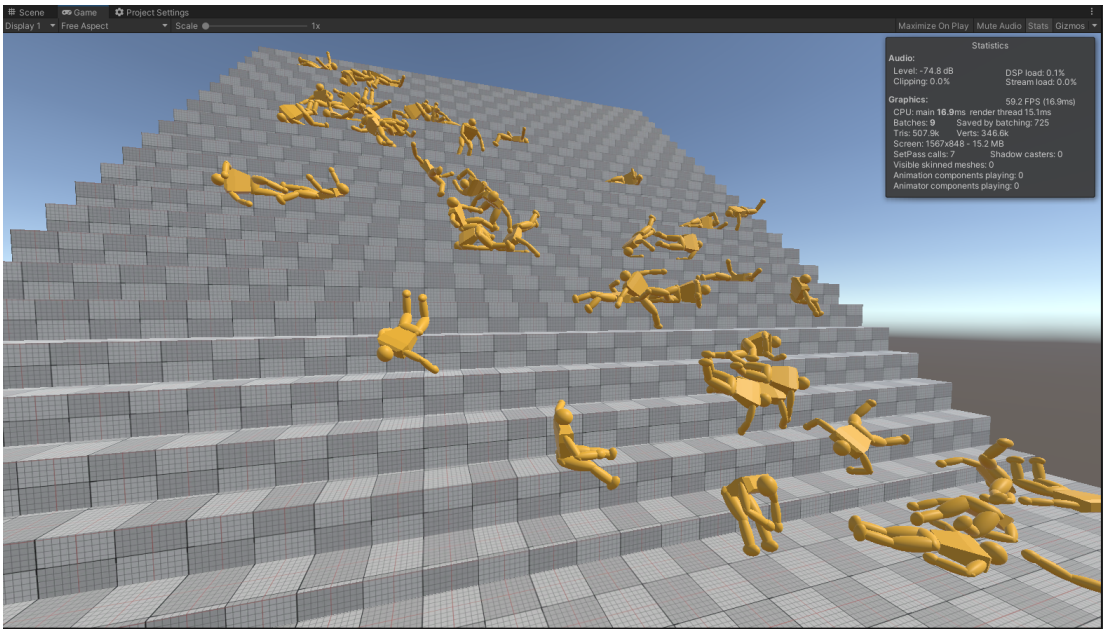


Fig. 4.2 DOTS Physics testing

4.2.3 XR Assets

As Unity provides several different methods to XR integration, each one needed to be tested to see which ones would be a good fit for this project. As this simulation is being developed on an Oculus Quest 2, the Oculus Integration package, acquired through the Asset Store, provides some Quest 2 specific features for Unity. The XR Plugin Management package was also tested, as it provides integration for a generic virtual reality HMD, but has little prebuilt VR functionality like other packages. The XR Plugin Management package with the Oculus plugin package was ultimately chosen over the Oculus Integration package as the additional features provided are incompatible with the DOTS framework.

4.2.4 VR and DOTS Compatibility

Due to the differences in backend implementation of standard GameObjects and ECS, GameObjects and entities do not interact. As a result, for GameObjects to influence entities, systems have to be created to convert GameObject data to ECS components. Therefore, to have the GameObject XR rig be able to interact with entities, an ECS version of the XR rig is needed. This system will use the transform of the GameObject XR rig to calculate the positions and rotations of an ECS rig. This link between a GameObject XR rig and an ECS rig will then allow the user to move around and interact with other entities in the scene. This link system will allow for any XR interactions that need to be performed by GameObjects to

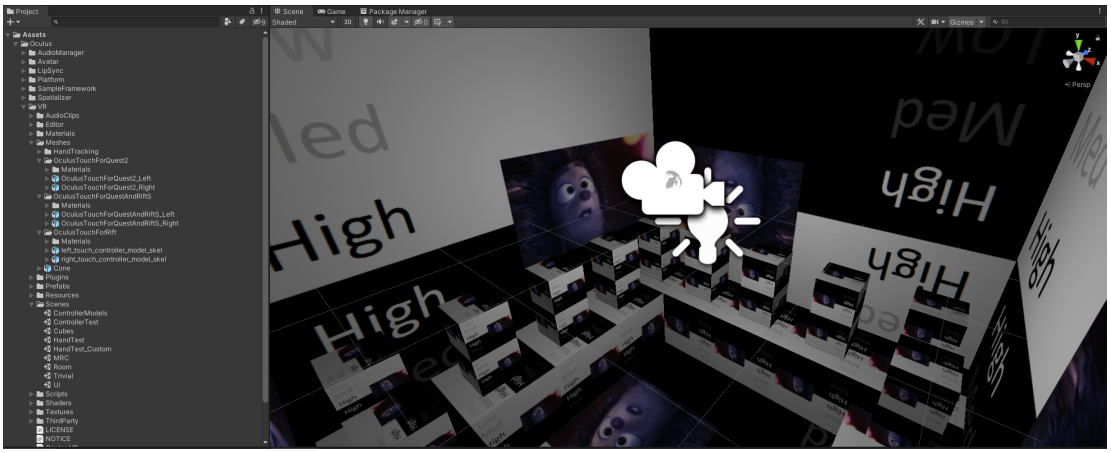


Fig. 4.3 Oculus Integration package testing

be done by the GameObject version of the rig, and any ECS interaction to be done by the ECS version of the XR rig.

4.2.5 Custom Assets

Over the course of experimenting with The DOTS framework, it became apparent that there would not be time to create custom 3D assets in Blender. The learning curve of the DOTS framework and the testing involved with integration with virtual reality technology exceeded expectations. There was no time was available to learn how to create any assets. Therefore, to mimic the functionality, and to provide the user with some type of visual feedback some temporary models will be created using Unity primitive models.

4.3 Prototyping

To test some of the functionality this simulation has, some prototypes were created based on the primary features stated in Section 3.5.1. These features are core to the success of this simulation so having a prototype of these features, and the systems they are dependant on, will help solidify each systems design.

4.3.1 XR DOTS Link

Since GameObjects do not interact with entities, An ECS version of an XR rig is required for the user to be able to manipulate the scene. Therefore, a prototype system to convert the movement of a user from a standard GameObject XR rig to a DOTS version of the rig

has been designed. This system matches the position and rotation of the head, left hand, and right hand on the GameObject Rig, to their respective entities on the ECS Rig. As the head, left hand, and right hand of each of the rigs are children of a root GameObject/Entity, any child will be required to get the local version of position and rotation instead of the global position. This process works from the perspective of the GameObject rig, by having a MonoBehaviour script on each of the GameObjects rigs nodes that hold a reference to its respective ECS rig entity. The position and rotation of each GameObject Rig node are then applied to that referenced entity in each frame. This link system functioned correctly, however, DOTS programming standards state that any operations that affect entities should be done by an ECS system. As a result, the final link system will use an ECS system for the link system.

```
private void SyncToTarget() {
    switch (syncType) {
        case SyncType.RootEntity:
            transform.position = _entityManager.GetComponentData<LocalToWorld>(
    TargetEntity).Position;
            transform.rotation = _entityManager.GetComponentData<LocalToWorld>(
    TargetEntity).Rotation;
            break;
        case SyncType.LocalEntity:
            _entityManager.SetComponentData(TargetEntity, new Translation() {
                Value = transform.localPosition
            });
            _entityManager.SetComponentData(TargetEntity, new Rotation() {
                Value = transform.localRotation
            });
            break;
    }
}
```

Listing 4.1 XR DOTS link prototype

4.3.2 Virus Particle Spawner

As a large number of entities will be spawned to represent virus particles emitted via the respiratory mechanics, a basic spawner was created, in conjunction with an ECS XR rig, to instantiate a prefab of a virus particle.

To create an entity version of a prefab, an authoring component had to be created to convert the GameObject prefab to an Entity. This Authoring component is placed on the GameObject prefab, and when the GameObjectConversionSystem is executed, an entity is created with ECS versions of the GameObject prefabs components.

```
public void Convert(Entity entity, EntityManager dstManager,
    GameObjectConversionSystem conversionSystem) {
    ParticleSpawnerData spawnerData = new ParticleSpawnerData() {
```

```
        Entity = conversionSystem.GetPrimaryEntity(particlePrefab),
    };
    dstManager.AddComponentData(entity, spawnerData);
}
```

Listing 4.2 Prefab conversion authoring

To spawn the virus particle entities, entities with the ParticleSpawnerData, InputData, and Translation components in their archetype are found. An EntityCommandBuffer is used to record an Instantiate() call, that instantiates the from the prefab, then a SetComponent() call to set the position of the new entity. This command buffer plays back the commands during the next frame, spawning the virus particles and setting their positions.

```
protected override void OnUpdate() {
    EntityCommandBuffer commandBuffer = _simulationEntityCommandBuffer.
    CreateCommandBuffer();

    Entities
        .WithName("ParticleSpawner")
        .ForEach((in ParticleSpawnerData spawner, in InputData inputData, in
    Translation translation) => {
            if (inputData.Trigger) {
                // create the new particle entity
                Entity newParticle = commandBuffer.Instantiate(spawner.Entity);
                commandBuffer.SetComponent(newParticle, new Translation() {
                    Value = translation.Value
                });
            }
        }).Schedule();

    _simulationEntityCommandBuffer.AddJobHandleForProducer(Dependency);
}
```

Listing 4.3 Particle spawning prototype

This spawning system proved the VR HMD could handle the large scale spawning of entities with physics bodies on them. The spawning system will be more complex in the final implementation, therefore a job will be used to record the spawning commands in parallel. This will greatly increase the speed at which the particle spawning can be performed.

4.3.3 Input Capture

As this project is using the Input System (New) package, a new Input Actions component, called "VRControls" was created to capture the Oculus Quest 2's controller inputs, as seen in Figure 4.4. Within this Input Actions, two action maps were created, each taking one of the Oculus Quest Controllers. The created Actions then had their action contexts defined within a MonoBehaviour. For the input actions to be used in a DOTS based simulation, a system then applied the collected actions onto entities with an Input Data Component.

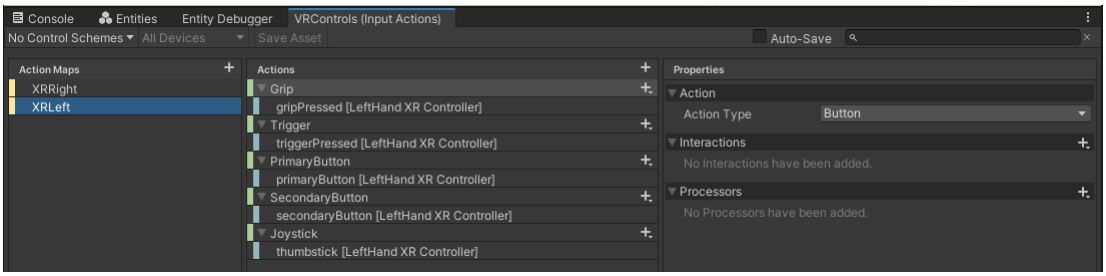


Fig. 4.4 Input Action Map

```
protected override void OnUpdate() {
    Entities.WithoutBurst().ForEach((ref InputData input) => {
        switch(input.Hand) {
            case ControllerHand.Left:
                input.Grip = _inputController.leftGrip;
                input.Trigger = _inputController.leftTrigger;
                input.PrimaryButton = _inputController.leftPrimaryBtn;
                input.SecondaryButton = _inputController.leftSecondaryBtn;
                input.Joystick = _inputController.leftJoystick;
                break;
            case ControllerHand.Right:
                input.Grip = _inputController.rightGrip;
                input.Trigger = _inputController.rightTrigger;
                input.PrimaryButton = _inputController.rightPrimaryBtn;
                input.SecondaryButton = _inputController.rightSecondaryBtn;
                input.Joystick = _inputController.rightJoystick;
                break;
            default:
                throw new ArgumentOutOfRangeException();
        }
    }).Run();
}
```

Listing 4.4 Input distribution prototype

This system will require all inputs to be distributed to entities that may not require all inputs. This system will be changed so that only the needed inputs will be distributed to components that require them. By having fewer data stored on each entity, memory usage will be greatly reduced.

4.4 Implementation

4.4.1 Quest 2 and Environment Setup

To develop applications on an Oculus Quest 2 HMD, the device must developer mode enabled. For the HMD to become development hardware, the device must belong to an organisation registered with Oculus. Once registered as part of an organisation, through the Oculus

mobile phone application, the devices developer mode can be enabled. This step can be seen completed in Figure 4.5a.

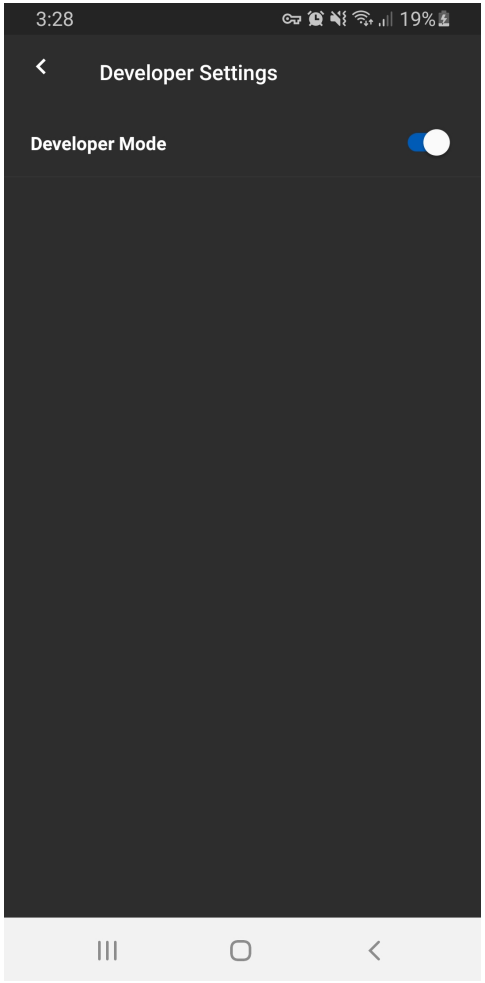
To set up Unity for the development of this project, a blank project was created. This project used the empty 3D project template which optimised the projects settings for 3D development. As this is a VR project, the projects Build Settings had to be altered to support the development of the Quest 2's Android operating systems. Various settings also needed to be adjusted to meet the requirements of the device. Under Player in the Project Settings the following adjustments need to be made:

Graphics APIs set to OpenGL instead of Vulcan

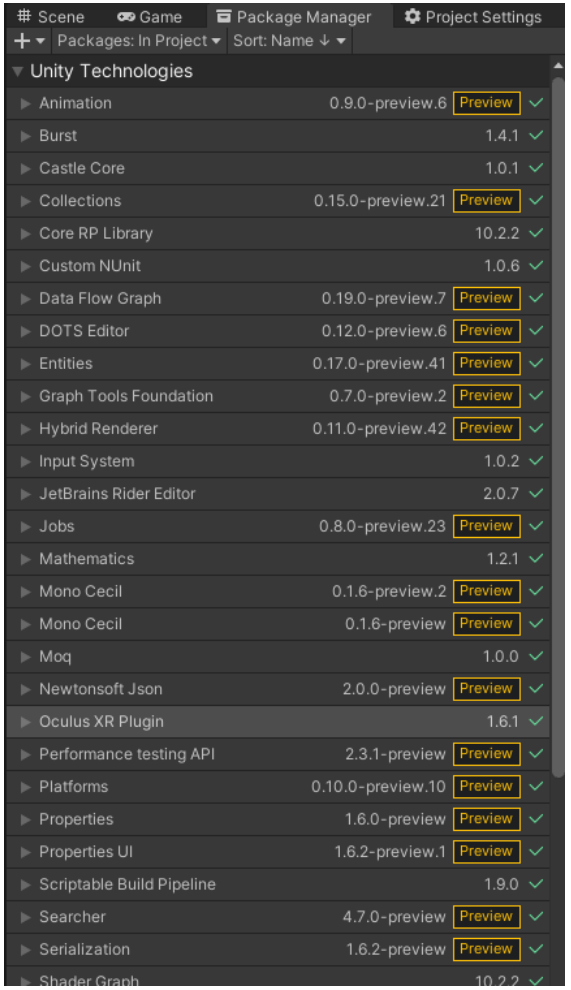
Minimum API Level set to Android 6.0 (API Level 23) or greater

Active Input handling set to Input system Package (New)

Additionally, the projects namespace was changed through Project Setting > Editor > Root Namespace. This will help with ensuring clean code is written. Once the Projects Settings have been adjusted, the packages this project will use are imported. The XR Plugin Management package can be enabled through the Project Settings menu. With the XR Plugin Management package enabled, the Oculus XR Plugin package could also be enabled. The packages listed in Section 3.2.2 also need to be imported. As some of the packages being used are preview packages, the Enable Preview Packages setting must be checked before import, this can be found under Project Settings > Package Manager > Advanced Settings.



(a) Quest 2 Developer Mode



(b) Importing Unity packages

Fig. 4.5 Environment setup

4.4.2 XR DOTS Link

Since entities do not interact with GameObjects, an ECS version of a VR rig is needed. As stated in Section 4.3.1, this system will be done from an ECS perspective. This ECS system functions by finding all GameObjects with RigType MonoBehaviour components, then applying the position and rotation from their Transform to the Translation and Rotation components on the ECS rig entities. As some the ECS rig needs to interact with other entities, each ECS rig node has PhysicsBody and PhysicsShape authoring components. By having these components the ECS rigs nodes will, over time, have gravity applied to them. To negate this, the PhysicsVelocity needs to be set to zero for each frame. As a consequence of an ECS system dealing with GameObjects, the conversion operation cannot be scheduled, and cannot be run with the burst compiler, therefore the Entities.ForEach() has the WithoutBurst() Lambda and run on the main thread using Run().

```
[UpdateInGroup(typeof(FixedStepSimulationSystemGroup))]
[UpdateBefore(typeof(BuildPhysicsWorld))]
public class RigConversionSystem : SystemBase {

    private RigType[] _rig;

    protected override void OnCreate() {
        // find all GameObject with a RigType component
        _rig = Object.FindObjectsOfType<RigType>();
    }

    protected override void OnUpdate() {
        Entities
            .WithName("RigConversion")
            .WithoutBurst()
            .ForEach((ref Translation translation, ref Rotation rotation, ref
    PhysicsVelocity velocity, in RigData rigData) => {
                // find the RigType with this entity
                foreach (var node in _rig) {
                    if (rigData.Type != node.type) continue;

                    // get rotation and position from Node
                    float3 nodePosition = node.transform.position;
                    quaternion nodeRotation = node.transform.rotation;

                    // apply the position and rotation to the entity
                    translation = new Translation() {Value = nodePosition};
                    rotation = new Rotation() {Value = nodeRotation};
                    velocity = new PhysicsVelocity();
                }
            }).Run();
    }
}
```

Listing 4.5 XR DOTS link

4.4.3 Input Handling

To handle inputs received performed on the Quest 2 controllers, two separate input Handling systems are used, One for GameObject MonoBehaviour scripts, and one for ECS systems. The MonoBehaviour input handler, InputHandler.cs, will collect inputs from the VRControls action map (Figure 4.4) and allow other MonoBehaviour features, like the locomotion system to access them. The ECS Input handler, InputHandlerSystem.cs, similarly collects inputs from the VRControls action map (See Listing 4.6), however, these inputs are then distributed to ECS components that require them. An example of this can be seen in Listing 4.7.

```
...

void VRControls.IXRRightActions.OnJoystickTouch(InputAction.CallbackContext context)
    => rightJoystickTouch = context.performed;
void VRControls.IXRRightActions.OnJoystickPress(InputAction.CallbackContext context)
    => rightJoystickPress = context.performed;
void VRControls.IXRRightActions.OnJoystick(InputAction.CallbackContext context) =>
    rightJoystick = context.ReadValue<Vector2>();

...

void VRControls.IXRLeftActions.OnGripPress(InputAction.CallbackContext context) =>
    leftGripPress = context.performed;
void VRControls.IXRLeftActions.OnJoystickPress(InputAction.CallbackContext context)
    => leftJoystickPress = context.performed;
void VRControls.IXRLeftActions.OnJoystick(InputAction.CallbackContext context) =>
    leftJoystick = context.ReadValue<Vector2>();
```

Listing 4.6 Input capturing

```
...
    Entities
        .WithName("BreathingMechanicInputDistribution")
        .WithoutBurst()
        .ForEach((Entity entity, ref BreathingMechanicInputData inputData) => {
            if (HasComponent<MouthBreathTag>(entity)) {
                inputData.Value = _rightPrimaryPress;
            }
            else if (HasComponent<CoughTag>(entity)) {
                inputData.Value = _rightSecondaryPress;
            }
            else if (HasComponent<NoseBreathTag>(entity)) {
                inputData.Value = _leftPrimaryPress;
            }
            else if (HasComponent<SneezeTag>(entity)) {
                inputData.Value = _leftSecondaryPress;
            }
        }).Run();
...
```

Listing 4.7 ECS input distribution

4.4.4 Particle Spawning

Each respiratory mechanic has a spawner to generate the virus particles in every frame. As each respiratory mechanic creates a large number of virus particle entities, the calculations and command recording is best left to a Job from the C# Job System. The Job type chosen for this operation is the IJobEntityBatch, which iterates over batches of entities executing operations on components an entity has. The spawning operation deals spawners having several different configurable settings, and several particle prefabs they can spawn, each with its own individual settings. Each spawner has settings defining the length of a spawning cycle, the change in range as the spawning cycle progresses, and the chance that a particle has to dissipate into the air. Each particle prefab has individual settings around the number of instances that can be spawned per frame, the initial linear force an instance can have, and a range of time until a particle can dissipate. These setting are used to calculate values for every virus particle prefab instances Translation, Rotation, CompositeScale, PhysicsVelocity, VirusParticleData, and DecayingParticleData components that a respiratory mechanic spawns. To ensure the spawning performs with high efficiency, the job is burst compiled and scheduled in parallel. Figure 4.6 show the number of particle entities that are created during a respiratory mechanics spawning cycle.

4.4.5 Particle Sticking

To imitate viral contamination, this simulation will attach the virus particle entities to object in the scene. By having the virus particles stick to an object, that object will be, in effect, contaminated. To detect when virus particles come into contact with objects the DOTS physics collision events will be used. These collision events are detected when two bodies with PhysicsCollider components intersect. Unlike the MonoBehaviour collision detection system, DOTS physics collision events cannot differentiate between the type of collision states (collision enter/stay/exit). As a result the DOTS sample project (Unity Technologies, 2021), includes a system to provide the same behaviour for DOTS physics. This system has been taken and modified for this project. This system will enable particles to detect if another object comes into contact with them, further contaminating the scene.

Sticking via joints

The first attempt at creating a system to stick particles to objects used the PhysicsJoint and PhysicsConstrainedBodyPair ECS components. The system worked based on every time a new enter collision event occurred, a third entity would be created, from an EnityArchetype, to hold the references to the particle and the collided object. This third entity contains the

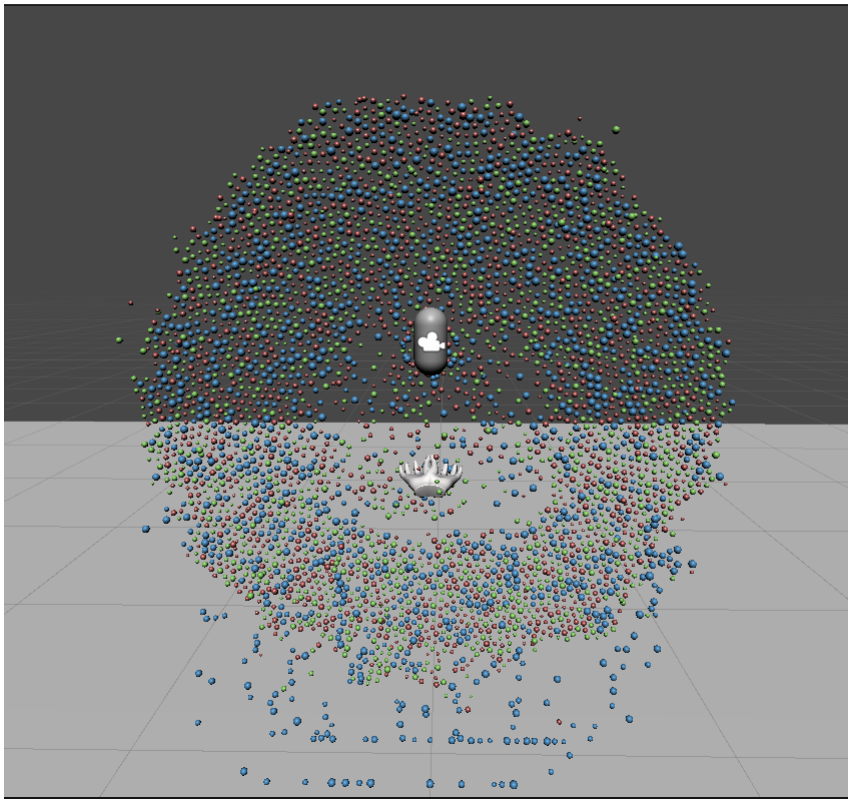


Fig. 4.6 Particle spawning

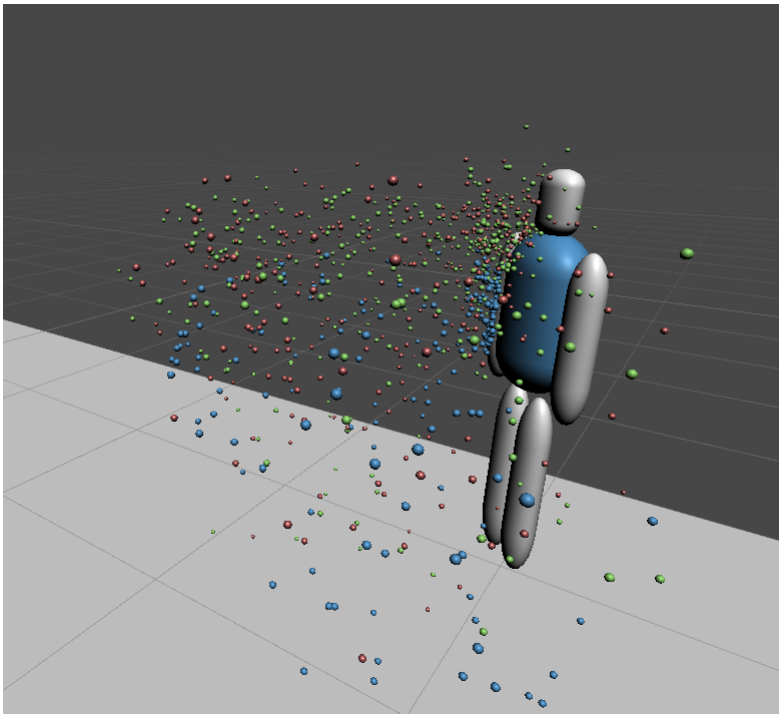


Fig. 4.7 Breathing test

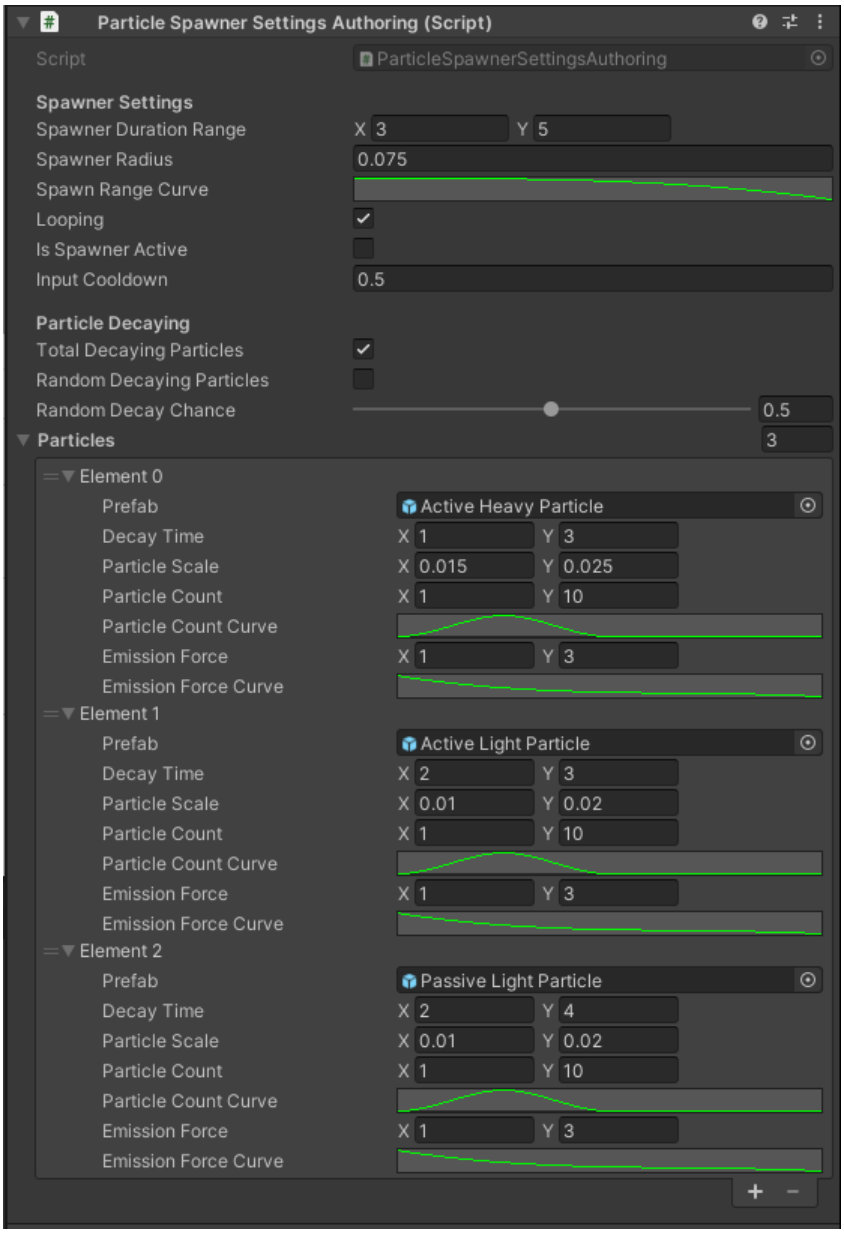


Fig. 4.8 Spawner settings

PhysicsJoint and PhysicsConstrainedBodyPair components. Whenever the particle becomes stuck, its velocity will set to zero, and the decay ability is disabled, and Brownian motion movement is disabled.

Unfortunately, the DOTS physics engine unable to support the number of particle joints on one entity and remain stable. Figure 4.9 shows Debug lines showing the connections between the connected bodies and the position of the joints. As more particles become connected to the object, the more other joints pull away causing the object to "glitch out" and fly away. As a result of the DOTS physics system being unable to support this level of joints, another method of sticking via parenting was explored.

```
Entities
.WithBurst()
.WithAll<VirusParticleData>()
.ForEach((Entity entity, ref PhysicsVelocity velocity, ref StickyParticleTag sticky,
    ref BrownianMotionData motionData, ref DecayingParticleData decayingParticleData,
     in DynamicBuffer<StatefulCollisionEvent> collisionBuffer,  in Translation
    translation, in Rotation rotation) => {
    if (sticky.value == Entity.Null) {
        if (collisionBuffer.IsEmpty) return;

        int collisionIndex = -1;
        for (int i = 0; i < collisionBuffer.Length; i++) {
            if (collisionBuffer[i].CollisionState == CollisionEventState.Enter) {
                collisionIndex = i;
                break;
            }
        }

        if (collisionIndex < 0 ) return;

        var other = collisionBuffer[collisionIndex].GetOtherCollisionEntity(entity);

        var otherTranslation = GetComponent<Translation>(other);
        var otherRotation = GetComponent<Rotation>(other);

        var entityRT = new RigidTransform(rotation.Value, translation.Value);
        var otherRT = new RigidTransform(otherRotation.Value, otherTranslation.Value)
    ;

        var entityBF = new BodyFrame(entityRT);
        var otherBF = new BodyFrame(otherRT);

        var joint = PhysicsJoint.CreateHinge(entityBF, otherBF);
        var cBP = new PhysicsConstrainedBodyPair(entity, other, false);

        var jointEntity = ecb.CreateEntity(jointArchetype);
        ecb.AddComponent(jointEntity, joint);
        ecb.AddComponent(jointEntity, cBP);
        ecb.SetComponent(other, new PhysicsVelocity());

        motionData.enabled = 0;
```

```
        decayingParticleData.isDecayingParticle = 0;
        sticky.value = jointEntity;
        velocity = new PhysicsVelocity();
    }
}).ScheduleParallel();
```

Listing 4.8 Sticking by PhysicsJoint components

Sticking via Parenting

ECS parenting operates by adding specific components to an entity that reference its parent's data. A child entity does not become directly linked to its parent but rather calculates its data relative to its parent's data. To have the virus particles become stuck to objects via parenting, a Parent component referencing the collided object, and a LocalToParent component. By adding these components, the virus particle entity becomes a child of the object. Any translation or rotation change will be applied to the virus particle to keep it in the same position and rotation to its parent. These changes are performed by the TransformSystem, provided by the ECS framework. Like with the joints system, the velocity of the particle is set to zero to stop the particle moving, the particle stops decaying, and Brownian motion is disabled on the particle.

In a similar fashion to the sticking via joints system, an unexpected behaviour arose. The stuck particles did not follow the movements of the parent object when moved, ultimately leaving particles stuck in the position they collided with the object, this can be seen in Figure 4.10. As other features need to implemented, this feature has been limited to zeroing out the velocity of the particle so that it can sit on the object it has collided with.

```
Entities
    .WithName("VirusParticleInitialSticking")
    .WithBurst()
    .WithAll<VirusParticleData, StickyParticleTag>()
    .WithNone<Parent, LocalToParent>()
    .ForEach((Entity entity, int entityInQueryIndex, ref PhysicsVelocity pv, ref
    DecayingParticleData decayData, ref BrownianMotionData motionData, in
    DynamicBuffer<StatefulCollisionEvent> collisionBuffer) => {
        if (collisionBuffer.IsEmpty) return;

        int collisionIndex = -1;
        for (int i = 0; i < collisionBuffer.Length; i++) {
            if (collisionBuffer[i].CollisionState == CollisionEventState.Enter) {
                collisionIndex = i;
                break;
            }
        }

        if (collisionIndex < 0 ) return;

        var other = collisionBuffer[collisionIndex].GetOtherCollisionEntity(entity);
```

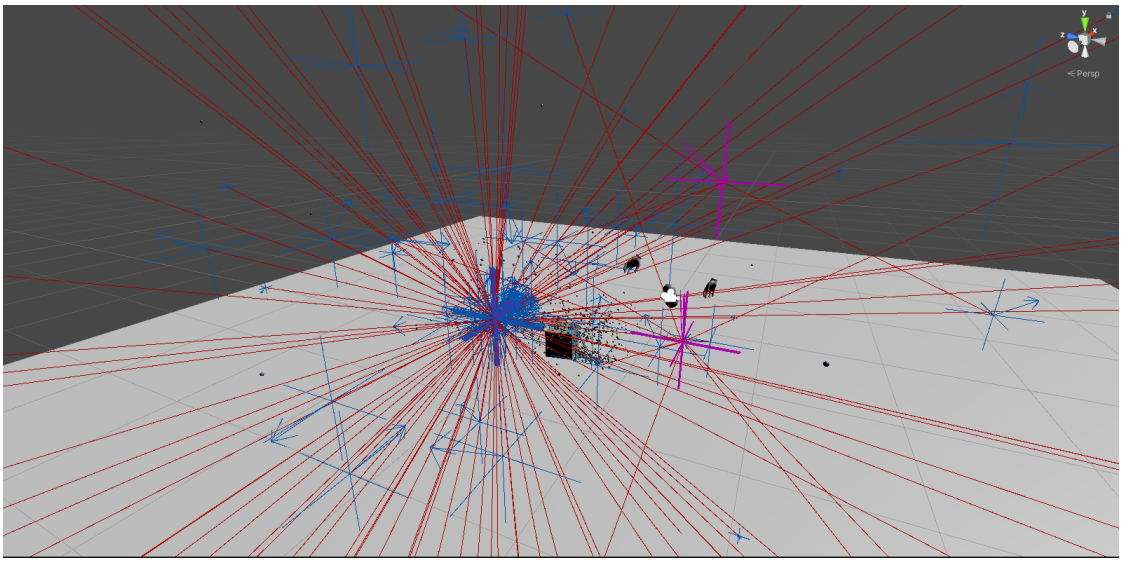


Fig. 4.9 Particle sticking by PhysicsJoint

```
20
21          motionData.enabled = 0;
22          decayData.isDecayingParticle = 0;
23          pv = new PhysicsVelocity();
24
25          ecb.RemoveComponent<PhysicsVelocity>(entity);
26          ecb.AddComponent(entity, new Parent() {Value = other});
27          ecb.AddComponent(entity, new LocalToParent() );
28      }).ScheduleParallel();
```

Listing 4.9 Sticking by parenting

4.4.6 Brownian Motion

To have the virus particle have a semi-realistic motion to their movements, Brownian motion will be applied to the particle entities. This Brownian motion will be applied by the random walk algorithm described by Algorithm 1. This random walk algorithm will be modified to work in an infinite space and to only apply motion if a particle has motion enabled on it. Only *Light* particles will have the motion applied to them. As stated in Section 4.4.5, any particles that have stuck to an object will no longer have Brownian motion applied to them.

```
1  Entities
2      .WithName("ApplyParticleBrownianMotion")
3      .WithBurst()
4      .WithAll<VirusParticleData>()
5      .ForEach((Entity entity, int entityInQueryIndex, int nativeThreadIndex, ref
       PhysicsVelocity pv, ref PhysicsMass pm, ref BrownianMotionData motionData, in
       Translation pos) => {
6          var random = randomArray[0];
```

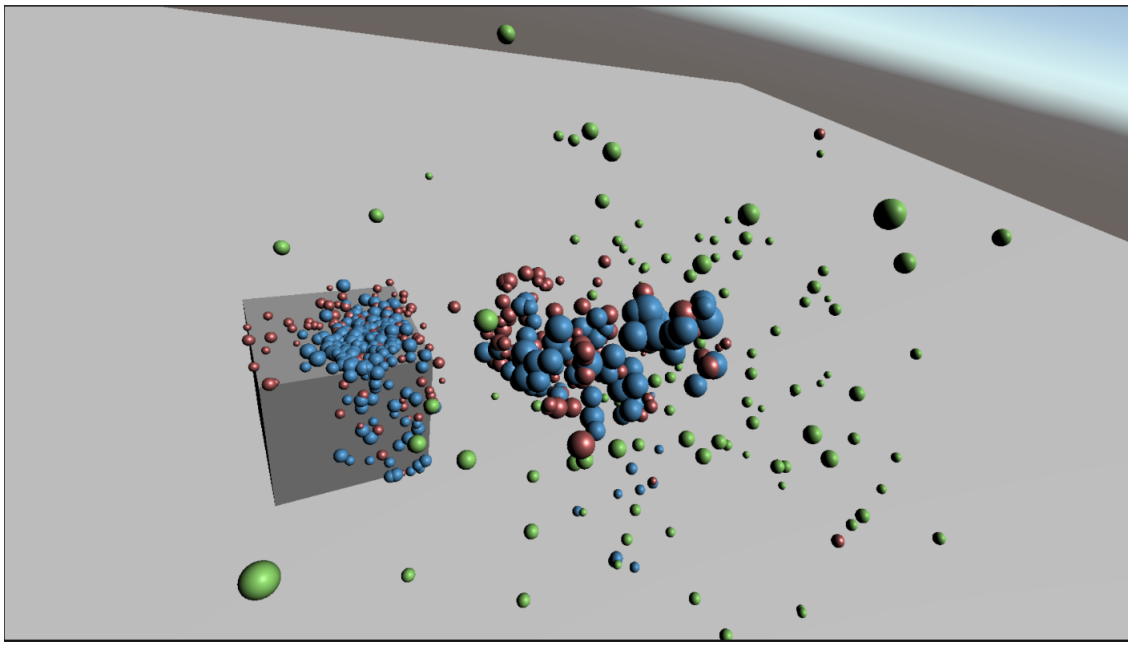


Fig. 4.10 Particle sticking by parenting

```
7
8           if (motionData.enabled == 1) {
9               float randomMotionChance = random.NextFloat();
10
11              if (randomMotionChance < motionData.motionChance) {
12                  float randomForce = random.NextFloat(motionData.force.x, motionData.
    force.y);
13                  var randomDirection = MathUtil.PointOnUnitSphere(ref random);
14                  pv.ApplyLinearImpulse(pm, randomDirection * randomForce);
15              }
16          }
17
18          randomArray[0] = random;
19      }).ScheduleParallel();
```

Listing 4.10 Brownian motion

4.4.7 Face Mask

To have a face mask that can stop emitted virus particles as are spawned, the face mask model was placed in front of the respiratory mechanic spawners. The virus particles will collide with the face mask and stick to them via the particle sticking system described in Section 4.4.5. This mask can be enabled and disabled by the user by clicking the mask enable/disable buttons defined in the control scheme (see Section3.12). To show how not covering all breath sources will allow the spewing particles while wearing a mask, the portion of the mask

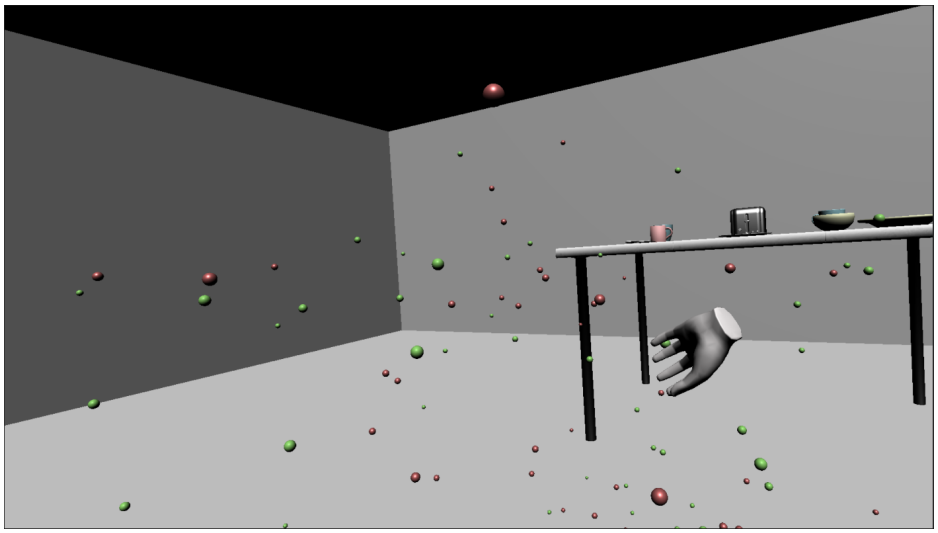


Fig. 4.11 Brownian motion

covering the nose respiratory mechanic spawners can be enabled and disabled. Unfortunately, as a result of the incomplete implementation of the particle sticking system, the mask does not contain the virus particles.

```
Entities
    .WithName("FaceMaskController")
    .WithBurst()
    .ForEach((Entity entity, int entityInQueryIndex, ref FaceMaskData maskData, in
    FaceMaskInput maskInput) => {

        var mask = maskData.faceMask;
        var nose = maskData.maskNose;

        if (maskInput.enableMask == 1 && time >= maskData.lastInputTime + maskData.
    inputCooldown) {
            maskData.isMaskEnabled = ( maskData.isMaskEnabled == 0 ) ? 1 : 0;
            maskData.lastInputTime = time;
        }
        else if (maskInput.enableNose == 1 && time >= maskData.lastInputTime +
    maskData.inputCooldown) {
            maskData.isNoseEnabled = ( maskData.isNoseEnabled == 0 ) ? 1 : 0;
            maskData.lastInputTime = time;
        }

        // enable/disable mask on input change
        if (HasComponent<Disabled>(mask) && maskData.isMaskEnabled == 1) {
            ecb.RemoveComponent<Disabled>(entityInQueryIndex, mask);
        }
        else if (!HasComponent<Disabled>(mask) && maskData.isMaskEnabled == 0) {
            ecb.AddComponent<Disabled>(entityInQueryIndex, mask);
        }

        // enable/disable nose covering on input change
```

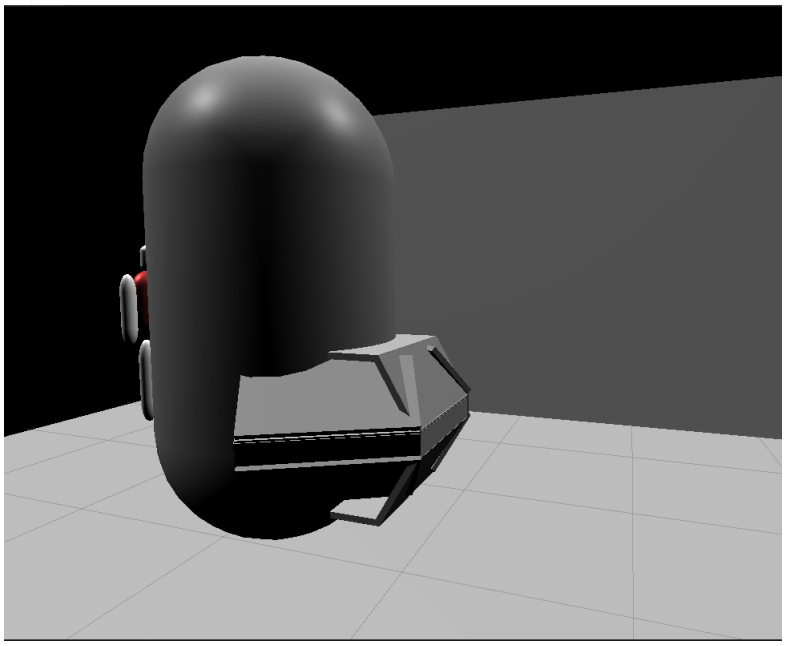


Fig. 4.12 Face mask

```
27          if (HasComponent<Disabled>(nose) && maskData.isNoseEnabled == 1) {
28              ecb.RemoveComponent<Disabled>(entityInQueryIndex, nose);
29          }
30          else if (!HasComponent<Disabled>(nose) && maskData.isNoseEnabled == 0) {
31              ecb.AddComponent<Disabled>(entityInQueryIndex, nose);
32          }
33
34      }).ScheduleParallel();
```

Listing 4.11 Face mask activation

4.4.8 VR Locomotion

For the user to have the ability to move around the scene to interact with objects, several locomotion systems are provided to the user. These systems have to interact with both versions of the VR rig. To move the rig in the scene, the motion has to be movement has to be applied to the GameObject VR rig, as the GameObject rig controls the position and rotation of the ECS rig. However, only the ECS rig can interact with entities in the scene so it also controls some movement mechanics.

Teleportation

As only entities can interact with other entities, the locomotion teleportation is handled by the ECS rig. This functions by having a ray cast out in front of the rigs hands, where the

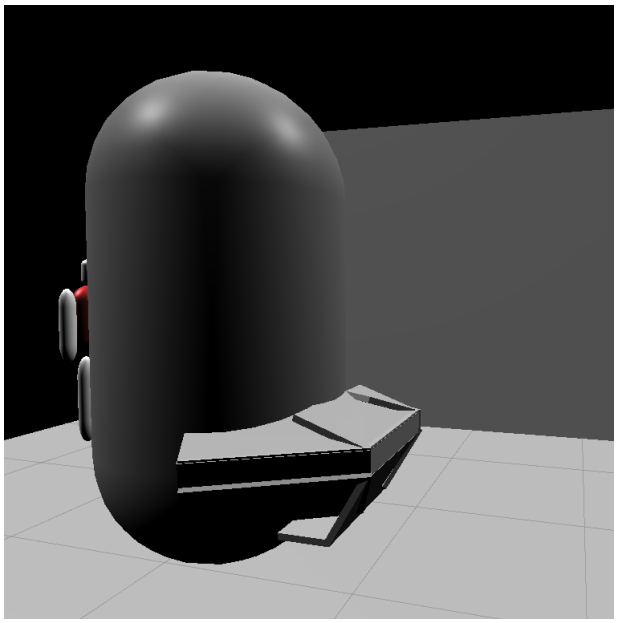


Fig. 4.13 Face mask without nose

ray intersects with the floor of the scene, the user can teleport to. To make the rigs teleport to that location the system holds a reference to the GameObject rig. From this reference, the teleportation system can set the transform of the GameObject Rig. The ECS version of the rig will move to the position of the GameObject rig in the next frame, due to the EntityCommandBuffer.

An issue with the DOTS physics engine ray casting system arose, where any attempt to cast the ray would fail. As successful ray casting is a necessity for the operation of this feature, the teleportation locomotion system cannot be completed. The code provided in Listing 4.12 show the extent to which this system was implemented.

```
Entities
    .WithName("LocomotionTeleportation")
    .WithoutBurst()
    .ForEach((Entity entity, int entityInQueryIndex, int nativeThreadIndex, in
    Translation translation, in Rotation rotation, in
    LocomotionTeleportationInputData input, in LocomotionTeleportationData
    teleportationData) => {
        if (input.enableTeleport == 1) {

            var startPos = translation.Value;
            var endPos = translation.Value + math.forward(rotation.Value) *
    teleportationData.distance;

            // construct raycast input
            var rayInput = new RaycastInput() {
                Start = startPos,
                End = endPos,
```

```
                Filter = new CollisionFilter() {
                    BelongsTo = ~0u,
                    CollidesWith = ~0u,
                    GroupIndex = 0
                }
            };

            // enable teleportation
            if (input.engageTeleport == 1) {
                bool didCast = collisionWorld.CastRay(rayInput, out var hit);

                // test raycast hit
                if (!didCast) {
                    Debug.Log("Failed to cast ray");
                }

                Debug.Log(hit.Entity);
            }
        }
    }).Run();
```

Listing 4.12 Teleportation via ray casting

Movement

The alternative movement method to teleportation is the continuous movement of the VR rigs. This movement method is controlled by the GameObject rig and the GameObject input handler. This system uses a Character Controller component to translate the rigs in the direction the GameObject rigs head node is facing. This direction is calculated by adding the Euler Y angle of the head node to the rotation of the root of the GameObject rig, then translating forward based on the input of the user.

In addition to the continuous movement locomotion system, a snap turning system is also provided to the user allowing them to rotate the rigs by a certain amount angle upon user input. To rotate the rigs, the root of the GameObject rig is rotated around the worlds up direction by a set amount, in the direction the user inputs.

```
private void ContinuousMovement(Vector2 input) {

    if (input == Vector2.zero) {
        return;
    }

    var orientation = CalculateMovementOrientation(input);

    var movement = Vector3.zero;
    movement += orientation * ( movementSpeed * Vector3.forward );

    _characterController.Move(movement * Time.deltaTime);

```

```
}

private Quaternion CalculateMovementOrientation(Vector2 input) {
    float rotation = math.atan2(input.x, input.y);
    rotation *= Mathf.Rad2Deg;

    var orientationEuler = new Vector3(0, referencePoint.eulerAngles.y + rotation, 0);
    return Quaternion.Euler(orientationEuler);
}
```

Listing 4.13 Locomotion continuous movement

```
private void SnapRotation(Vector2 input) {
    if (input == Vector2.zero || Time.timeSinceLevelLoad < _nextTurn) {
        return;
    }

    int direction = ( input.x > 0 ) ? 1 : -1;
    transform.RotateAround(hmd.position, Vector3.up, turnAngle * direction);

    _nextTurn = Time.timeSinceLevelLoad + cooldown;
}
```

Listing 4.14 Locomotion snap turning

Object Pickup

To implement the ability for the user to interact and pick up objects in the scene, the player must be able to detect what object is in front of their hands. To perform this detection, The DOTS physics engine provides an OverlapSphere() method on the PhysicsWorld to collect a list of all colliders, belonging to a collision filter group, inside a spherical area. Once an object has been detected, that object becomes a child of the user's hand, following its rotation and position. Each object's held position and rotation can be customised to sit in a more realistic orientation in the hand. To stop the objects colliding with the hand, causing physics issues, the objects collision filter is altered to not intersect with the VR rigs. When the user releases the pickup object button, the object becomes free of the hand, reverting any changes made on pickup back to default.

This is one of the most complex systems in this project due to ECS and the DOTS physics engine not having a fully implemented system to change the properties PhysicsCollider components at runtime. Unlike the translation/rotation following issue that the sticking via parenting system and the face mask system have, held object do successfully follow the movements of the user's hands.

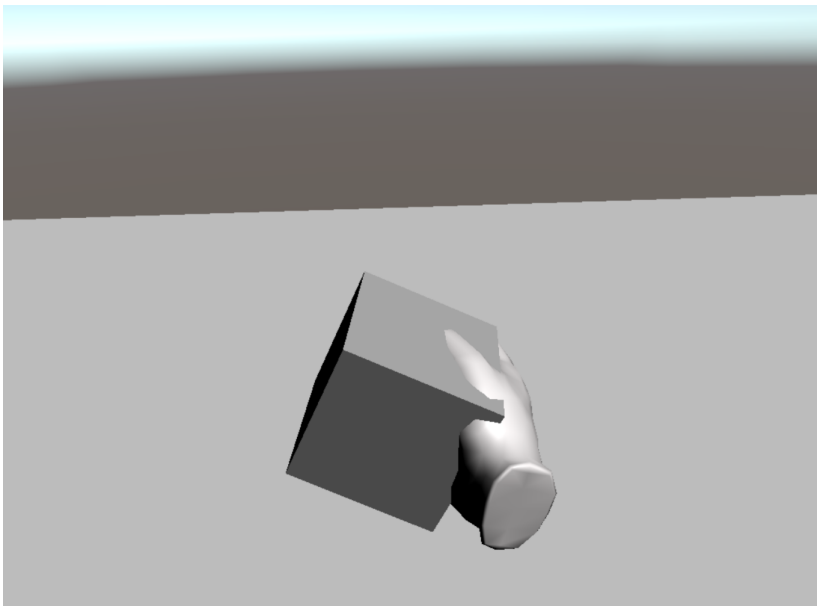


Fig. 4.14 Object pickup

4.4.9 Non-Player Character

For the user to get another perspective on the respiratory mechanics while active, two non-player characters are provided for the user to interact with. These NPCs were originally designed to provide contact with the user under handshaking and high-fiving interactions, unfortunately, due to time constraints on the development side of this project only NPC respiratory mechanics and some basic movements via steering behaviours could be implemented.

respiratory Mechanics

Similar to the user's respiratory mechanics, the NPCs can spawn multiple different types of particles during a breathing cycle. The difference between these systems is that the NPC respiratory mechanics do not require the input of the user to activate, instead of activating at the start of play and remaining active throughout the simulation.

Movement

To allow the user to observe, from a third-person perspective, how particles can spread while a person is moving, an NPC with a simple steering behaviour was implemented. This steering behaviour system calculates the forces required to move the NPC to a specified position. This system manually calculates the physics due to the DOTS physics inability to lock axes,

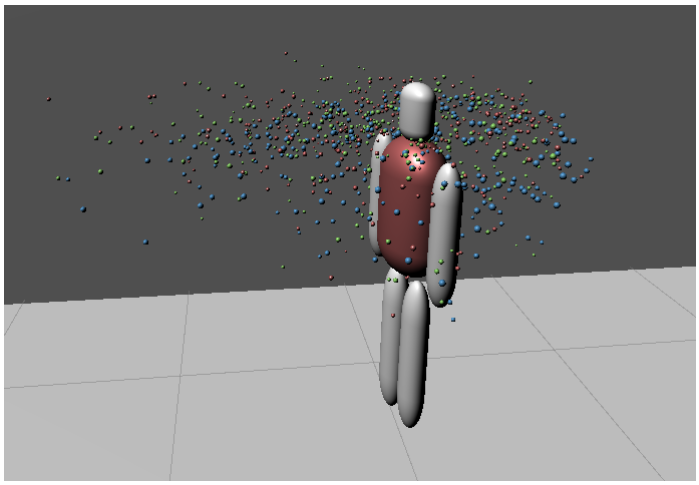


Fig. 4.15 NPC breathing in motion

ensuring the NPC would not fall over. This system also rotates the NPC to face the waypoint it is travelling to.

```
protected override void OnUpdate() {
    float dt = Time.DeltaTime;
    Entities
        .WithName("NPCLocomotion")
        .WithBurst()
        .ForEach((Entity entity, ref Translation position, ref Rotation rotation, ref
     NPCLocomotionData locomotionData, in DynamicBuffer<WaypointPositionElement>
    waypoints, in LocalToWorld ltw) => {

            if (Vector3.SqrMagnitude(waypoints[locomotionData.waypointIndex].value -
    position.Value) <= locomotionData.stopThreshold) {
                locomotionData.waypointIndex += 1;
                locomotionData.waypointIndex %= waypoints.Length;
            }

            var dirToTarget = waypoints[locomotionData.waypointIndex].value -
    position.Value;
            var desired = math.normalizesafe(dirToTarget) * locomotionData.
    movementSpeed;
            var force = desired - locomotionData.velocity;

            var acceleration = force / locomotionData.mass;
            locomotionData.velocity += acceleration * dt;
            position.Value += locomotionData.velocity * dt;

            rotation.Value = quaternion.LookRotation(position.Value + locomotionData.
    velocity * 15, ltw.Up);
        }).Schedule();
}
```

Listing 4.15 NPC movement

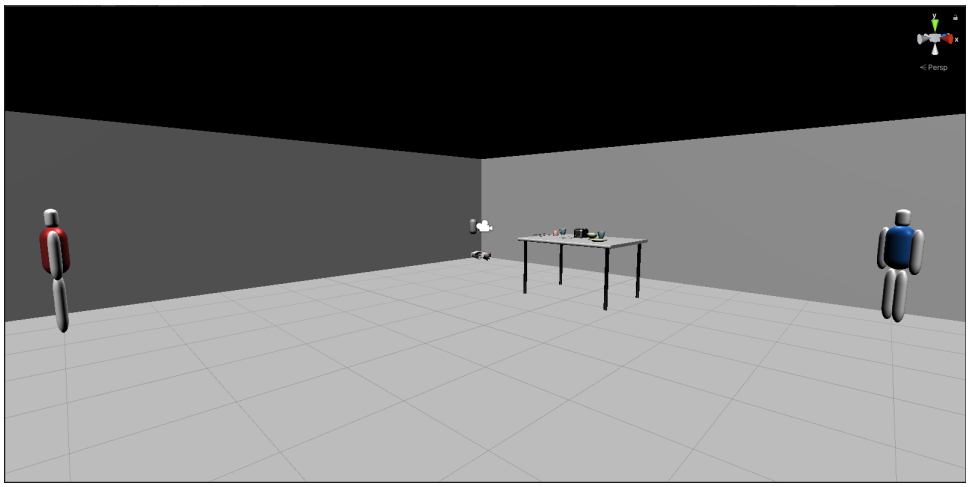


Fig. 4.16 Scene setup

4.4.10 Scene Setup

For the user to be able to experience this simulation in the intended way, an environment must be set up to experiment with viral transmission. This environment has been slightly altered from the design described in Section 3.3.2 due to not all system being implemented fully. However, the environment provided should fulfil the requirements for the simulation to be completed. As stated in Section 4.4.9, there will be two NPCs with breathing active respiratory mechanics. One NPC will is stationary in a corner for the user to move to, the other is in motion, showing how particles can spread while moving. The scene also contains a table, on which some objects are available for the user to test the pickup system (see Section 4.4.8) and to show how virus particles can become stuck to objects (if the particle sticking system was completed).

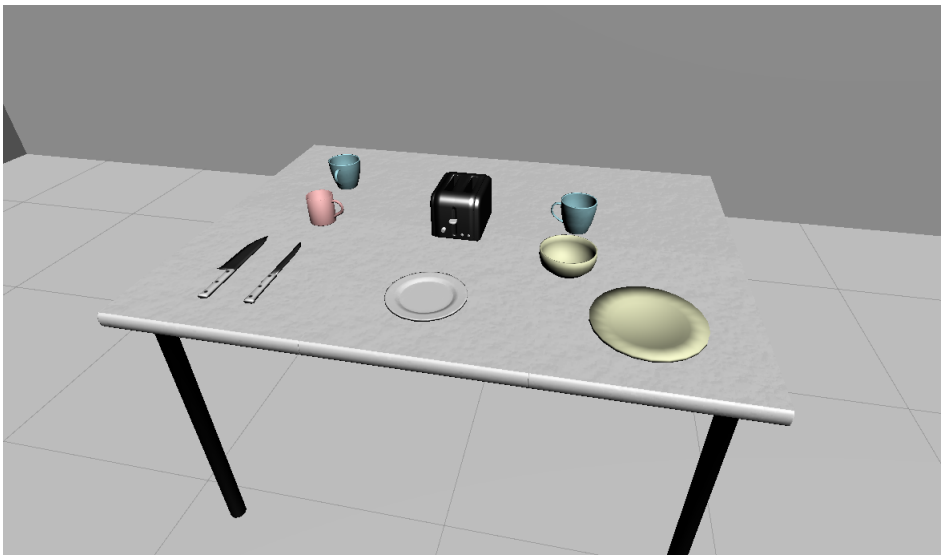


Fig. 4.17 Interactable objects

Chapter 5

System Validation

5.1 Introduction

This chapter will discuss the testing performed over the course of this projects development, as well as some of the issues encountered. First, the process of unit testing that was performed will be outlined. Following this, the outcome of playtesting the simulation during development will be discussed. Next, the results of user testing will be detailed, this user testing will be split between the simulation accuracy evaluation, as well as the VR experience evaluation. The performance of the simulation will also be examined. Finally, the conclusions drawn as a result of testing will be stated.

5.2 Unit Testing

This project had initially planned to use the white box testing method of unit testing. This unit testing would be provided by the Unity's Test Framework, an implementation of NUnit framework for C#. These unit tests would help fulfil the Agile's Feature Driven Development software development methodology that was chosen for this project. Unfortunately, due to the learning curve of the DOTS framework, the many issues plaguing the Entity Component System and DOTS physics engine, and the limited development time available, unit testing had to be sacrificed for the project to be finished on time. For each system, the amount of time required to design, and build, each unit test, assuming no DOTS framework bugs, would exceed the amount of development time allotted for a feature.

5.3 Playtesting

Over the course of development, several playtests were carried out to ensure features operated as designed. These tests were especially important to the VR DOTS link system and the locomotion systems. Many adjustments had to be made to the positions and rotations of the rigs hand models so they would match the movements of the controllers, the locations of breathing mechanic spawners also had to be moved to give more visibility to the user. The locomotion systems had to be tuned to reduce the effects of motion sickness. For the breathing mechanics on the user and NPCs, many variables can be adjusted to improve the dispersion of respiratory droplets. Due to this simulation not being strictly medically accurate, the level of adjustment as a result of playtesting was minimal, only to improve the experience for the user. During playtesting, several issues with the DOTS framework were discovered, especially features needed for the particle sticking systems. Any issue that could be fixed without a major feature redesign was, however some features, namely the locomotion teleport system, had to discarded or disabled as a result.

5.4 User Testing

To see how effective this simulation is at demonstrating viral transmission, user testing was performed. A total of 11 people tested the simulation. Each tester was shown how to use the simulation, then allowed to explore on their own. After testing, testers were asked to fill out a survey to give feedback on what they thought of the project and its purposes. This survey was divided into two parts, one for evaluating the medical accuracy of the simulation, and another to judge the performance, experience, and comfort of virtual reality.

5.4.1 Simulation Accuracy

From the results of the medical accuracy section of the survey, 81.2% of testers stated they were able to see the effects of transmission, and 88.9% said the virus particles acted as they would foresee. With regards to the simulated respiratory mechanisms, 63.3% said the breathing, sneezing, and coughing mechanics acted as they would expect if they could see them. Additionally, as a result of experiencing the simulation, 27.3% choose maybe and 36.4% said yes to making some type of changes to their lifestyle in regards to potential transmission. The main comment made about the simulation was the lack of particle sticking. The testers stated the sticking would help further show the effects of transmission, but one tester said the particles that lay on objects helped her see how they could potentially be transmitted. Some suggestions were also given for what extra features the testers would like

to see. One suggestion was to add an algorithm to show the chance that a particle may spread and to show the path of that spread. Another suggestion was to add an extra dimension to show transmission, from hand to hand or object to hand transmission. Overall, the accuracy feedback provided showed that even with this simulations low level of accuracy, the effects of transmission could be seen.

5.4.2 Virtual Reality Experience

From the results of the virtual reality experience section of the survey, 63.6% of testers stated they had little to no knowledge of extended reality technologies no tester owning any type of XR device. For the performance of the simulation an average score of 4.8 out of 5 given. This score indicates little to no drops in frame rate. Testers also gave an average score of 3.7 out of 5 was given for the simulations comfort, citing little to no motion sickness experienced during use. In regards to locomotion, only 1 participant had any issues with the locomotion systems, stating they would have liked to have the ability to teleport around the scene. 40% of testers had issues with the locomotion pickup system, asserting sometimes the wrong object was found, or the clipping of models. In general, testing participants were approved of the experience they had while using the simulation, given its simplicity, but stated some adjustments need to be made.

5.5 Performance Testing

To evaluate the performance of this simulation, several measurements were taken using Unity's statistics and profiling tools. During play, the simulation was able to perform at a stable 72 FPS, of which the time is taken to calculate each frame averaged 14 milliseconds. The simulation was able to keep this frame rate during the spawning cycle of two NPCs and the user. Using Unity's Profiler's Deep Profile setting, some measurements of execution time for the major systems were taken. These measurements were then averaged to produce the values seen in Table 5.1.

Table 5.1 Simulation performance

Feature	System Name	Execution Time (ms)
XR DOTS Link	RigConversionSystem	0.12
Virus particle spawning	VirusParticleSpawnerSystem	0.18
Virus particle sticking	ParticleCollisionParentingSystem	0.20
Particle motion	BrownianMotionRandomWalkSystem	0.11
Particle clean-up	SceneCleanupSystem	0.23
Face mask	FaceMaskSystem	0.13
Locomotion Teleportation	LocomotionTeleportationSystem	0.58
Object Pickup	LocomotionPickupSystem	0.60
NPC Movement	NPCLocomotionSystem	0.13

5.6 Validation Conclusions

With the testing performed over the course of this project, several conclusions have been drawn. These conclusions encompass the usability and performance of the project, as well as the success of the project overall.

As a result of having no unit testing in this project, the functionality of the simulation could be implemented in a reasonable amount of time. However, the lacking of means that any change that is made to the project has the potential to break some functionality. As a result, any change to the code had to be made very carefully as to not break any feature.

Any minor change to the settings of features was made as of playtesting. By having this ability to test the effectiveness, and functionality, behind each feature the correct operation could be performed as designed. Some of these changed have proved tedious to make as the DOTS framework does not allow direct adjustments to data at runtime. Ultimately, this repetitive process did allow the simulation to perform as intended.

With the results of the user testing, this project can be seen as somewhat successful in demonstrating the effects of transmission. Additionally, the resulting XR experience was satisfactory enough to cause minimal motion sickness in users. Any complaint about the project was associated with the systems that had issues being implemented and was therefore expected.

The performance provided by the DOTS framework has tremendously helped this project. Without DOTS a simulation that could perform any type of real-time simulation of thousands of objects would not be possible on an extended reality device. Additionally, the performance experienced as a result of DOTS has allowed the simulation to operate with little stuttering and latency. With DOTS still being in development, and as this project can still be further developed, performance improvements can still be made.

Chapter 6

Project Evaluation

6.1 Introduction

In this final chapter, the evaluation of the project will be given in summary. This chapter discusses the results of the goals originally set out in Section 1.2, the planning as set out in Section 3.5, as well as the issues that were encountered and persisted over the course of this project. The plans for future design and development will also be clarified. Finally, a definitive summary of what this project has achieved.

6.2 Project Results

6.2.1 Goals

Over the course of the development of this project, several objectives have tried to be met. These objectives, outlined in Section 1.2, have supplied a guide to what aspects of the system need to be researched, implemented, tested and evaluated. The following are the conclusions reached in regards to these goals:

- With the results of the user testing, this project has shown that it is possible to create an extended reality simulation to show the effects of transmission of a virus. However, due to the limitations of the technology used to implement the simulation, some aspects of transmission could not be successfully demonstrated.
- Due to issues with the simulated virus particles not successfully sticking to the user, or objects in the environment, the effects of indirect transmission could not be easily seen. Nevertheless, The particles that remained sitting on objects lessened the desire to interact with objects, according to one user tester.

- This project was designed to include two preventative measures against transmission, being a face mask and social distancing. The effects of social distancing can be seen through the interaction with NPCs, as to not have virus particles reach the user a certain distance must be observed. Unfortunately, due to issues with the particle sticking system, the face mask implementation remains unfinished.
- Over the course of development, the user comfortably was centrepiece. A benefit to the addition of the DOTS framework was the added performance the user would experience making the simulation easier to use. the locomotion system options were also designed to give the user options to help reduce potential motion sickness effects, regrettably, The teleportation system retains issues as a result of the DOTS physics system.
- During the research phase of this project. The extent to which extended reality technology has been considered for use in industry, education, entertainment, and research became clear. This project showed there are still many unexplored opportunities that extended reality has in regards to making simulations and visualisation tools.
- The limitations of extended reality technologies became very noticeable during the development phase of this project. The restrictions imposed by the hardware showed that a large amount of effort is required to produce a performant application, as there is a limit on the processing power available to the developer.

With the use of Unity's Data Oriented Technology Stack as an element of this project, some other observations were made:

- The DOTS framework has the potential to be a significant tool in the development of many applications as it provides an extreme boost performance with only a change in programming paradigm as a consequence. This type of boost will enable many applications, that otherwise would have difficulty being implemented, to be created.
- As the DOTS framework is still in development, many improvements and bugs need to be fixed before wide use is possible. Additionally, a lack of complete documentation on certain aspects means the learning curve is much higher. For more complex projects, a deep understanding of how to use the system is required.
- The DOTS Physics package that provides the ability to apply physics to entities requires more time to become the core physics engine for DOTS. This is caused by the extensive bugs and lack of features that the other physics engine that works with DOTS, the Havok physics engine, has. One such feature, the OverlapSphere method from the

collision queries features, was only added days before the start of development on the final simulation.

- The DOTS Animation package can provide the necessary features for animation use in projects. However, there are some issues with the DOTS Physics package that mean many common applications that can be performed with standard Unity, are not currently possible. One such issue is the lack of support for physics colliders of custom meshes to follow animation clips. These issues need to be fixed before release.

6.2.2 Planning

This project had been initially planned to take 32 days to complete development (See Table 3.1 for breakdown). Each feature was given a specific length of time for development based on the complexity of the system. However, due to some issues encountered during development, and the underestimation of time required to implement some features, certain features were not implemented.

6.2.3 Issues

Version Control

One issue that remained over the course of this project was a method of version control. Unity provides a business level version control service, however, for the scale of this project, it is not suitable. Instead, version control via Git and GitHub was chosen, however, this method brought issues. As Unity projects can use brought assets, some of which might be required for operation, having these assets stored on a public repository could lead to legal

Table 6.1 Feature completion list

ID	Feature	Completion Time - day(s)
1	Integrate Unity DOTS with XR	2
2	Add controller interaction	4
3	Particle spawning	5
4	Particle sticking	12
5	Particle Motion	2
6	Locomotion	4
7	Face mask	1
8	NPC breathing mechanics	1
10	Setup Scene	3

problems. As such this project had to be stored in a private repository, where collaborators had to be added manually. Additionally, as some files contain computer-specific details, posting these online poses security issues.

Unity DOTS

Due to the DOTS framework being highly experimental, limited documentation and limited examples were available for reference. As there were limited learning material, the learning curve of the DOTS framework was extremely harsh. Furthermore, the underlying complexity of certain aspects of the system, learning how to implement certain functionality consumed more time than expected. The experimental nature of this technology means bugs were bound to be found. Several issues with the Entity Component System, as well as the DOTS physics engine, meant some features could not be implemented. However, as the Data-Oriented Technology Stack continues to be developed, this technology will become more stable leading to fewer issues and increased performance. Furthermore, in-house DOTS versions of the XR and input system may be developed, reducing the need for the systems created in this project.

Testing

Unfortunately, as a result of the COVID-19 global pandemic, the number of testers had to be decreased due to government restrictions. As a result, the survey results received have the potential to be slightly biased due to the small sample size. Therefore, if more user testing were to be performed, conclusions about this project may be different.

6.3 Future Plans

With more time to develop this project, the features missing in the initial development can be implemented. These additional features would make the simulation easier for a user to operate and understand. Moreover, a fully featured project would enable demonstrations of the potential other uses this simulation has.

As part of completing the feature list, the custom models that were initially planned can be created. By having custom models users will be able to more clearly see the effects of viral transmission. These models will also provide the base for better NPC-user interaction.

For additional features, past those that are planned, a better implementation of respiratory droplet dispersion and viral transmission would help make this project more accurate. Additionally, instead of showing viral transmission of a generic virus, having accurate representations of different viral pathogens would make this project more educational and research

relevant. To help portray how these particles are invisible to the naked eye, a lighting system could be developed to only show the particles, and their contamination of the scene, when active.

As this project currently only supports the Oculus Quest 2, extending support for other XR devices would make this simulation much more accessible to users. By extending support for other HMDs, the quality of the DOTS VR link system and locomotion systems will also need to be improved. Having more advanced VR systems would help further reduce the negative side effects a user could potentially experience.

As the DOTS framework becomes more developed, new features and systems will likely be added. These new additions could potentially implement ECS systems that otherwise would have to be designed and implemented. Changes to the DOTS framework will also bring optimisations. These optimisations could likely change how certain systems are implemented, meaning maintenance on this project could be extensive.

6.4 Final Conclusions

With the results of the performed testing, the VR Transmission Simulation has shown itself to demonstrate the effects of viral transmission with some semblance of accuracy. Additionally, the VR Transmission Simulation has shown the potential of Unity's Data Oriented Technology Stack as a method to increase performance in XR simulations. Therefore, this project has shown that XR technologies do have the potential to be used for both the education and research of transmission. However, to create certain simulation, some technical changes may have to be made to have an application that can be used without any resulting side effects.

References

Andrews, C., Southworth, M.K., Silva, J.N. and Silva, J.R. (2019) Extended reality in medical practice *Current treatment options in cardiovascular medicine* **21**(4), pp. 1–12

Autodesk Inc. (1988) 3ds Max | 3D Modeling, Animation and Rendering Software | Autodesk https://www.autodesk.eu/products/3ds-max/overview

Blender Foundation (1998) blender.org - Free and Open 3D Creation Software https://www.blender.org/

Bourouiba, L., Dehandschoewercker, E. and Bush, J.W.M. (2014) Violent expiratory events: on coughing and sneezing *Journal of Fluid Mechanics* **745**, pp. 537–563

Bowman, D.A. and McMahan, R.P. (2007) Virtual reality: How much immersion is enough? *Computer* **40**(7), pp. 36–43

Bozgeyikli, E., Raij, A., Katkoori, S. and Dubey, R. (2016) Point & teleport locomotion technique for virtual reality in: *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play* pp. 205–216

Burdea, G.C. and Coiffet, P. (2003) *Virtual reality technology* John Wiley & Sons

Chen, T.M., Rui, J., Wang, Q.P., Zhao, Z.Y., Cui, J.A. and Yin, L. (2020) A mathematical model for simulating the phase-based transmissibility of a novel coronavirus *Infectious diseases of poverty* **9**(1), pp. 1–8

Choi, J.O., Forlizzi, J., Christel, M., Moeller, R., Bates, M. and Hammer, J. (2016) Playtesting with a purpose in: *Proceedings of the 2016 annual symposium on computer-human interaction in play* pp. 254–265

Christou, C. (1970) Virtual Reality in Education https://www.igi-global.com/chapter/virtual-reality-education/40560

Coad, P., Luca, J.d. and Lefebvre, E. (1999) *Java modeling color with UML: Enterprise components and process with Cdrom* Prentice Hall PTR

Da Zhou, C., Sivathondan, P. and Handa, A. (2015) Unmasking the surgeons: the evidence base behind the use of facemasks in surgery *Journal of the Royal Society of Medicine* **108**(6), pp. 223–228

Daka, E. and Fraser, G. (2014) A Survey on Unit Testing Practices and Problems in: *2014 IEEE 25th International Symposium on Software Reliability Engineering* pp. 201–211

Desurvire, H. and Kreminski, M. (2018) Are game design and user research guidelines specific to virtual reality effective in creating a more optimal player experience? yes, vr play in: *International Conference of Design, User Experience, and Usability* pp. 40–59 Springer

Einstein, A. and Furth, R. (1956) *Investigations on the theory of the Brownian movement* Dover publications

Ellis, M.A. and Stroustrup, B. (1994) *The annotated C++ reference manual* Addison-Wesley

Epic Games (1998) Unreal Engine https://www.unrealengine.com/en-US/

Facebook Technologies, LLC (2020) Oculus Quest 2 https://www.oculus.com/quest-2/

Fletcher, F.E., Allen, S., Vickers, S.M., Beavers, T., Hamlin, C.M., Young-Foster, D., Harris-Turner, S. and Erwin, P.C. (2020) Covid-19's impact on the african american community: a stakeholder engagement approach to increase public awareness through virtual town halls *Health Equity* **4**(1), pp. 320–325

Furht, B. (2011) *Handbook of augmented reality* Springer Science & Business Media

Gandhi, R.D. and Patel, D.S. (2018) Virtual reality–opportunities and challenges *Virtual Reality* **5**(01)

Grandviewresearch.com (2020) virtual reality in gaming market size report https://www.grandviewresearch.com/industry-analysis/virtual-reality-in-gaming-market

Guo, J., Weng, D., Been-Lirn Duh, H., Liu, Y. and Wang, Y. (2017) Effects of using hmds on visual fatigue in virtual environments in: *2017 IEEE Virtual Reality (VR)* pp. 249–250

Harrison, J.M. and Shepp, L.A. (1981) On skew Brownian motion *The Annals of probability* pp. 309–313

Helsel, S. (1992) Virtual Reality and Education *Educational Technology* **32**(5), pp. 38–42

Hughes, C., Stapleton, C., Hughes, D. and Smith, E. (2005) Mixed reality in education, entertainment, and training *IEEE Computer Graphics and Applications* **25**(6), pp. 24–30

Hui-Wen Chuah, S. (2018) Why and who will adopt extended reality technology? literature review, synthesis, and future research agenda *SSRN Electronic Journal*

Ito, K., Tada, M., Ujike, H. and Hyodo, K. (2019) Effects of weight and balance of head mounted display on physical load in: *International Conference on Human-Computer Interaction* pp. 450–460 Springer

Kenwright, B. (2018) Virtual reality: Ethical challenges and dangers [opinion] *IEEE Technology and Society Magazine* **37**(4), pp. 20–25

Kwon, S.B., Park, J., Jang, J., Cho, Y., Park, D.S., Kim, C., Bae, G.N. and Jang, A. (2012) Study on the initial velocity distribution of exhaled air from coughing and speaking *Chemosphere* **87**(11), pp. 1260–1264

Ladhani, L., Pardon, G., Moons, P., Goossens, H. and van der Wijngaart, W. (2020) Electrostatic sampling of patient breath for pathogen detection: A pilot study *Frontiers in Mechanical Engineering* **6**

Leung, H. and White, L. (1990) A study of integration testing and software regression at the integration level *Proceedings. Conference on Software Maintenance*

Lindsley, W.G., Noti, J.D., Blachere, F.M., Szalajda, J.V. and Beezhold, D.H. (2014) Efficacy of face shields against cough aerosol droplets from a cough simulator *Journal of Occupational and Environmental Hygiene* **11**(8), pp. 509–518

Liu, L., Wei, J., Li, Y. and Ooi, A. (2016) Evaporation and dispersion of respiratory droplets from coughing *Indoor Air* **27**(1), pp. 179–190

Liu, L., Wei, J., Li, Y. and Ooi, A. (2017) Evaporation and dispersion of respiratory droplets from coughing *Indoor air* **27**(1), pp. 179–190

Loebenstein, G. and Carr, J.P. (2006) *Natural resistance mechanisms of plants to viruses* Springer

Mantovani, F., Castelnuovo, G., Gaggioli, A. and Riva, G. (2003) Virtual reality training for health-care professionals *CyberPsychology & Behavior* **6**(4), pp. 389–395

Matrajt, L. and Leung, T. (2020) Evaluating the effectiveness of social distancing interventions to delay or flatten the epidemic curve of coronavirus disease *Emerging Infectious Diseases* **26**(8), pp. 1740–1748

McGrail, D.J., Dai, J., McAndrews, K.M. and Kalluri, R. (2020) Enacting national social distancing policies corresponds with dramatic reduction in covid19 infection rates *PLOS ONE* **15**(7), p. e0236619

Microsoft Corporation (2000) c# docs - get started, tutorials https://docs.microsoft.com/en-us/dotnet/csharp/

Microsoft Corporation (2019) HoloLens 2 https://www.microsoft.com/en-us/hololens/hardware

Mujber, T.S., Szecsi, T. and Hashmi, M.S. (2004) Virtual reality applications in manufacturing process simulation *Journal of materials processing technology* **155**, pp. 1834–1838

Nichols, S. and Patel, H. (2002) Health and safety implications of virtual reality: a review of empirical evidence *Applied ergonomics* **33**(3), pp. 251–271

Noël, P.A., Davoudi, B., Brunham, R.C., Dubé, L.J. and Pourbohloul, B. (2009) Time evolution of epidemic disease on finite and infinite networks *Physical Review E* **79**(2), p. 026101

Oldstone, M.B. (2020) *Viruses, plagues, and history: past, present, and future* Oxford University Press

Palmer, S.R. and Felsing, M. (2001) *A Practical Guide to Feature-Driven Development* Pearson Education 1st ed.

Patrão, B., Pedro, S. and Menezes, P. (2020) How to Deal with Motion Sickness in Virtual Reality in: *22o Encontro Português de Computação Gráfica e Interação 2015*, (Eds.) P. Dias and P. Menezes The Eurographics Association

Poole, C. and Prouse, R. (2019) NUnit https://nunit.org/

Ratcliffe, J., Soave, F., Bryan-Kinns, N., Tokarchuk, L. and Farkhatdinov, I. (2021) Extended reality (xr) remote research: a survey of drawbacks and opportunities *arXiv preprint arXiv:2101.08046*

Rodríguez-Hidalgo, A.J., Pantaleón, Y., Dios, I. and Falla, D. (2020) Fear of covid-19, stress, and anxiety in university undergraduate students: A predictive model for depression *Frontiers in Psychology* **11**, p. 3041

Roy, D., Tripathy, S., Kar, S.K., Sharma, N., Verma, S.K. and Kaushal, V. (2020) Study of knowledge, attitude, anxiety & perceived mental healthcare need in indian population during covid-19 pandemic *Asian journal of psychiatry* **51**, p. 102083

Rutala, W. and Weber, D. (2001) Surface disinfection: should we do it? *Journal of Hospital Infection* **48**, pp. S64–S68

Rutala, W.A. and Weber, D.J. (2008) Guideline for disinfection and sterilization in healthcare facilities, 2008

Scheuch, G. (2020) Breathing is enough: For the spread of influenza virus and sars-cov-2 by breathing only *Journal of Aerosol Medicine and Pulmonary Drug Delivery* **33**(4), pp. 230–234

Sinai, Y.G. (1983) The limiting behavior of a one-dimensional random walk in a random medium *Theory of Probability & Its Applications* **27**(2), pp. 256–268

Straume, P.M. (2019) Investigating Data-Oriented Design Master's Thesis NTNU

Tang, J.W. and Settles, G.S. (2008) Coughing and aerosols *New England Journal of Medicine* **359**(15), p. e19

Turnbull, P.R.K. and Phillips, J.R. (2017) Ocular effects of virtual reality headset wear in young adults *Scientific Reports* **7**(1)

Unity Technologies (2005) Unity https://unity.com/

Unity Technologies (2019) Unity - DOTS https://unity.com/dots

Unity Technologies (2021) Entitycomponentsystemsamples https://github.com/Unity-Technologies/EntityComponentSystemSamples

Valve Corporation (2020) Valve Index https://www.valvesoftware.com/en/index

Virtuix Inc. (2017) Virtuix Omni https://www.virtuix.com/

Weissman, G.E., Crane-Droesch, A., Chivers, C., Luong, T., Hanish, A., Levy, M.Z., Lubken, J., Becker, M., Draugelis, M.E., Anesi, G.L. *et al.* (2020) Locally informed simulation to predict hospital capacity needs during the covid-19 pandemic *Annals of internal medicine* **173**(1), pp. 21–28

Wendling, J.M., Fabacher, T., Pebay, P.P., Cosperec, I. and Rochoy, M. (2020) Experimental efficacy of the face shield and the mask against emitted and potentially received particles

Wilder-Smith, A. and Freedman, D.O. (2020) Isolation, quarantine, social distancing and community containment: pivotal role for old-style public health measures in the novel coronavirus (2019-ncov) outbreak *Journal of Travel Medicine* **27**(2)

Xu, Z., Shen, F., Li, X., Wu, Y., Chen, Q., Jie, X. and Yao, M. (2012) Molecular and microscopic analysis of bacteria and viruses in exhaled breath collected using a simple impaction and condensing method *PLoS ONE* **7**(7), p. e41137

Yan, Y., Li, X. and Tu, J. (2019) Thermal effect of human body on cough droplets evaporation and dispersion in an enclosed space *Building and Environment* **148**, pp. 96–106