



# **CoviSim: Research into the Demonstration of Transmission of COVID-19 Final Project Report**

**DT211C  
BSc in Computer Science (Infrastructure)**

**Kyle Heffernan**

**C17444434**

**Bryan Duggan**

School of Computer Science  
Technological University, Dublin

**31/03/2021**

# Abstract

Computer simulation has always been an invaluable tool when it comes to researching infectious diseases, as real-life experiments have many potential risks. Over the course of the past year, countless scientists and doctors all over the world have been continuously researching Coronavirus in a global effort to overcome the pandemic and get back to normal everyday life. There have been numerous Coronavirus related simulations made over the past year focusing on a wide variety of aspects of the virus.

Many simulations offer a high-level overview of the pandemic on a large scale, having only a few variables affecting the results. These simulations tend to focus on the spread throughout a city, and the virus is transmitted when agents come within a certain range of an infected agent. While this serves as a good visualisation of spread throughout a population, it is a drastic oversimplification of how transmission can occur and does not show how the virus actually transmits between people.

This project is focused on transmission in a closed environment, highlighting the actual methods of transmission and allowing the user to truly understand how certain countermeasures affect the results. There is a surplus of medical papers and scientific studies from around the world which provide statistics on transmission rates and the effects of various countermeasures. Some of these statistics have been utilised in the simulation as parameters to give a more accurate result.

As Coronavirus continues to grow, so does misinformation about it on social media. While a small amount of information is given to the public about countermeasures that they can take to prevent transmission, the results of these countermeasures are not easy to identify. This simulation is a practical solution to this, using real figures to visualise transmission and the effectiveness of various countermeasures in a real-time closed environment.

# Declaration

I hereby declare that the work described in this dissertation 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.

Signed:

A handwritten signature in black ink, reading "Kyle Heffernan", is written over a solid horizontal line. The signature is cursive and fluid.

Kyle Heffernan

Date: 31/03/2021

# Acknowledgements

I would like to thank my supervisor Bryan Duggan for his continuous support, guidance, and motivating words throughout the course of this year. I would also like to thank my friends for keeping me motivated and on track this year with our countless online study sessions. I would also like to thank Ryan for all of his advice and support.

A special thanks to my family for always keeping me grounded and helping me through the more stressful times, and finally Danniella for her constant love and support which I could not have done this project without.

## Table of Contents

Table of Figures .....	7
1. Introduction .....	9
1.1. Project Background .....	9
1.2. Project Description .....	9
1.3. Project Aims and Objectives .....	10
1.4. Project Scope .....	10
1.5. Thesis Roadmap .....	11
2. Literature Review .....	12
2.1. Introduction .....	12
2.2. Alternative Existing Solutions to Your Problem .....	12
2.3. Technologies You've Researched .....	16
2.4. Other Research You've Done .....	18
2.5. Existing Final Year Projects .....	19
2.6. Conclusions .....	19
3. Experiment Design .....	20
3.1 Introduction .....	20
3.2. Software Methodology .....	20
3.3. Project Management .....	20
3.4. Overview of System .....	21
3.4.1. Technical Architecture .....	21
3.4.2. System Flowchart .....	21
3.4.3. Table of Requirements .....	23
3.5. Front-End .....	24
3.5.1. Key Screens .....	24
3.6. Back-End .....	27
3.6.1 Behaviour Tree Diagrams .....	27
3.6.2 Class Dependencies Diagram .....	28
3.7. Conclusions .....	29
4. Experiment Development .....	30
4.1. Introduction .....	30
4.2. Environment Setup .....	30
4.3. Entity Component System .....	30
4.4. SNAPS Tool and Navigation Mesh .....	31
4.5. Simple Path Following and Infection Model .....	33
4.6. UI and Timescale .....	34

4.7. Behaviour Trees .....	36
4.8. Particle Simulation .....	40
4.9. Cameras and User Movement .....	42
4.10. Animated Models.....	43
4.11. Working Hours and End Screen .....	46
4.12. Masks and Vaccines .....	49
4.13. Contaminated Surfaces and Agents.....	52
4.14. Finishing UI.....	53
4.15. Conclusions .....	56
5. Testing and Evaluation.....	57
5.1. Introduction .....	57
5.2. System Testing .....	57
5.2.1 Testing During Development .....	57
5.2.2 Testing After Development.....	57
5.3. System Evaluation.....	63
5.3.1 Evaluating Simulation Results .....	63
5.3.2 Evaluating Whole System.....	64
5.4. Conclusions .....	64
6. Conclusions and Future Work.....	65
6.1. Introduction .....	65
6.2. Future Work .....	65
6.3. Gant Chart.....	66
6.4. Project Conclusions.....	67
6.4.1 Literature review.....	67
6.4.2 Experiment Design .....	67
6.4.3 Experiment Development .....	67
6.4.4 Evaluation .....	67
6.4.4 Final Reflections .....	68
Bibliography .....	69

## Table of Figures

Figure 1 – Grocery Store simulation .....	12
Figure 2 – Spread simulation .....	14
Figure 3 – Live output .....	14
Figure 4 – Case studies.....	15
Figure 5 – Github task tracking .....	21
Figure 6 -Technical Architecture .....	21
Figure 7 – System flowchart.....	22
Figure 8 – Key screen 1 .....	24
Figure 9 – Key screen 2 .....	25
Figure 10 – Key screen 3 .....	26
Figure 11 – Key screen 4 .....	27
Figure 12 – Behaviour tree diagram index.....	27
Figure 13 – Behaviour tree diagram .....	28
Figure 14 – Dependencies diagram.....	29
Figure 15 – Sample assets .....	31
Figure 16 – Office environment .....	32
Figure 17 - NavMesh .....	32
Figure 18 – Movement Code.....	33
Figure 19 – Capsule agents .....	34
Figure 20 – Status counter displays .....	34
Figure 21 – Stats counter code .....	35
Figure 22 – Altering timescale .....	35
Figure 23 – Start button.....	35
Figure 24 – Place manager .....	36
Figure 25 – Behaviour tree.....	37
Figure 26 – Getting agents desk.....	37
Figure 27 – Check working behaviour.....	38
Figure 28 – Check and go to desk behaviour .....	38
Figure 29 – Finish working behaviour .....	39
Figure 30 – Get rec target behaviour .....	39
Figure 31 – Check and go to rec point behaviour .....	39
Figure 32 – Finish rec behaviour .....	40
Figure 33 – Start of particle collision script .....	40
Figure 34 – Setting all materials of an object .....	41
Figure 35 – Breath particle system – coming from the right .....	42
Figure 36 – Camera switching script .....	42
Figure 37 – Timescale independent movement script .....	43
Figure 38 – Character model.....	43
Figure 39 - Animator .....	44
Figure 40 – Animation state controller .....	44
Figure 41 – Make agent face desk .....	45
Figure 42 – Updated behaviour tree.....	45
Figure 43- Changing agents material .....	46
Figure 44 – Final behaviour tree .....	46
Figure 45 – Working hours behaviour .....	47

Figure 46 – Working hours slider .....	47
Figure 47 – End screen code .....	48
Figure 48 – Quit application code .....	48
Figure 49 – Restart scene code .....	49
Figure 50 – Blender mask model .....	49
Figure 51 – Face player script .....	50
Figure 52 – Vaccine indicator .....	50
Figure 53 – Infection chances .....	51
Figure 54 – Adding contamination particles .....	52
Figure 55 – Contaminated surface with particles .....	52
Figure 56 – Exposed agent with particles on their body.....	53
Figure 57- Final start screen.....	54
Figure 58 – Final UI elements.....	54
Figure 59 – Time displays.....	55
Figure 60 – FPS display.....	55
Figure 61 – End screen .....	55
Figure 62 – Form Q1.....	58
Figure 63 – Form Q2.....	58
Figure 64 – Form Q3.....	59
Figure 65 – Form Q4.....	59
Figure 66 – Form Q5.....	60
Figure 67 – Form Q6.....	60
Figure 68 – Form Q7.....	61
Figure 69 - Form Q8, Q9.....	62
Figure 70 – Form Q10 .....	63
Figure 71 – Gant chart .....	66



# 1. Introduction

## 1.1. Project Background

As the number of Coronavirus cases continues to grow worldwide, scientists and medical professionals from all over the world have been researching and studying the virus and its transmission to better understand and subsequently overcome it. Due to many real-life experiments being too risky to carry out, computer simulation has been an invaluable tool for developing further understanding of the virus and its transmission. This project involves creating a simulation of an environment in which transmission commonly occurs, an office.

There is a vast number of platforms available for development in this field, but Unity stands out with its countless invaluable features and tools that enable swift and efficient development of real-time simulations. The use of Unity also allows the use of various complex technologies, such as navigation mesh which creates a map of traversable areas in a scene and grants agents the ability to find the shortest path to their destination. A behaviour tree is another technology available in Unity which is a mathematical model of plan execution, meaning an artificially intelligent agent can switch between a set of tasks in a modular fashion. Unity also grants the ability to implement particle systems that can be used to simulate particles being expelled during breathing. Finally, Unity also has the entity component system, which is a new data-orientated design system that significantly boosts the performance of the simulation if implemented correctly.

## 1.2. Project Description

The purpose of this project is to simulate the environment of a populated office. The program starts off on a screen that allows the user to adjust certain variables which will affect the result.

Once the simulation is started, autonomous agents enter the building representing workers going about their daily work shift. The building has a navigation mesh which is utilised by the agents so they can path find through the office. At the start of the simulation, each agent goes to their respective desk, which is chosen at random at the start and their daily work. Agents will intermittently do various tasks such as retrieving a file or printing something off and then return to their desks. This is due to each of the agents having a behaviour tree, so they have a set of tasks that they switch between.

One of the agents is infected with COVID-19 and is continuously spreading it throughout the office as the day goes on via a particle system that emulates breathing. The virus is spread by the particles emitting, which can contaminate a surface or expose an agent if they collide with it. Agents also have a chance of becoming exposed if they touch another exposed agent or if they touch a contaminated surface. The chances of an agent becoming exposed when they come in contact with an infectious particle are affected by what the user selected at the start of the simulation, and the figures used in these calculations are taken from real studies which are discussed in the literature review.

As the simulation runs, the user can walk around the office to get a better view of the virus spreading, or they can look through the office security cameras. The user is also able to change the

rate at which time passes, so they could have the simulation run at times ten-speed to see the results faster.

Once the working hours set by the user have ended, the agents begin to leave the office. Once they all leave, a screen is displayed to the user with statistics from the simulation that was just run, and the user has the option to run the simulation again with different options.

*Factors the user can change:*

<b>Working Hours</b>	The amount of time the users stay in the office.
<b>If healthy agents are vaccinated</b>	This alters the chances of a healthy agent becoming exposed.
<b>If healthy agents wear masks</b>	This alters the chances of a healthy agent becoming exposed.
<b>If infected agents wear masks</b>	This alters the number of particles emit by the infectious agent.
<b>Time scale</b>	The rate at which time passes in the simulation.

### 1.3. Project Aims and Objectives

1. Identify and review suitable literature and other references relevant to this project
2. Describe some other software systems that are like this project
3. Undertake a thorough design process, including a methodology and detailed design
4. Develop a working software system using suitable technologies
5. Test and Evaluate the developed system
6. Critically reflect on the outcomes of this entire process

### 1.4. Project Scope

This project allows users to view a COVID-19 simulation in real-time and alter certain variables to see how they affect the transmission results. The simulation is made using Unity, and the environment in which the simulation takes place is a populated building with autonomous agents using behaviour trees. Navigation Mesh is used to map out the walkable paths for the agents throughout this environment. The agents have human models and custom AI allowing them to go to their assigned desk and work, intermittently going to do various tasks around the office. Then once their working hours end, they leave the office.

Infected agents emit particles using Unity's Particle System that leave surfaces contaminated and they can expose other agents to the virus based on their susceptibility. Agents can also become exposed from a contaminated surface or from getting too close to a different exposed agent. As the simulation is running, the user can walk around or look through the office security cameras. The user can alter the time scale to speed up the simulation, and they can also adjust variables that affect the result of the simulation. When the agents all leave the office, a screen is displayed with some statistics from the simulation that was just run, and the user can then restart the simulation with different variables.

## 1.5. Thesis Roadmap

### *Literature Review*

In this chapter, a description of the main technologies and resources researched is presented, including academic papers, tutorials, books, and websites. The main technologies involved with the system are discussed, along with some other related research. It also looks at existing virus simulations made in Unity and previous final year projects with similarities to this project.

### *Experiment Design*

In this chapter, the software methodology used during the development process is discussed with the reasoning behind its choice, and an overview of the system is presented. The technical architecture of the system is presented, and the design of the front-end and back-end of the system is described with the aid of numerous diagrams.

### *Experiment Development*

In this chapter, the entire software development process is discussed in detail. Beginning with setting up the environment and moving through the process describing the development of each feature in detail, along with the issues faced and how they were overcome.

### *Evaluation*

This chapter discusses how the system was tested and how feedback was considered during and after the development process. The chapter also contains an in-depth evaluation of the system, discussing its performance, accuracy, and comparing its results to alternative similar systems.

### *Conclusions and Future Work*

This chapter summarises the complete project, discussing the major concepts learned during the development process as well as the issues that arose and how they were approached. This chapter also includes a discussion of the future of this project and how it can be expanded upon with certain features and generally improved.

## 2. Literature Review

### 2.1. Introduction

In this chapter, a review of relevant research and other software is presented as it relates to the simulation system. First existing software that performs similar functions to this project are presented, and following that, the technologies used in this system are discussed. Other research including academic papers and web information is then presented. Finally, two existing final year projects are discussed.

### 2.2. Alternative Existing Solutions to Your Problem

#### *Exploring new ways to simulate the coronavirus spread (1)*

Released in May 2020, this Unity Blog is about a Coronavirus spread simulation that is developed in Unity and C#. The project contains a simulation of a grocery store, with customers coming and going to and from the store. Some customers are infected and can expose other customers to the virus if they are within a certain range for long enough. The project has a graphical user interface (GUI) at the side of the screen which allows the user to alter various parameters, apply the changes, and see how they affect the results which are also displayed on the GUI.

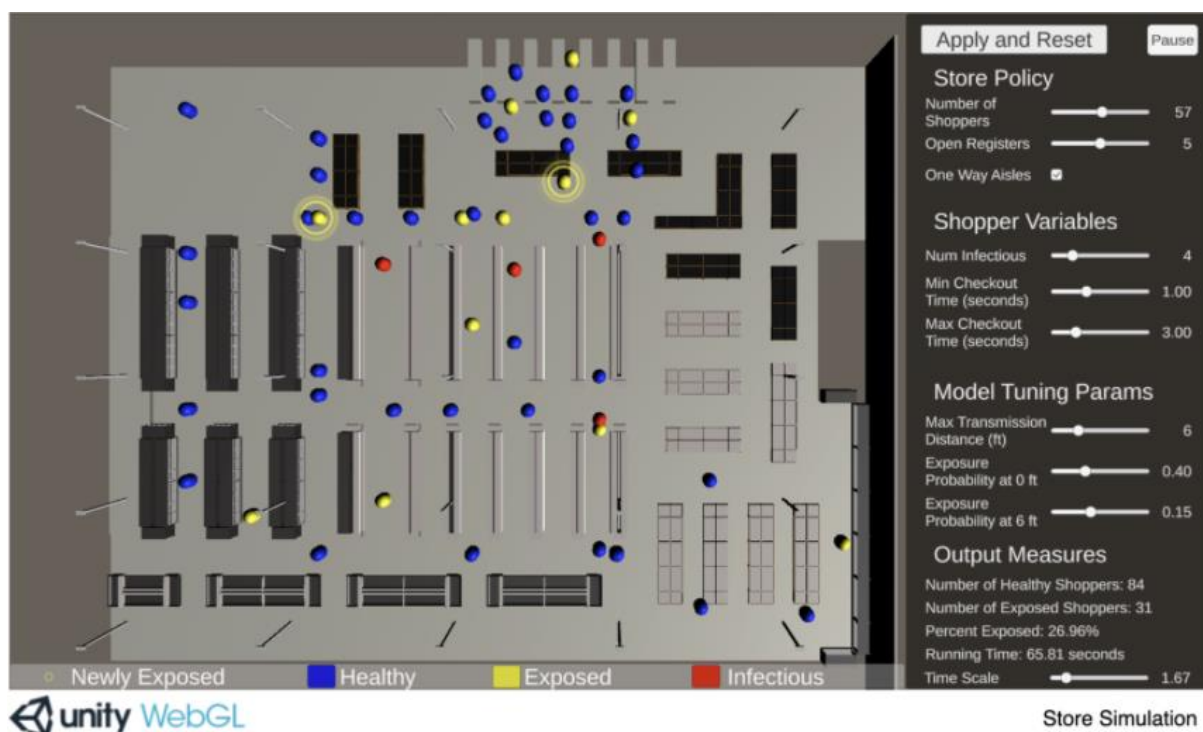


Figure 1 – Grocery Store simulation

### *Software Features:*

**Grocery Store Environment:** The project contains a simulated grocery store, with aisles, registers, entrances etc. The shoppers travel around this simulated store.

**Shoppers:** There are agents in the shape of capsules which represent shoppers. They follow certain routes throughout the store.

**Configurable parameters:** Parameters like exposure distance and transmission probability are adjustable using the sliders in the GUI on the right of the screen. Once the “Apply and Reset” button is pressed, the actual variables which are used in the simulation are updated accordingly, and the effects will be visible.

**Time scale:** The scale of the simulation can be adjusted using the GUI, allowing the user to choose how fast they would like time to go by in the simulation.

**Mapping:** The traversable routes are determined procedurally based on criteria including entrances and exits, whether certain sections are one way only, and making sure there are no collisions.

**Movement:** When shoppers spawn, they pick random traversable paths throughout the store. These paths start at the entrance, have random amounts of intermediate goals, and end at the exit.

**Exposure:** Shoppers spawn as either healthy or infectious. When infectious shoppers come close with other shoppers, they can expose them to the virus based on some set parameters. These shoppers are then set to exposed.

**Queuing:** Before each shopper approaches the registers, they check if there are any open registers, and then get queued accordingly based on the store policy parameters.

This grocery store simulation has many similar features to this project. The concept of having a GUI screen with configurable parameters is close to the GUI that this project has, although this project has the GUI screen only at the start. A lot of the other features are rather similar too, such as having agents walk throughout the simulated environment with a chance of becoming exposed. The logic of having infectious agents exposing healthy agents to the virus is the same, yet this project is much more in-depth, having the actual particles emit from breathing being the carrier of the virus rather than just a simple collision behaviour. The grocery store itself is also similar to the simulated office in which this project takes place, although this project offers a 3d space in which the player can move around in rather than just a top-down view.

Both the grocery store project and this project are made completely in Unity and C#, so the technologies used are closely related, although this project makes use of some more complicated technologies such as behaviour trees for AI and particle systems to emulate breathing.

### *How coronavirus spreads through a population and how we can beat it (2)*

Published in early 2020, this article presents a simulation of the spread of certain viruses throughout a population of people. It allows the user to adjust some parameters using the sliders at the top, and then shows how the virus would spread over a period of time. As well as allowing the user to adjust these parameters, they can also select one of the case studies and see a visualisation of the spread using statistics from the actual case study.



In this model every susceptible member of the group (80% of the population) was infected after 8.6 phases. **18 people died.**

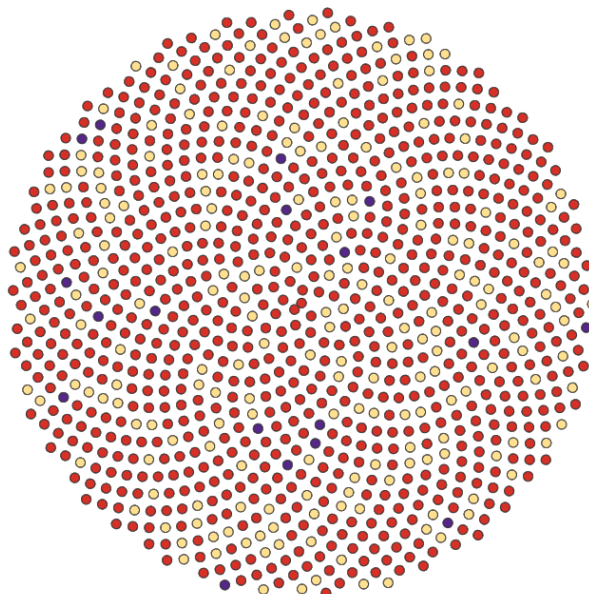
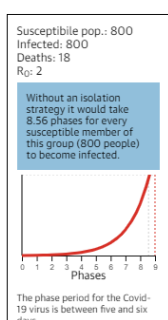


Figure 2 – Spread simulation

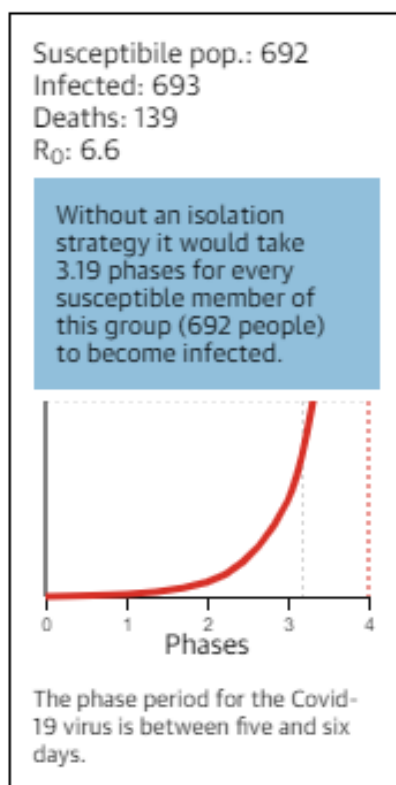


Figure 3 – Live output

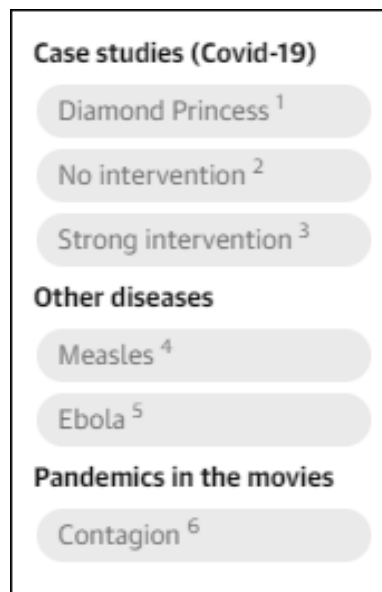


Figure 4 – Case studies

#### Software Features:

**Infectious indicators:** Members of the population start off as yellow which indicates a healthy person. Red indicates they are infected with the virus, and purple represents people who have died from the virus.

**Adjustable parameters:** As seen at the top of the screenshot, the user can move the sliders to change the parameters of the simulation. They can then see of visualisation of how the chosen values would affect the results.

**Case studies:** The user can select from a short list of case studies to see a visualisation of the spread that took place during these case studies.

**Utilising real statistics:** If a case study is chosen by the user, the simulation will run using parameters taken from real-life statistics.

**Displaying results:** As the simulation runs through the phases, it updates the visualisation of the population with the corresponding colours. It also displays the numbers after each phase and displays the stage on a chart as it updates.

This population spread simulation also has numerous similarities to this project. Both projects take some statistics from real life and use them as parameters for the simulation, and also allow the user to adjust variables and see the results. They also both focus on visualising the spread of the virus, although the population spread simulation went in a completely different direction, focusing on spread over a long amount of time, and as a result, it is much less detailed than this project is and does not touch on the transmission methods of the virus, in turn making it a somewhat simple system.

## 2.3. Technologies You've Researched

### *Godot (3)*

Godot is an open-source game engine that is known for its node-based architecture and object-oriented API. It was released under the MIT license and runs on most operating systems. It has many useful tools for game development, such as the scene tree editor, the script editor, a script debugger, etc. It also has an asset store from which numerous plugins can be downloaded to extend functionality. Godot contains engines for physics and lighting and many other mechanics that make game development swift and efficient.

Godot is a useful tool for developing projects such as simulations due to its long list of features, but it is nowhere near as widespread or as popular as Unity, therefore there is much less documentation and tutorials available online for it.

### *Unity (4)*

Unity is a cross-platform game engine that is widely used for a variety of applications. It was developed by Unity Technologies and released in 2005. The Unity asset store has an ever-growing catalogue of assets and tools which make project development with Unity considerably faster than many alternatives. Unity is also full of useful tools such as a debugger, a script editor, a scene editor etc.

It is extremely accessible and used globally, so there is a surplus of tutorials and online resources to learn from. These resources include plenty of sample projects full of detailed documentation which allows users to develop a detailed understanding of the underlying concepts in these projects. It also excels in real-time simulation, which is perfect for this project.

### *Unity Render Pipelines (5)*

In Unity, a project can use one of the various render pipelines. The render pipeline performs a set of operations that entail taking the contents of a scene and displaying them on the screen. Different render pipelines have different capabilities and performance, so it depends on the nature of the project. The built-in render pipeline is the default render pipeline for Unity. It has limited customisation, for general purposes. There are other render pipelines available that focus more on graphics, but this project does not centre on graphics, so it is using the built-in render pipeline.

### *Unity Navigation Mesh (6)*

NavMesh (Navigation Mesh) is a tool for mapping out the traversable areas of an environment and the paths that agents can take through this environment. The process entails rendering a mesh of the walkable areas, allowing agents to determine the shortest possible paths between locations. This helps AI look more natural as it travels through an environment. This project has autonomous agents following paths through the course of the simulation, so navigation mesh was an obvious choice to assist in the pathfinding.



### *ParticleSystem (7)*

ParticleSystem is Unity's in-built implementation of a particle system, containing a vast number of properties and methods which can be altered to get different effects. When properties are set, they are passed immediately into native code to give the best performance. ParticleSystem is used to display a wide array of items such as fire, liquids, explosions, gasses etc.

This simulation uses ParticleSystem to emulate breathing and implement the actual virus particles being expelled from infectious agents which is the method of virus transmission.

### *Behaviour Trees (8)*

Behaviour trees are a hierarchical branching system of nodes that all share a parent known as the root. They begin evaluating from the top and run through each child based on certain set conditions. They allow for an AI agent to follow a strictly defined set of rules based on the position of each node in the hierarchy. They have slowly become extremely popular, being used for the AI in well-known games such as Halo and The Sims.

Behaviour trees were perfect for this project as the agents need to have AI which made them enter the office, work, do random tasks, and eventually go home. Behaviour trees make this kind of AI possible and less complicated than other options.

### *Entity Component System (9)*

Entity Component System (ECS) is a new way to develop in Unity that focuses on data-oriented design rather than object-oriented design. It breaks the project into 3 sections:

**Entities** – The actual things in your simulation

**Components** – The data associated with these entities but organised by the data rather than by entity.

**Systems** – The behaviours that update the component data. For example, A movement system would update positions of moving entities by their velocity and time passed.

Projects using ECS have greatly improved performance, making it an extremely useful instrument for simulations with a lot going on, however, it is still in beta and can be quite unreliable and buggy.

### *C# (10)*

C# is a modern object-oriented, component orientated programming language. It was developed by Microsoft in 2000 as part of its .NET initiative and approved as an international standard in 2002. Like Unity, due to its widespread use, there is a vast number of resources available online to assist in understanding the underlying concepts. Applications made with C# are generally quite robust due to its many supportive features. Exception handling is a feature of C# which allows the detection and recovery of errors. Garbage collection is another useful feature that automatically reclaims unused memory.

It is the language that Unity scripts are mainly written in, so the coding in this project is mostly done in C#.

### *C# job system (11)*

The Unity C# Job System allows users to have multithreaded code in their projects. It integrates with Unity's native job system, so user-written code and Unity share worker threads. This ensures that there are not more threads than CPU cores. This multithreaded code can greatly improve the performance of the project. The C# job System works well with the Unity Entity Component System due to its efficient way of writing code.

The C# Job System improves performance, although it can be very confusing and there is not as much documentation or resources available online.

## 2.4. Other Research You've Done

### *COVID-19 transmission research*

There has been a great amount of research done in the last year regarding the transmission of COVID-19, and numerous factors have been found to influence the probability of transmission. Physical distancing has been shown to reduce transmission rates (12) as the infected particles can only be expelled a certain amount. (13) Masks have also been shown to reduce the particles expelled from an infectious person and reduce the chance of someone breathing in infectious particles. (14) Factors such as vitamin D levels (15) or age can determine a person's susceptibility to the virus due to the strength of their immune system. Vaccinations can reduce the risk of transmission by over 95%. (16)

Closed environments have also been found to be a contributor to secondary transmission and can lead to superspreading events. (17)

This simulation has a number of parameters that the user can change and see how they alter results. The parameters this system uses have been chosen as they have been shown to have an effect on transmission.

### *Data Visualisation*

The use of images and simulations to visualise data has been shown to help develop a greater understanding and comprehension of data than ever before. (18) Many people struggle to truly grasp the implications of raw data without some useful kind of visualisation. Some methods of visualisation do a much better job than others though.

Game techniques and mechanisms such as real-time simulations have been shown to aid in the understanding of certain topics as they are a more engaging form of learning. (19)

### *SNAPS prototype asset packs (20)*

SNAPS prototype asset packs are packs with collections of themed assets found on the Unity Asset Store. The asset prefabs contained are accurately scaled prefab objects, and a script is included in the pack which allows for these objects to snap together on all three axis in the Unity scene editor, making environment design quick and simple. This project is simulating a workplace, so an asset

pack of accurately scaled objects that scale together are perfect for speeding up environment design.

#### *Mixamo (21)*

Mixamo is a site that contains a vast number of template character models and animations which work with these models. The models contain numerous joints throughout their body making them easily animated. Both the character models and the animations can be downloaded as .fbx files, which can then be imported and used with an animator. This project contains autonomous agents that will walk around the scene so default models and animations would be useful for assisting in the implementation of animated agents.

## 2.5. Existing Final Year Projects

#### *Traffic Simulation System for Driverless Vehicles by Fionn McGuire.*

A traffic simulation system for the deployment of driverless vehicles in modern-day society by using Unity3D. The platform utilizes an interactive OSM map of Manhattan populated with both drivers and driverless vehicles. The vehicles generate a route to follow while perpetually responding to changes in the environment.

This traffic simulation is also made in Unity and contains many agents which populate an environment and have a set of behaviours to follow. This project has a lot of similar mechanics, as it is also simulating multiple agents in an environment which have a set of behaviours and have interactions.

#### *Irish Crime Data Visualisation by Max Curtis.*

A system to allow for the visualization of Ireland's crime statistics. This data is an untapped resource in its current state. This project is an application that helps users understand a mountain of data using data visualisation techniques.

This data visualisation application has some similar concepts to this project, the main one being that the application helps users develop a greater understanding of the available data. There is a vast amount of data available online about Coronavirus and its transmission but having a good visualisation can help users truly comprehend what the data implies.

## 2.6. Conclusions

In this chapter, the research done for the project was shown and presented. This research included similar existing systems, technologies related to the simulation, Coronavirus transmission research, and data visualisation research. Finally, two similar existing fourth-year projects were discussed. When the project was being planned, these technologies were researched and reviewed thoroughly to determine what would be used.

## 3. Experiment Design

### 3.1 Introduction

In this chapter, the design of the project and simulation are discussed. First, the methodology used in this project will be outlined and following this a discussion of the technical architecture will be presented. The front-end design of the system will be presented next, showing the key screens of the simulation. The back-end is also discussed with diagrams to show the design of the objects.

### 3.2. Software Methodology

Agile software methodologies focus on continuous delivery of valuable software, and the primary measure of progress is working software. (22)

The software methodology this project uses is agile scrum. Agile scrum focuses on dividing the project into sprints, which are short-timed periods in which an amount of work is set to complete, generally focusing on a specific feature of the project. Before each sprint, it is planned what work will be delivered from the sprint, and how that work will be achieved. One feature of the scrum methodology is regularly reflecting on work done and learning from it, in turn becoming more efficient as behaviour is adjusted accordingly. (23)

Scrum works well for projects with many important features, so it is perfect for this project. This project contains numerous important features which can be developed and implemented during these sprints, and then further improved with future iterations. This approach allows for many core features to be implemented successfully with some complexity, and then they can be improved and fleshed out later with less important features.

The waterfall model is a good example of a software methodology that would not work with this project. The waterfall model entails running through the entire project in a single iteration, never going back and making modifications or changes. This would not work as features of this project need to be iteratively designed, with a simple design working first, and then after some testing, reflecting, and further development, they can be revisited with a greater understanding of requirements and design.

### 3.3. Project Management

GitHub was chosen to be used for project management and keeping track of development. It is very straightforward to use, having a simple GUI program and well as a console interface. Every step of development was saved in a commit and pushed to the project, allowing for effortless version control and for making sure that no work is lost. There were multiple times during development where the project had numerous errors which were easily fixed by reverting to a previous version. GitHub also has a task board feature, allowing you to separate out the various features to be developed with tags denoting their importance and what section of the project they belong to. This was invaluable for keeping track of features during the different development stages.

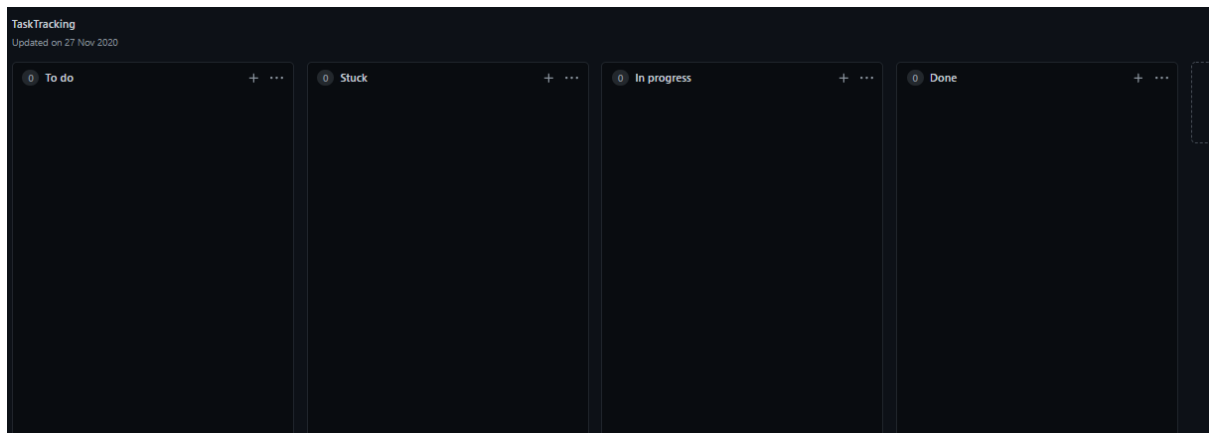


Figure 5 – Github task tracking

### 3.4. Overview of System

#### 3.4.1. Technical Architecture

As this project is entirely in Unity and C#, its technical architecture is a standalone system.

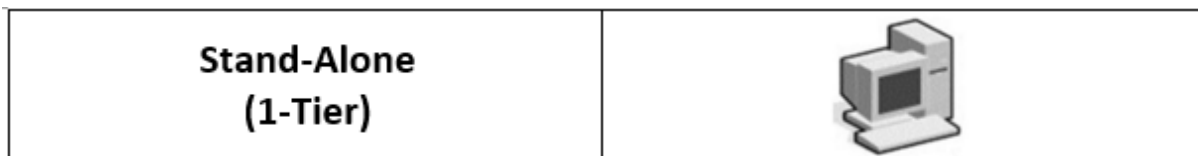


Figure 6 -Technical Architecture

#### 3.4.2. System Flowchart

The following flowchart describes general user interaction with the system. It begins with the start screen, at which point the user sets the working hours and other variables of the simulation. Once the user starts the simulation, they can alter the timescale which changes the rate at which time passes in the simulation. They can also walk around the office or switch to the office cameras. At any point, the user can press escape to go back to the start screen. Once the simulation ends, results are displayed and the user can either restart or quit the application.

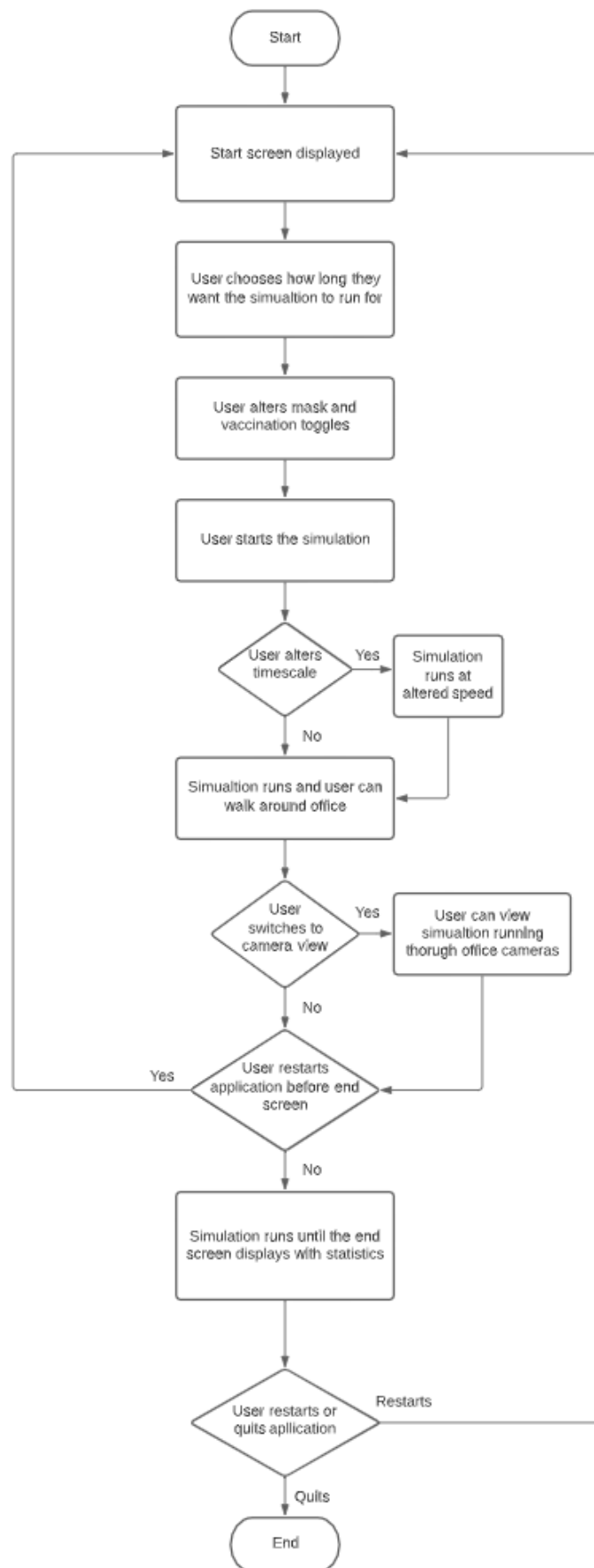


Figure 7 – System flowchart

### 3.4.3. Table of Requirements

Outlined below are the features that were decided as system requirements after the research and design of the project concluded. Each feature has a description, priority, and version of the project that they are planned to be implemented in.

ID	Name	Description	Priority	Version
1	Workplace environment	The simulation will take place in a workplace environment.	High	1.0
2	Autonomous Agents	The agents follow certain behaviours as they path find throughout the building following various feasible paths.	High	1.1
3	Utilize Unity's Navigation Mesh System	The autonomous agents in this simulation will be able to path find through the building with the help of the navigation mesh.	High	1.0
4	Infection system	Agents will either be healthy, infectious or exposed. Most spawn as healthy and can become exposed if they come in contact with the virus.	High	1.1
5	GUI with sliders	A UI will be displayed to the user showing statistics as the simulation is running.	High	1.2
6	Time Scale	The user can alter the time scale via the UI to have the simulation run faster or slower depending on their preferences.	High	1.2
7	Display Results	The user can see statistics about the number of agents exposed to the virus as the simulation runs.	High	1.1
8	Complex AI with behaviour trees	The autonomous agents will have complex ai with multiple behaviours, having them to tasks around the office.	High	1.3
9	Utilize Unity's Particle System	The infectious agents will be emitting infectious particles from their mouths with the use of Unity's particle system.	High	1.4
10	User movement and cameras.	The user will be able to walk around the scene or look through cameras throughout the scene.	Medium	1.0
12	Animated character models	The autonomous agents will have human character models with animations.	High	1.5
13	Adjustable variables	The user will be able to adjust multiple variables and see how they affect the simulation results.	High	1.6
14	End screen	The simulation will end off with a screen displaying the results from the simulation that was just run.	High	1.8
15	Medical data	The calculations made to determine whether an agent becomes infected will use data from real studies.	Medium	1.6

16	Contaminated surfaces	The infection will also be spread by surfaces that become contaminated from the virus.	High	1.7
17	Optimization	The underlying technologies will be optimized to increase the performance of the simulation.	Medium	1.9
18	Utilize Entity Component System	The entity component system is a data-oriented way of programming that significantly increases performance.	Low	1.9
19	Utilize C# Job System	The C# Job System would allow for Scripts and certain processes to be multithreaded.	Low	1.9

### 3.5. Front-End

#### 3.5.1. Key Screens

The project contains 4 main key screens, the first of which is the start screen. A small description of the simulation is displayed, and the user can alter certain variables which will affect the results of the simulation. There is a button to start the simulation or quit the application. Finally, in the bottom corner, there is a small tip explaining that the user will be able to adjust the time scale of the simulation.

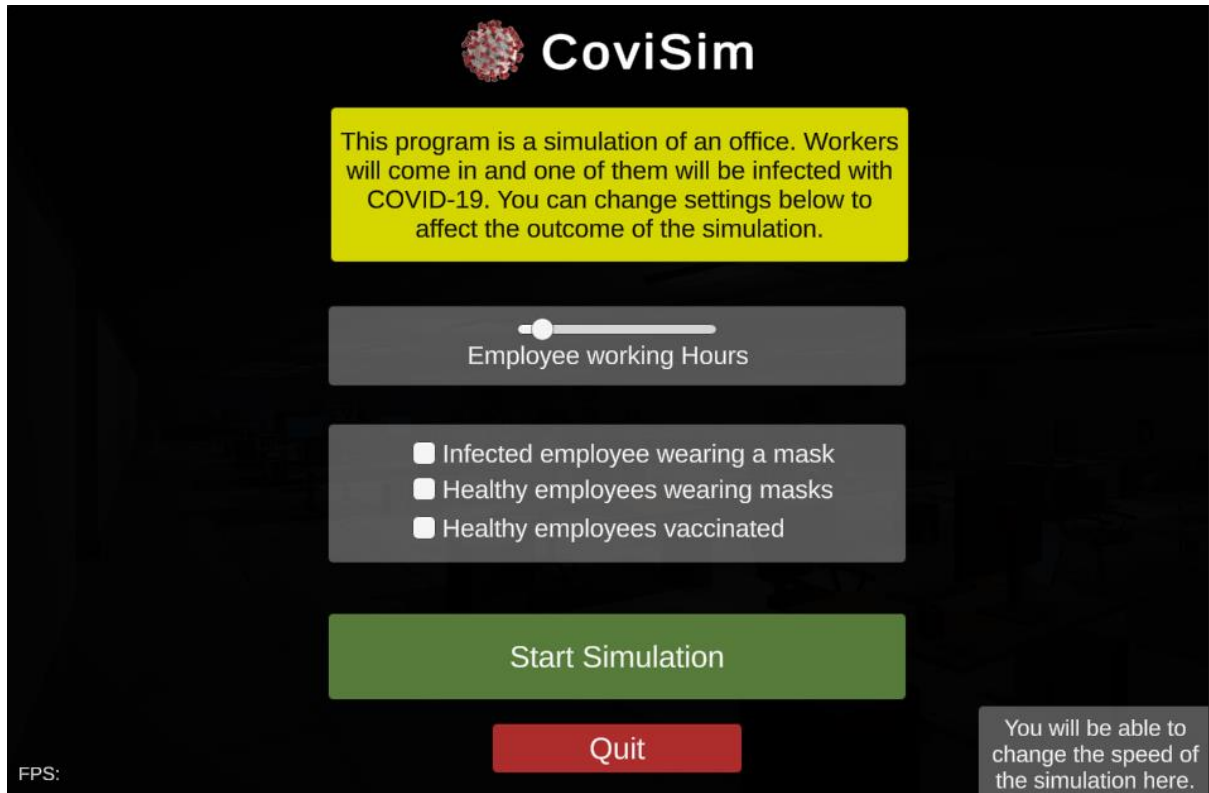


Figure 8 – Key screen 1



The second screen is the actual game view of the 3D simulation, with UI components overlaid on the screen as the simulation runs. The user is able to walk around the office with the W, A, S and D keys, and switch to the office cameras with the 2, 3 and 4 keys. A panel explains this in the top left corner of the screen. The user can also press the escape key to go back to the start screen, which is also explained in a panel just below the top left corner of the screen. Statistics from the simulation currently running are displayed in a panel on the top right of the screen, and a frames per second counter is displayed in the bottom left corner of the screen.

Finally, in the bottom right corner of the screen, a panel displays the time passed, the time left, and a slider that lets the user change the time scale of the simulation.



Figure 9 – Key screen 2

The third key screen is the office camera view. The main difference between this and the player view is that the user cannot move or walk around. The panel in the top left of the screen is also slightly different. There are 3 office cameras, which the user can cycle through, but the only difference between them is the location that they display.

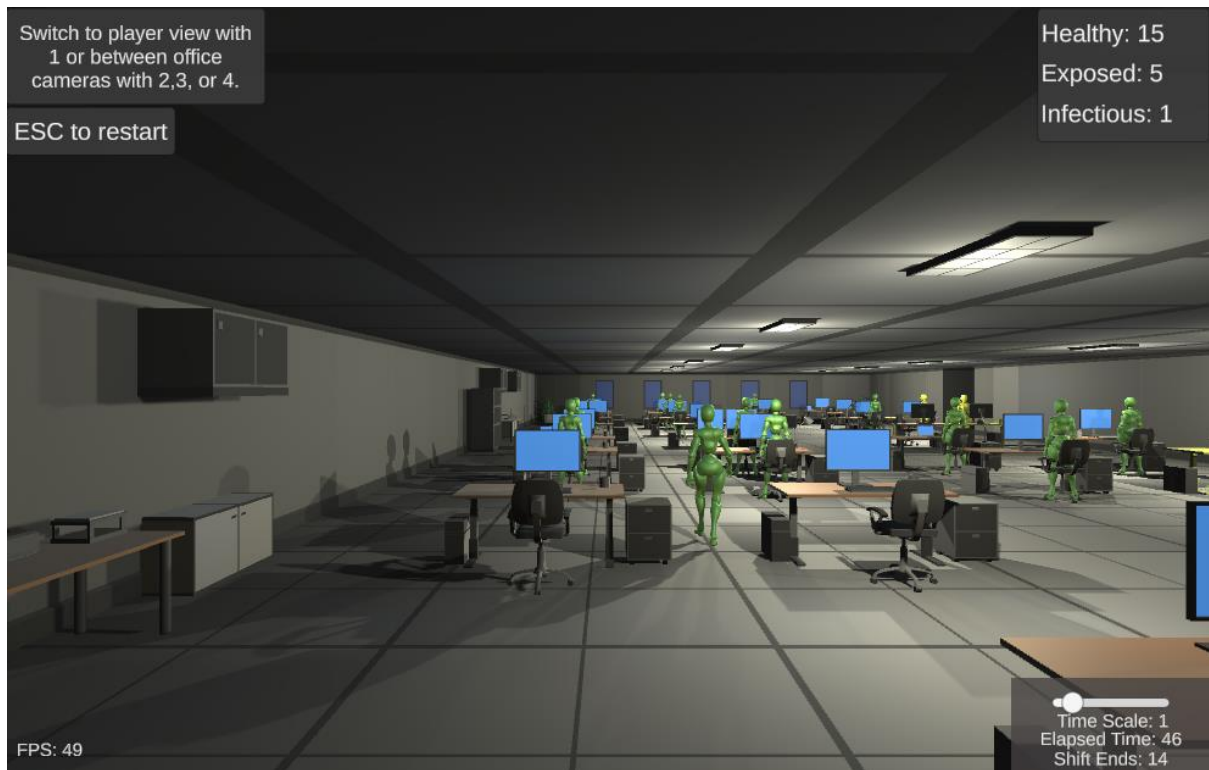


Figure 10 – Key screen 3

The fourth and final screen is the end screen, which is displayed after the workers finish their shift, and it displays statistics about the simulation that was just run allowing the user to restart or quit the simulation.

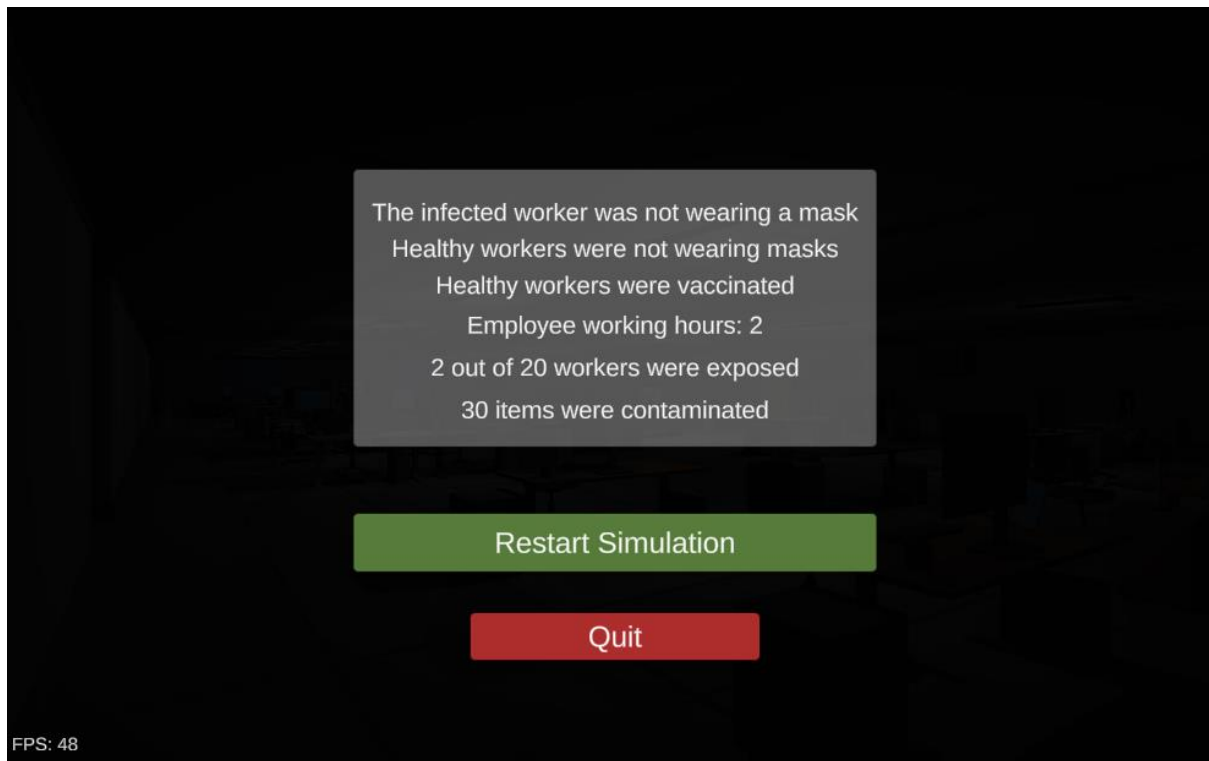


Figure 11 – Key screen 4

### 3.6. Back-End

#### 3.6.1 Behaviour Tree Diagrams

Below, the behaviour tree diagram is presented along with its index. This diagram demonstrates the design and structure of the behaviour trees that the autonomous agents will use as their artificial intelligence, moving through the various tasks in the trees based on success or failure conditions. The implementation process of these behaviour trees is thoroughly discussed throughout the experiment development chapter.

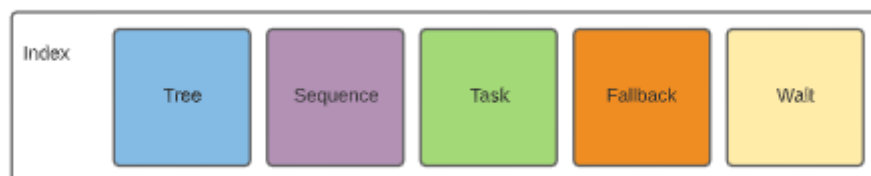


Figure 12 – Behaviour tree diagram index

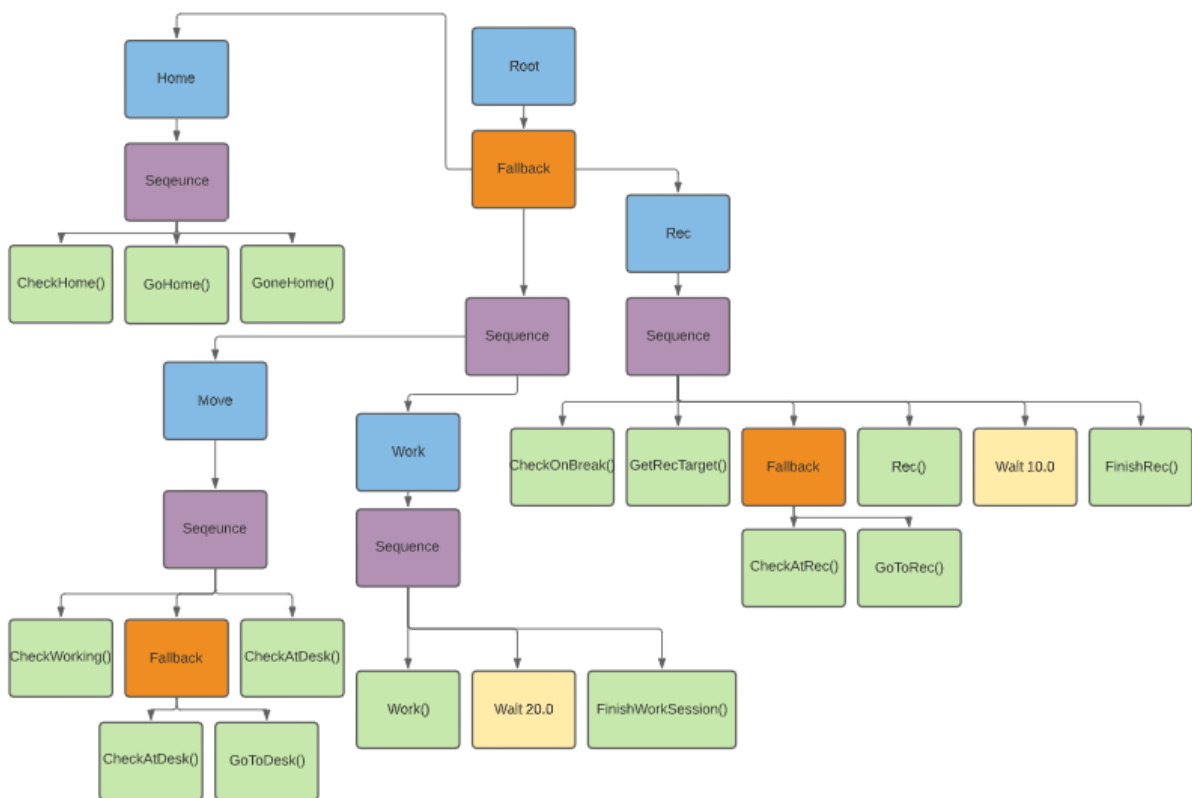


Figure 13 – Behaviour tree diagram

### 3.6.2 Class Dependencies Diagram

Below, a diagram of the class dependencies of the system is presented, showing which classes depend on other classes in the system. This screenshot was taken from Unity with a script dependency visualiser tool. There are many other system-related dependencies, although they were not included as many of them were common Unity processes.

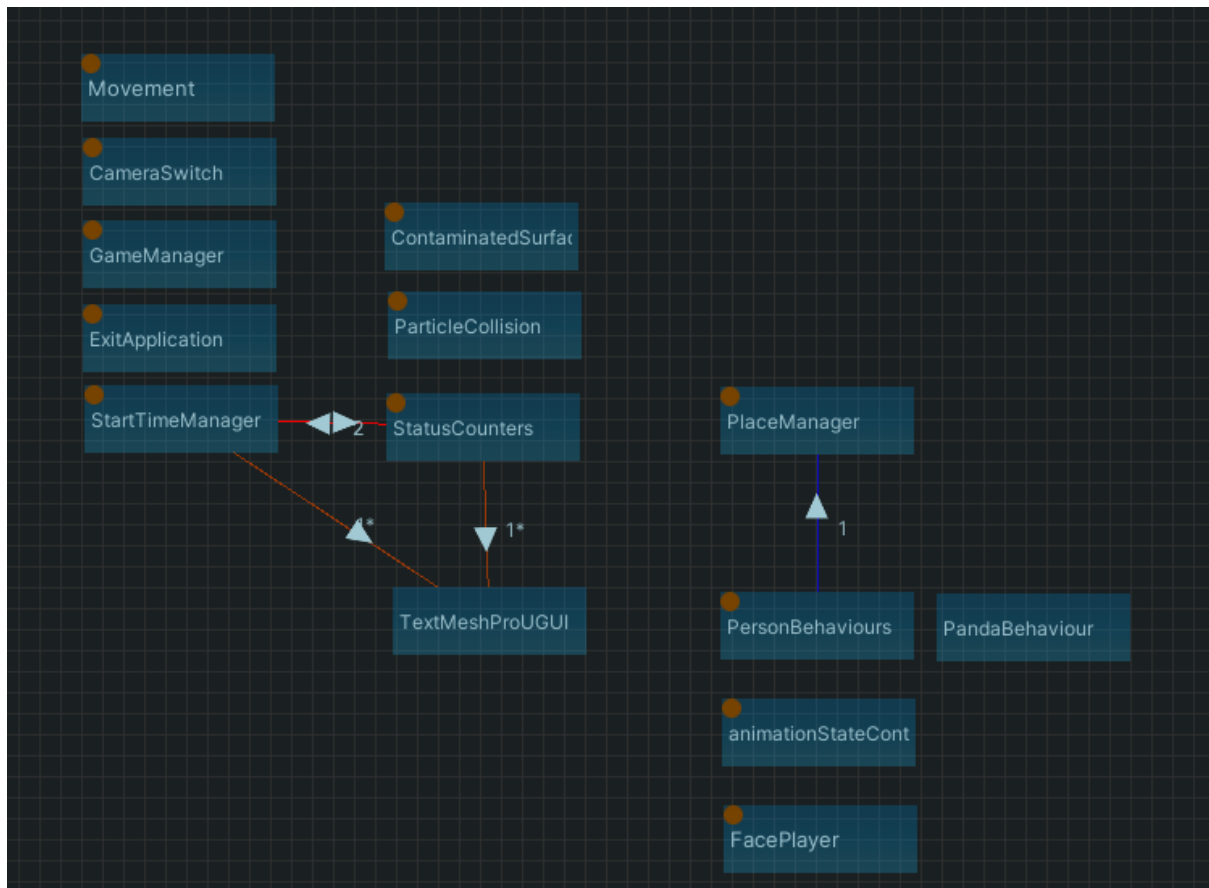


Figure 14 – Dependencies diagram

### 3.7. Conclusions

In this chapter, the design of the simulation system was presented. First, the agile scrum methodology was discussed as the approach to be used in this project. Following this, the technical architecture of the system was presented.

The front-end design of the system was presented next, showing the 2 key screens of the simulation. The back-end was also discussed with a class diagram to show the design of the objects.

## 4. Experiment Development

### 4.1. Introduction

In this chapter, the entire development process will be outlined and discussed. The chapter will begin with setting up the environment and the beginning of the development and then goes through the feature-based development cycle discussing the underlying technologies of each feature and the issues faced during the implementation of these features.

### 4.2. Environment Setup

Before development began, a GitHub repository was created and a Unity project and a “.gitignore” file were added. The “.gitignore” file was for making sure certain unnecessary files were not uploaded every time a commit was made. The Unity project was set up using Unity 2019.4.14f1, which is the 2019 long term support version. This version was chosen as it was the newest LTS release which is known to be more stable than other versions. A task board and Google Doc were also set up as a way to keep track of development.

### 4.3. Entity Component System

Due to the simulation having to have numerous calculations and behaviours run at real-time, performance was sure to be important. The Entity Component System (ECS) would be a solution to this, as it is a great way to optimize importance due to its data orientated design and multithreading capabilities.

Development began with getting some simple ECS systems working. Some online tutorials were looked at to help with this process, and although they were generally unrelated to the project, they were invaluable for achieving a greater understanding of the scope of ECS and whether it would be a viable option for this project. The ECS preview packages needed to be imported into Unity for the project to work.

These tutorials comprised of getting objects to hover and spin and have a player be able to move around and collect them, having a stats counter be constantly displayed. Traditionally in Unity, this could be achieved by attaching scripts to a couple of game objects, but with ECS rather than game objects, there were some entities in the scene with corresponding data components and separate systems which acted as the behaviours and could be multithreaded using the C# job system. This data-oriented design over the usual object-oriented design results in the program running significantly better as it is a much more efficient method of doing things.

Even though this was a small-scale test project, many issues were encountered. The main issues being the volatile nature of ECS due it still being in beta. There were frequent updates that completely changed certain aspects of the system, making a vast amount of the available online material redundant. This was detrimental to development as it is a relatively new system and attempting to develop more complex behaviours with no material available would have would not

have been possible during the length of this project. Another issue with ECS still being in beta was the fact that there were numerous bugs with every release, which could end up with certain not being possible or other features breaking with no feasible fixes.

Due to these issues, it was decided that the project would be developed in Unity without ECS, with the possibility of porting it over in future work.

#### 4.4. SNAPS Tool and Navigation Mesh

After the decision to abandon ECS was made, a new project was created in the Unity 2019 LTS and added to the GitHub repository. This project was made with the standard render pipeline as graphics are not a focus. During the research for this project, an asset in the Unity Asset Store called “SNAPS” which aids in the process of environment design was discovered. This tool contains themed asset packs containing accurately scaled prefab objects, and a script to allow for these objects to snap together on all three axis in the Unity scene editor, making environment design much more efficient. Each prototype pack follows a specific theme, and due to the fact that the simulation takes place in a workplace, the SNAPS Office asset pack was imported to the project.

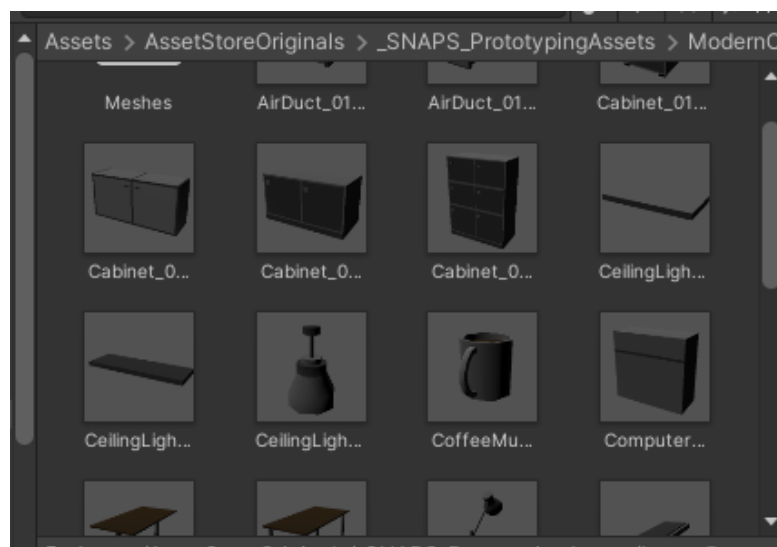


Figure 15 – Sample assets

These prefab objects were placed in the scene and a simple office design was made. With an environment constructed, navigation mesh could now be implemented.

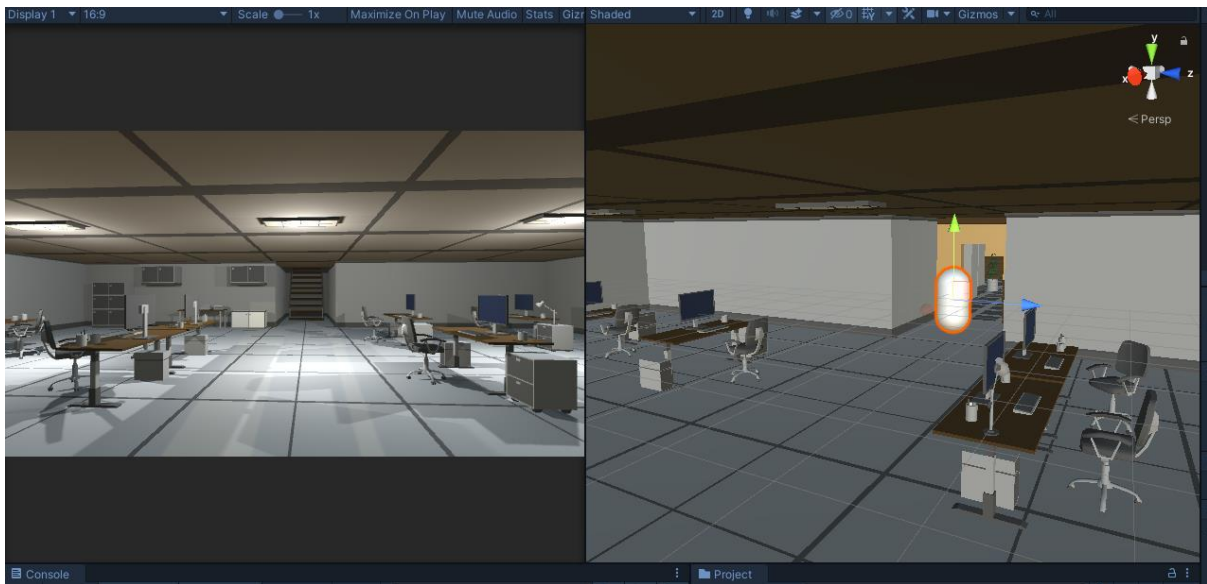


Figure 16 – Office environment

In Unity 2019, implementing a navigation mesh is relatively straightforward once the scene is correctly set up. Any object that should be walkable by agents must be set to “navigation static”, and then they are all used in a process called “baking” which returns a map of all the walkable surfaces in the scene. This map can be shown in the Unity scene editor with Gizmos turned on.

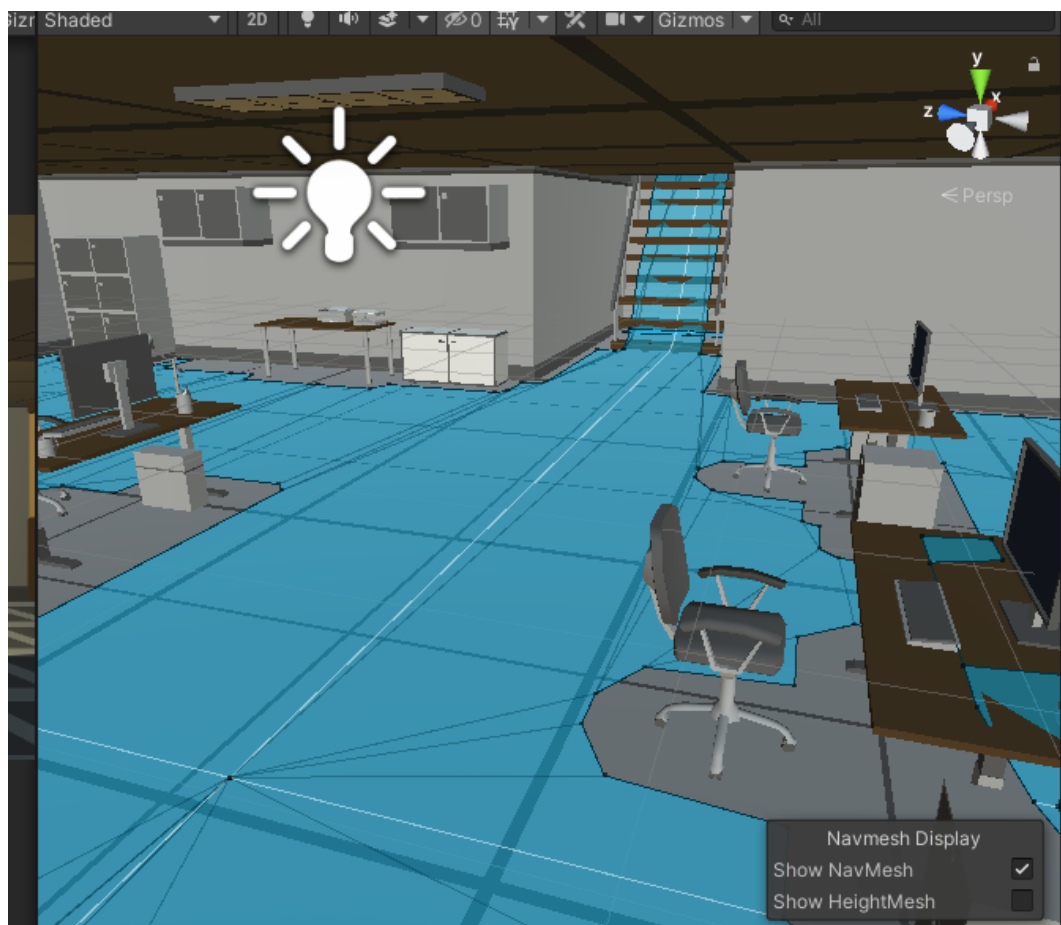


Figure 17 - NavMesh



With a navigation mesh baked, objects with the “NavMesh Agent” component will be able to traverse the environment, finding the shortest paths to their destinations. To test this, a capsule was added to the scene with a NavMesh Agent component. A script was then written which takes the location clicked by the user by sending out a Raycast into the scene and sets it as the capsule’s destination. The update method is run every frame, so every frame the project was checking for any user input. This feature was presented in the prototype to show the navigation mesh working and soon removed.

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

public class AgentAI : MonoBehaviour
{
    private NavMeshAgent _navMeshAgent;

    private void Awake() => _navMeshAgent = GetComponent<NavMeshAgent>();

    // Update is called once per frame
    void Update()
    {
        if (Input.GetMouseButtonDown(0) &&
            Physics.Raycast(Camera.main.ScreenPointToRay(Input.mousePosition), out var hitInfo))
            _navMeshAgent.SetDestination(hitInfo.point);
    }
}
```

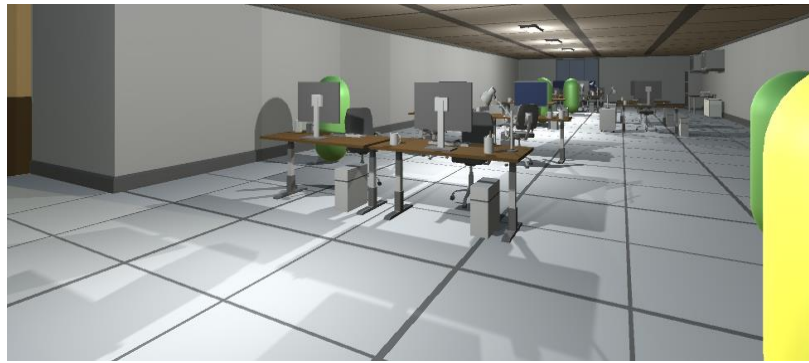
Figure 18 – Movement Code

Another capsule was added to the scene with a separate movement script to allow the player to move around the office with keyboard controls, and the main camera was attached to this object. This script was also a placeholder script to show the office environment and would later be replaced by a more advanced script.

#### 4.5. Simple Path Following and Infection Model

The next step of development was to have a simplistic prototype of the final project, with autonomous agents entering the office with a chance of becoming exposed to the virus. This was started off by adding more capsule objects with NavMesh Agent components to the scene and making a path following behaviour. Some random places in the office were added to a list of locations and a script was made which simply iterated through this list, setting these locations as destinations for the agents. This emulated the agents walking around the office and would soon be removed but was useful for the infection mechanics to take place.

With agents walking around the office, a simple infection model could be implemented. This was done using tags, with all but one of the agents being tagged as healthy and the final agent being tagged as infectious. Each agent had a capsule collider, so a simple script was made which changed the agent's tag from healthy to exposed if anything tagged as infectious triggered their collider. Three corresponding materials were added to the project, with green representing healthy, yellow representing exposed and red representing infectious. The agents were set to their respective material and the script was modified so it would also change the agent's material as well as their tag for visual clarity.



*Figure 19 – Capsule agents*

The office environment was also extended so the agents had more room to walk around the scene.

#### 4.6. UI and Timescale

With a simple infection model working, some status counters which display statistics as the simulation is running were added to the corner of the screen.



*Figure 20 – Status counter displays*

These counters display the current amount of healthy, exposed and infectious agents currently in the scene. This was done by making a list of all objects in the scene with their respective tag and then displaying the length of each list.

```
// Update is called once per frame
void Update()
{
    //Finding all of the objects with their respective tags and constantly updating the display
    GameObject[] exposed = GameObject.FindGameObjectsWithTag("Exposed");
    exposedText.text="Exposed: " + exposed.Length;

    GameObject[] healthy = GameObject.FindGameObjectsWithTag("Healthy");
    healthyText.text="Healthy: " + healthy.Length;

    GameObject[] infectious = GameObject.FindGameObjectsWithTag("Infectious");
    infectiousText.text="Infectious: " + infectious.Length;
}
```

Figure 21 – Stats counter code

The other element that was added to the screen was a timescale slider. This lets the user change the speed at which the simulation runs, so if they were to set the slider to 10, the simulation would run at times ten-speed. This was done by multiplying the “Time.timescale” variable by the value of the slider. “Time.timescale” is a Unity variable that controls the rate at which time passes.

```
public void SetTime (Slider slider)
{
    //Changes the timescale to the value of the slider
    Time.timeScale = slider.value;
}
```

Figure 22 – Altering timescale

Finally, a simple start screen with a button was added which is on the screen before the simulation begins. Once the user clicks the button, the player movement, NavMesh agents and status counters are set to active. This was done by simply having them disabled by default, and having the buttons OnClick action set them to active.

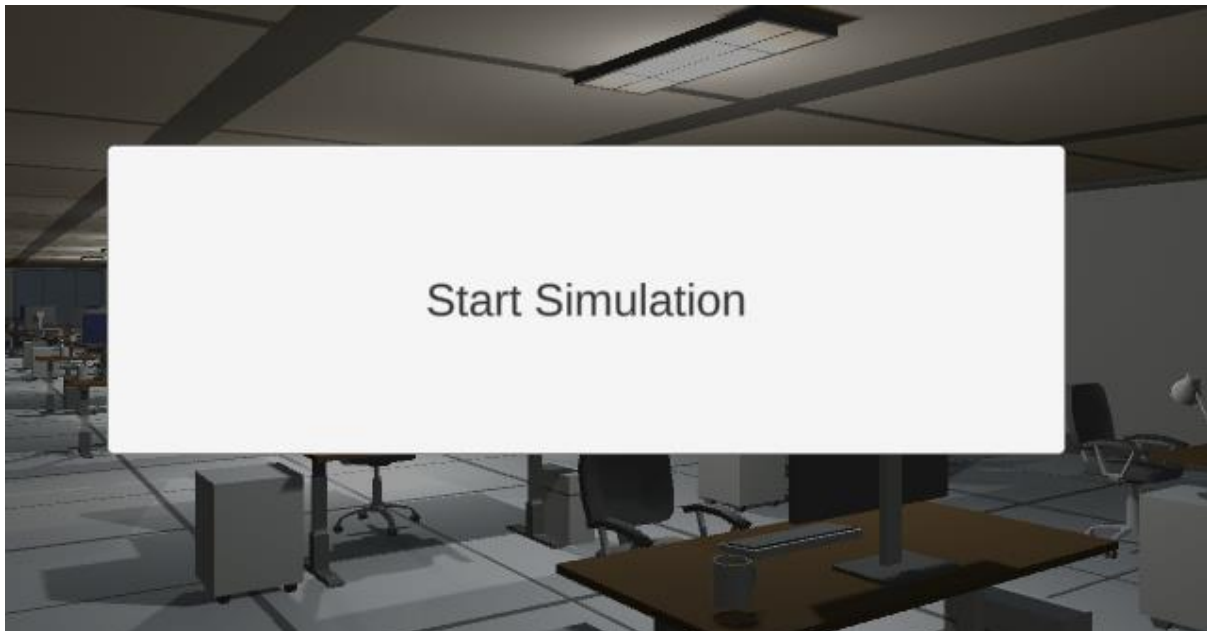


Figure 23 – Start button

## 4.7. Behaviour Trees

With the simulation having agents walking around, the virus spreading between them and status counters on the screen, behaviour trees were implemented and the path following behaviour was removed to make the agents follow a set list of behaviours rather than just follow the same path.

Before implementing behaviour trees, a script was written which is run on “Awake” which means it is run when the scene begins loading. This script searches the scene for all objects tagged as desks and adds them to a list. It does the same for all objects tagged as rec points. The agents will be able to take their destinations from these lists once behaviour trees are implemented.

```
public List<GameObject> availableDesks;
3 references
public List<GameObject> recPoints;

2 references
public Transform Home;

0 references
void Awake()
{
    PopulateLists();
}

// Update is called once per frame
0 references
void Update()
{
}

1 reference
void PopulateLists()
{
    //Called on awake, this finds all of the desks in the scene and populates a list
    GameObject[] desks = GameObject.FindGameObjectsWithTag("Chair");
    foreach (var ob in desks)
    {
        availableDesks.Add(ob);
    }
    //Called on awake, this finds all of the recpoints in the scene and populates a list
    GameObject[] recs = GameObject.FindGameObjectsWithTag("Rec");
    foreach (var ob in recs)
    {
        recPoints.Add(ob);
    }
}
```

Figure 24 – Place manager

The first step of implementing behaviour trees was to import the Panda BT asset from the Unity Asset Store. This asset contained a behaviour tree engine which allowed the creation of behaviour tree files that could be attached to agents as components to make them follow a set of logic written in the behaviour script.

A behaviour tree file and a behaviour script were both created after planning out and drawing a diagram of how the agents should behave. The simple plan was to have them enter the office and go to their desk, and after a set amount of time spent working, they would have a chance of leaving their desk to do a random task in the office and then go back and continue working.

```

tree("Root")
  fallback
    tree("Work")
    tree("Rec")

tree("Work")
  sequence
    CheckWorking()
    fallback
      CheckAtDesk()
      GoToDesk()
    Work()
    Wait 20.0
    FinishWorkSession()

tree("Rec")
  sequence
    GetRecTarget()
    fallback
      CheckAtRec()
      GoToRec()
    Rec()
    Wait 10.0
    FinishRec()

```

Figure 25 – Behaviour tree

This is the structure of the behaviour tree, split into two trees, the work tree making them go to their desk and the rec tree making them go do a random task. Each element or “task” in these trees is a method in the behaviour script which is denoted as a [Task].

Once each agent is enabled by the start button, their Start method is called. In this method, the agent gets a random desk from the list mentioned above and then removes it from the list so no agents will go to the same desk.

```

//Each agent gets assigned a random desk and removes it from the list so no agents will get the same desk
assignedDesk = placeManager.availableDesks[Random.Range(0, placeManager.availableDesks.Count-1)].transform;
placeManager.availableDesks.Remove(assignedDesk.gameObject);

```

Figure 26 – Getting agents desk

After this, they run through their various tasks based on their current position in the tree. The first Task checks if they should be working, or if they should be going to a rec point. If they indeed should be working, the task is set to success and they continue in the working tree, otherwise they fall back to the rec tree.

```

[Task]
0 references
void CheckWorking()
{
    if(workingCheck == true)
    {
        Task.current.Succeed();
    }
    else
    {
        Task.current.Fail();
    }
}

```

Figure 27 – Check working behaviour

If they continue in the work tree, they check if they are at their desk, and if they are not, their target is set to their assigned desk. This makes the agent move towards their assigned desk using the navigation mesh. It was unknown at this stage of development, but this behaviour is slightly bugged as the agent immediately succeeds in this task and goes straight to the next one. This is corrected at a later stage of development.

```

//Checking if the agent has reached their desk
[Task]
0 references
void CheckAtDesk()
{
    if(Vector3.Distance(this.transform.position, assignedDesk.transform.position) < 0.1f)
    {
        Task.current.Succeed();
    }
    else
    {
        Task.current.Fail();
    }
}

//Setting the agents desk as their destination
[Task]
0 references
void GoToDesk()
{
    target = assignedDesk;
    _navMeshAgent.destination = target.transform.position;
    Task.current.Succeed();
}

```

Figure 28 – Check and go to desk behaviour

At this point, the agent is at their desk and begins working. A 20 second timer starts before the next behaviour is started. Once this is over, there is a random number generated resulting in a 1 in 5 chance that a Boolean variable will be set to false, which results in the agent failing the check working behaviour and falling back into the rec tree.

```

//1 in 5 chance that the agent will be set to go to a rec point before their next work session
[Task]
0 references
void FinishWorkSession()
{
    if(Random.Range(0, 5) == 1)
    {
        workingCheck = false;
    }
    Task.current.Succeed();
}

```

Figure 29 – Finish working behaviour

The Rec tree is constructed in a similar way, starting off by getting a random rec target from the list of rec points mentioned above. Each of these points represents a different task, e.g., retrieving a file or printing something off.

```

//Set the target to be a random rec point in the office
[Task]
0 references
void GetRecTarget()
{
    target = placeManager.recPoints[Random.Range(0, placeManager.recPoints.Count-1)].transform;
    Task.current.Succeed();
}

```

Figure 30 – Get rec target behaviour

Similarly, to the work tree, they check if they are at their rec target, and set their destination to their current rec target, which makes the agent move to their rec target along the navigation mesh.

```

//Checking if the agent arrived at the rec point
[Task]
0 references
void CheckAtRec()
{
    if(Vector3.Distance(this.transform.position, target.transform.position) < 1)
    {
        Task.current.Succeed();
    }
    else
    {
        Task.current.Fail();
    }
}

//Setting the agents destination to the rec target
[Task]
0 references
void GoToRec()
{
    _navMeshAgent.destination = target.transform.position;
    Task.current.Succeed();
}

```

Figure 31 – Check and go to rec point behaviour

Now that they are at their rec point, a 10 second timer is started before the next behaviour begins. Once this finishes, the Boolean variable is set back to true, making them pass the check working behaviour and go back into the working tree.

```
//After the 10 second wait, set their workingcheck back to true so they will go back to working
[Task]
0 references
void FinishRec()
{
    workingCheck = true;
    Task.current.Succeed();
}
```

Figure 32 – Finish rec behaviour

These trees result in a relatively accurate simulation of how workers would generally behave as they work throughout the day in a workplace.

#### 4.8. Particle Simulation

At this stage in development, an interview was conducted with Dr Zach Tan, who suggested that the two main modes of transmission of the virus that should be focused on are aerosol transmission and contaminated surfaces.

Implementation of a particle system to simulate breathing then began. A particle system component was attached to the infectious agent and set to “Send Collision Messages” which would allow a script to iterate through objects that the particles collide with. A script was then written and attached which did exactly that.

```
// Start is called before the first frame update
0 references
void Start()
{
    particle = GetComponent<ParticleSystem>();
    collisionEvents = new List<ParticleCollisionEvent>();
}
0 references
private void OnParticleCollision(GameObject other)
{
    int numCollisionEvents = particle.GetCollisionEvents(other, collisionEvents);
    Rigidbody rb = other.GetComponent<Rigidbody>();
    int i = 0;
    //Iterating through all of the collisions
    while(i < numCollisionEvents)
    {
```

Figure 33 – Start of particle collision script

The particle system and a list of all the particle collision events were passed in, allowing the script to constantly check how many particle collision events there are, and then iterate through each of them.



At this stage, as the script is iterating through the collisions, it splits into 2 sections based on an if statement. The if statement checks whether the object that the particle collided with has a Rigidbody or not. (Rigidbody is a component that puts the object they are attached to under the control of Unity's physics engine). If the object does have a Rigidbody, it is treated as another agent. If the object does not have a Rigidbody, it is treated as an object.

If it succeeds the if check and is treated as an agent, it is made sure that the agent is not already an infectious or exposed agent. It then makes sure that this agent has not had a collision event in the past 2 seconds. This is to prevent hundreds of unnecessary events from taking place all on one agent in the space of a second or two. It then sets the tag and material of the agent to exposed.

If it fails the if check and is treated as an object, it is first made sure that the object is not tagged as ceiling, floor, wall or player. The decision to not have the ceilings, floors and walls be included here was made as it ended up being unclear what exactly was happening when the whole environment changed colour. These surfaces becoming contaminated in a real workplace would not have much if any effect on transmission anyway as workers would not be interacting with or touching them.

After this, the object tag and material are set to contaminated and exposed, respectfully. At this stage, a bug was discovered in which only one of many materials was being changed on objects with multiple materials. This was fixed by getting the number of materials on the object and creating a new list of materials with the same length, and then changing all of the materials to the list just created.

```
//Setting tag and all materials
other.gameObject.tag = "Contaminated";
Material[] newMaterials = other.GetComponent<Renderer>().materials;
for(int j=0; j<newMaterials.Length;j++)
{
    newMaterials[j] = Exposed;
}
other.GetComponent<Renderer>().materials = newMaterials;
```

*Figure 34 – Setting all materials of an object*

Particle systems in Unity have an extremely large number of settings that can be changed, so there was a lot of trial and error in the process of making it look better. The lifetime and speed determined how fast the particles would be emitted and how long they would last until they dissipated. These values were adjusted so the particles would spread to a distance close to about 2 meters, but this was relatively inconsistent. The simulation space was set to world, letting the particles spawn into the world and not move with the agent as they walked, as this would look and be very unrealistic. The particles were set to emit in a cone-like shape from the mouth area of the agent, with settings such as angle and radius tailored in an attempt to be as realistic as possible. Finally, noise was added to the particle system on a random graph between two curves, to more accurately capture how particles would be emitted from the mouth and get moved around by air the further they get from the mouth. The size of the particles was also set to extremely small, they would be even smaller in a real transmission scenario but they were set to this size so the user could still see them.

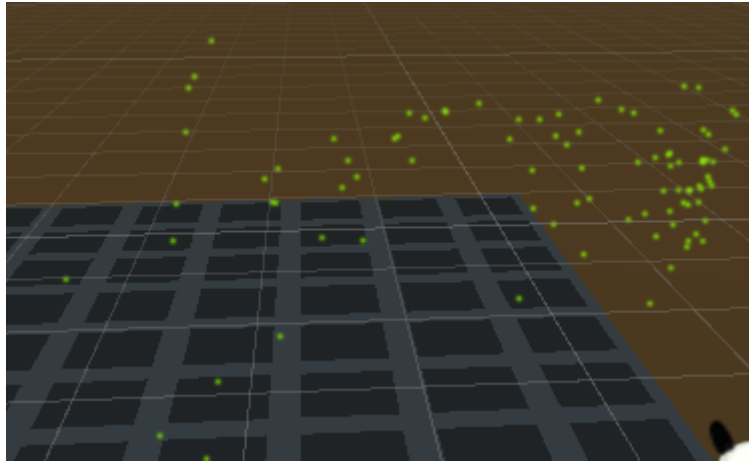


Figure 35 – Breath particle system – coming from the right

#### 4.9. Cameras and User Movement

To add more intractability to the simulation, camera switching was added to the scene, allowing the user to look through various office cameras. This was a relatively simple process, entailing adding multiple cameras to the scene and making a script allowing the user to switch between them based on their input by enabling and disabling certain objects. A panel of text in the corner of the screen was added to explain this to the user.

```
void Update()
{
    // If statements which turn on/off panels/cameras based on which camera the user is looking through
    if(Input.GetButtonDown("Switch1"))
    {
        panel1.SetActive(true);
        panel2.SetActive(false);
        camera1.SetActive(true);
        camera2.SetActive(false);
        camera3.SetActive(false);
        camera4.SetActive(false);
    }
    if(Input.GetButtonDown("Switch2"))
    {
        panel1.SetActive(false);
        panel2.SetActive(true);
        camera1.SetActive(false);
        camera2.SetActive(true);
        camera3.SetActive(false);
        camera4.SetActive(false);
    }
    if(Input.GetButtonDown("Switch3"))
    {
        panel1.SetActive(false);
        panel2.SetActive(true);
        camera1.SetActive(false);
        camera2.SetActive(false);
        camera3.SetActive(true);
        camera4.SetActive(false);
    }
    if(Input.GetButtonDown("Switch4"))
    {
        panel1.SetActive(false);
        panel2.SetActive(true);
        camera1.SetActive(false);
        camera2.SetActive(false);
        camera3.SetActive(false);
        camera4.SetActive(true);
    }
}
```

Figure 36 – Camera switching script

The player movement script was also improved. Due to the user being able to alter the rate at which time passes in the simulation, the user movement script had to be independent of the timescale.

This was achieved by dividing by 1/timescale in the movement calculations. This meant the user could now walk at the same speed no matter what the timescale is set to.

```
0 references
void FixedUpdate()
{
    //Getting rotation
    Vector3 moveRotation = new Vector3(0, Input.GetAxisRaw("Horizontal") * rotSpeed, 0);
    //Getting movement
    moveDirection = new Vector3(0.0f, gravAmount, Input.GetAxis("Vertical"));
    moveDirection = this.transform.TransformDirection(moveDirection) * walkSpeed * Time.deltaTime;

    //Apply changes to the character controller
    characterController.Move(moveDirection * (Time.deltaTime * 1 / Time.timeScale));
    this.transform.Rotate(moveRotation * (Time.deltaTime * 1 / Time.timeScale));
}
```

Figure 37 – Timescale independent movement script

The office environment was again expanded, and more desks and agents were added to the scene so it would better resemble a busy workplace.

#### 4.10. Animated Models

Up until this stage of development, the autonomous agents' models were just placeholder capsule objects and they had no movement animations. To make the simulation more realistic and easier to understand, human character models with walking and idle animations were implemented. The first step of this was to download and import a basic human character model and the two basic animations from Mixamo, a site with a variety of template models and corresponding animations.

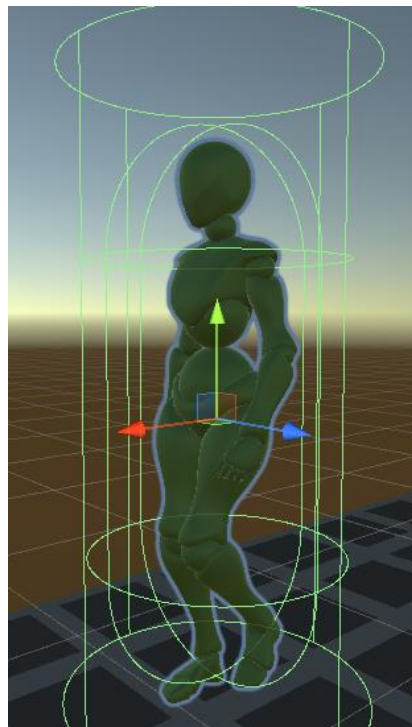


Figure 38 – Character model

A new agent prefab was then created using this model and an animator was attached. The animator was designed so the agent's animations would transition between the imported idle and walking animations based on a Boolean variable.

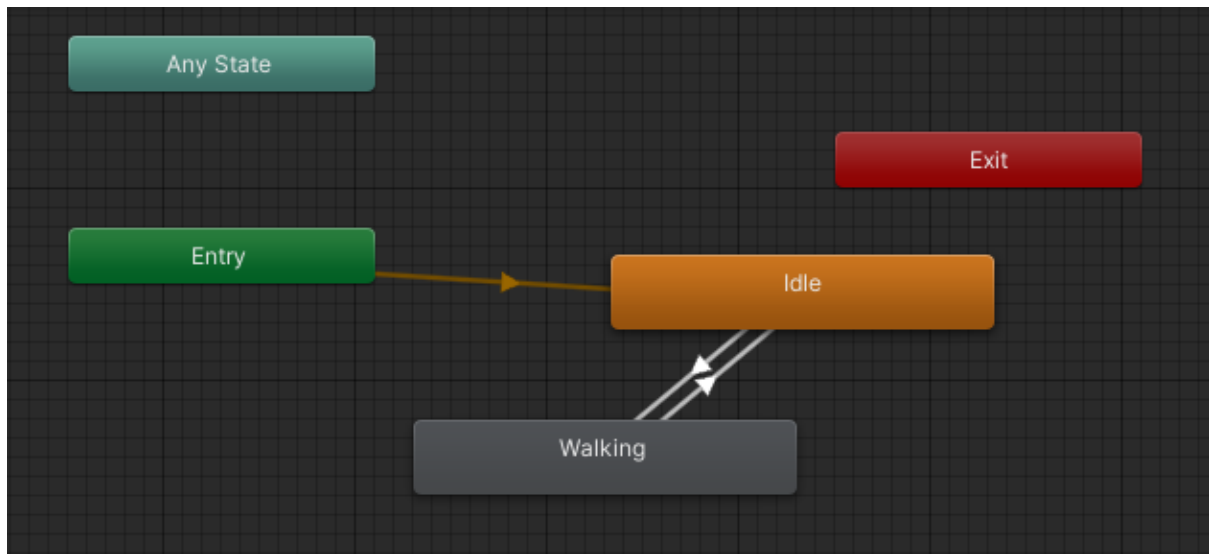


Figure 39 - Animator

An animation state controller script was then written and attached to the agent. This script constantly checks the magnitude of the agent, and if it is above a set threshold, the walking check Boolean is set to true which makes the animator play the walking animation. Otherwise, if the magnitude is below the threshold, the walking check Boolean is set to false which makes the animator play the idle animation.

```
public class animationStateController : MonoBehaviour
{
    4 references
    Animator animator;
    4 references
    int isWalkingHash;

    // Start is called before the first frame update
    0 references
    void Start()
    {
        animator = GetComponent<Animator>();
        isWalkingHash = Animator.StringToHash("isWalking");// To improve performance
    }

    // Update is called once per frame
    0 references
    void Update()
    {
        bool isWalking = animator.GetBool(isWalkingHash);

        if (!isWalking && this.GetComponent<NavMeshAgent>().velocity.magnitude > 0.1f)
        {
            animator.SetBool(isWalkingHash, true); // If they're moving, play animation
        }
        if (isWalking && !(this.GetComponent<NavMeshAgent>().velocity.magnitude > 0.1f))
        {
            animator.SetBool(isWalkingHash, false); // If they stop moving, switch to idle
        }
    }
}
```

Figure 40 – Animation state controller

With the agents now having real models, code was added to the working behaviour which made the agent face the same direction their chair is pointing in, so they would face their desk once they arrived.

```
[Task]
0 references
void Work()
{
    Vector3 lookDirection = -assignedDesk.transform.forward;
    transform.rotation = Quaternion.LookRotation(lookDirection, transform.up);
    Task.current.Succeed();
}
```

You, a month ago • Added Behaviour Trees For Agent AIs

Figure 41 – Make agent face desk

After the successful implementation of character models and getting the agents to face their desk, a bug was discovered in the behaviour tree. As soon as the agents began moving to their desk, they went straight to the work behaviour, so they did not face their desk when they eventually arrived. This was fixed by rearranging the trees and making a move tree so the agents would only begin working once they arrived.

```
tree("Work")
sequence
    Work()
    Wait 20.0
    FinishWorkSession()

tree("Move")
sequence
    CheckWorking()
    fallback
        CheckAtDesk()
        GoToDesk()
        CheckAtDesk()

tree("Rec")
sequence
    CheckOnBreak()
    GetRecTarget()
    fallback
        CheckAtRec()
        GoToRec()
    Rec()
    Wait 10.0
    FinishRec()
```

Figure 42 – Updated behaviour tree

Due to the agents no longer having a single material, the particle collision script was updated to change all the agent's materials, using similar code from when the material of an object is changed.

```
//Changing all materials to exposed for visual confirmation
SkinnedMeshRenderer[] newMeshRenderer = other.GetComponentsInChildren<SkinnedMeshRenderer>();
foreach(var m in newMeshRenderer)
{
    m.material = Exposed;
}
other.gameObject.tag = "Exposed";//For exposed counter
```

Figure 43- Changing agents material

#### 4.11. Working Hours and End Screen

Now that the simulation has most of the core concepts implemented, a working hours mechanic and end screen were added. To begin implementing the concept of working hours, the behaviours were adjusted one final time. The concept was to have the agents only work in the office for a set amount of time, and be constantly checking if that time had ended, at which point they would leave the office. This was done by adding another behaviour tree which the agents move to and check every time they finish another tree.

```
tree("Root")
  fallback
    tree("Home")
    sequence
      tree("Move")
      tree("Work")
      tree("Rec")

tree("Work")
  sequence
    Work()
    Wait 20.0
    FinishWorkSession()

tree("Move")
  sequence
    CheckWorking()
    fallback
      CheckAtDesk()
      GoToDesk()
    CheckAtDesk()

tree("Rec")
  sequence
    CheckOnBreak()
    GetRecTarget()
    fallback
      CheckAtRec()
      GoToRec()
    Rec()
    Wait 10.0
    FinishRec()

tree("Home")
  sequence
    CheckHome()
    GoHome()
    GoneHome()
```

Figure 44 – Final behaviour tree

An empty game object was added to the scene outside the front door of the office, so once the set time has passed, the agent's target is set to this location. Then once they get near the front door the agent's colliders are disabled to prevent any infection from taking place after they leave the office.

```
//Checking if the time passed has gona above working hours
[Task]
0 references
void CheckHome()
{
    if(homeTime < (Time.time - startTime))
    {
        Task.current.Succeed();
    }
    else
    {
        Task.current.Fail();
    }
}

//Set the agents destination to home
[Task]
0 references
void GoHome()
{
    _navMeshAgent.destination = placeManager.Home.transform.position;
    Task.current.Succeed();
}

//Once the agent has gotten out of the main office, turn off their collider
[Task]
0 references
void GoneHome()
{
    if(Vector3.Distance(this.transform.position, placeManager.Home.transform.position) < 42)
    {
        gameObject.GetComponent<CapsuleCollider>().enabled = false;
    }
    Task.current.Succeed();
}
```

Figure 45 – Working hours behaviour

With the agents entering the scene at the beginning of the simulation and leaving after a set time, a slider was added to the start screen to allow the user to change the length of time the agents work in the office. This was easily done by having the agent prefab reference the value of this slider when it sets the working hours once the agent is enabled by the start button. The time at which the agents were enabled also had to be captured, so the time spent on the start screen was not considered in the check.

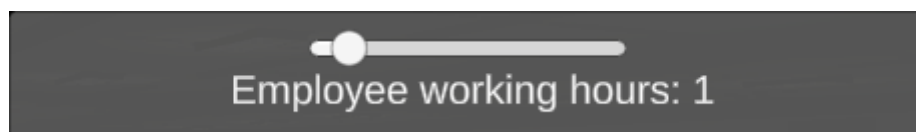


Figure 46 – Working hours slider

For the sake of user experience, the value on the slider was multiplied by 60 and then the working hours were set to that number of seconds. So, if the user selected 2 hours, the actual time would be set to 2 minutes.

An end screen was then added for when agents leave the office. This was implemented by capturing the time at which the agents were enabled and checking if the set working hours had passed. An extra 25 seconds were added to the working hours for this check to give the agents enough time to finish their current task and then walk to the door. Once this check succeeds, the other user interface (UI) elements are disabled and an end screen is displayed, which displays the number of agents that were set to exposed during the simulation. This was done by finding all the objects in the scene with the exposed tag and displaying the value. The code has a check to make sure it is only run once so values will not change once they are added to the end screen.

```
void Update()
{
    //25 seconds after the workers shift has finished, giving them time to leave
    if((statusCounters.WorkingHours.value + 25) < (Time.time - startTime) && runOnce == 0)
    {
        //Only run following code once as values will not change
        runOnce = 1;
        //Disabling other GUI and showing end screen
        timePanel.SetActive(false);
        statsPanel.SetActive(false);
        camPanel1.SetActive(false);
        camPanel2.SetActive(false);
        tipPanel.SetActive(false);

        endPanel.SetActive(true);
        //Updating the end screen stats
        GameObject[] exposedList = GameObject.FindGameObjectsWithTag("Exposed");
        exposedFinal.text = exposedList.Length + " out of 20 workers were exposed";
    }
}
```

Figure 47 – End screen code

Buttons to restart or quit the application were then also added to the end screen. The quit button calls a method that quits the application, or if it is open in the Unity editor it just “unplays” the scene.

```
//Called from the "exit" button on start and end screen. Closes program
0 references
public void Quit()
{
    Application.Quit();
    #if UNITY_EDITOR
    UnityEditor.EditorApplication.isPlaying = false;
    #endif
}
```

Figure 48 – Quit application code

The restart button just calls a method that gets the name of the scene currently running and loads it again.



```

public void RestartScene()
{
    //Reloads the scene
    string currentSceneName = SceneManager.GetActiveScene().name;
    SceneManager.LoadScene(currentSceneName, LoadSceneMode.Single);
}

```

Figure 49 – Restart scene code

During this stage of development, a bug began occurring in which some of the particle collisions were not registering. In an effort to fix this, the Unity version was updated to Unity 2020.3.1f1, which is the 2020 LTS release of Unity. After this, some debugging and further research were done and the solution was found, which was to set all the agents to “isKinematic”. This controls whether the agent’s Rigidbody is affected by physics or not and was most likely accidentally unset earlier in development.

#### 4.12. Masks and Vaccines

To add more intractability, some more options were added to the start screen which alter certain features of the simulation. The first toggle being whether the infectious agent is wearing a mask or not, then a toggle for whether the healthy agents wear masks, and finally a toggle for whether the healthy agents are vaccinated or not. There were more toggles planned to be implemented but development fell behind schedule and these toggles were prioritised as they would have the most visible effects on the simulation.

The first step was to make a 3d model of a mask which would be displayed on the agents face if they are set to wearing a mask. This 3d model was made with a 3d modelling tool called blender, then imported to the project, at which point colour was added and it was put onto the head of the agent prefab.

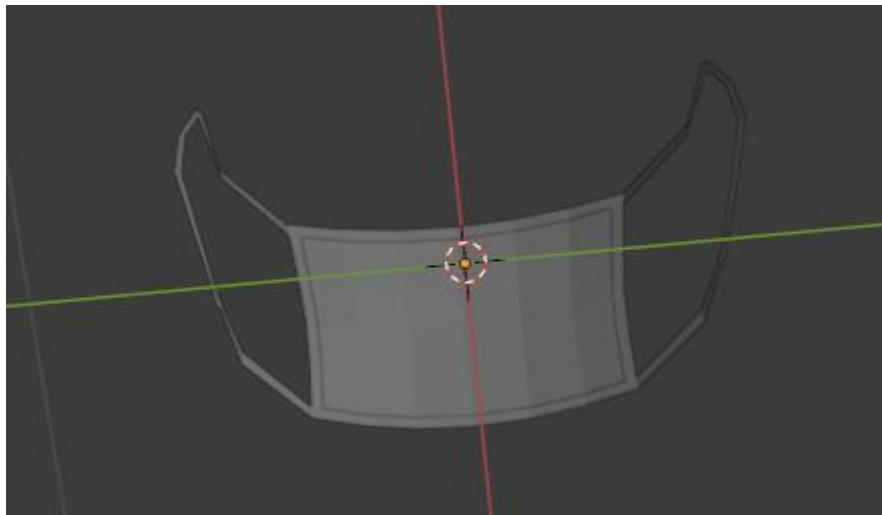


Figure 50 – Blender mask model

A Boolean variable was then added to each agent’s behaviour script which is changed based on the toggles. If the agent is tagged as infectious, it checks the “infectious agent wearing mask” toggle and sets the Boolean to that value. Otherwise, it checks the “healthy agent wearing mask” toggle and

sets the Boolean to that value. It then checks the value of the Boolean and sets the mask to active if it's true or inactive if it's false.

Due to masks highly reducing the number of particles emitted in a real situation, a second particle system was added to the infectious agent that is disabled by default. This particle system emits particles significantly slower that also travel a much smaller distance. The number of particles emitted is also cut down by 77%, which is a number taken from a WHO backed study on masks which was mentioned in the literature review.

There is then another if statement which checks if the agent is tagged as infectious and if their mask boolean is set to true. If this if succeeds, it disables the normal particle system and enables this variant particle system.

Then for vaccines, a similar concept was implemented by importing a PNG of a vaccine and adding it to a canvas above the prefab agent's head which was disabled by default. To ensure this PNG always faces the player, a script was written and attached to the canvas that was constantly changing the rotation to face the direction of the active camera.

```
void Start()
{
    cam = GameObject.FindWithTag("MainCamera").transform;
}

// Update is called once per frame
0 references
void Update() You, 4 days ago • Added a vaccine png display
{
    Vector3 relativePos;//For holding point to look

    relativePos = cam.transform.position - transform.position;

    //For holding new rotation
    Quaternion toRotation = Quaternion.LookRotation(relativePos, cam.transform.up);
    //Lerps the vaccine icon to face the player
    transform.rotation = Quaternion.Lerp( transform.rotation, toRotation, lerpSpeed * Time.deltaTime );
}
```

Figure 51 – Face player script



Figure 52 – Vaccine indicator

Then the same logic was used, getting the value of the vaccine toggle, and enabling the vaccine indicator if the toggle is set to true and the agent is tagged as healthy.

Finally, some code was added to the particle collision script to make these toggles have an effect on the chances of infection. This started off by having 2 variables, maskPreventionChance and vaccinePreventionChance, which are both set to negative values by default. Then, if the collided agent is wearing a mask, the maskPreventionChance is set to 0.77, representing a 77% chance of prevention which was taken from the same WHO backed study on masks mentioned earlier. If the collided agent is vaccinated, the vaccinePreventionChance is set to 0.95, representing a 95% chance of prevention which was taken from the Pfizer website, as mentioned in the literature review. The calculation is then made, generating a random value between 0 and 1 for each calculation and checking if the random values are greater than the prevention chances.

```
//Gets the agents behaviour to see if they have mask/vaccination
PersonBehaviours collidedPersonBehaviours = other.GetComponent<PersonBehaviours>();
//2 second collision cooldown to prevent stats being irrelevant due to multiple collisions
if(collidedPersonBehaviours.lastHit < Time.time - 2)
{
    //Affect chance based on mask/vaccination
    if(collidedPersonBehaviours.wearingMask == true)
    {
        maskPreventionChance = 0.77f;
    }
    (field) bool PersonBehaviours.isVaccinated
    if(collidedPersonBehaviours.isVaccinated == true)
    {
        vaccinePreventionChance = 0.95f;
    }
    //Calculate chance of infection
    if(Random.value > maskPreventionChance && Random.value > vaccinePreventionChance)
    {
        //Changing all materials to exposed for visual confirmation
        SkinnedMeshRenderer[] newMeshRenderers = other.GetComponentsInChildren<SkinnedMeshRenderer>();
        foreach(var m in newMeshRenderers)
        {
            m.material = Exposed;
        }
        other.gameObject.tag = "Exposed";//For exposed counter
    }
}
```

Figure 53 – Infection chances

If the agent were neither wearing a mask nor vaccinated, both prevention chances would still be a negative value, so the random values have a 100 % chance of being greater than them and infecting the agent.

If the agent were only wearing a mask, the vaccination prevention chance would be irrelevant as it would still be negative, so the chance of infection would be 23 % as there is a 23/100 chance the random value will be above 77.

If the agent were only vaccinated, the mask prevention chance would be irrelevant as it would still be negative, so the chance of infection would be 5 % as there is a 5/100 chance the random value will be above 95.

If the agent were wearing a mask and was vaccinated, both checks would take place. There would be a 23% chance it succeeds the first check and a 5% chance it succeeds the second check. It needs to succeed both to go forward ( $.23 * .005$ ), so the final chance of infection would only be 1.15%

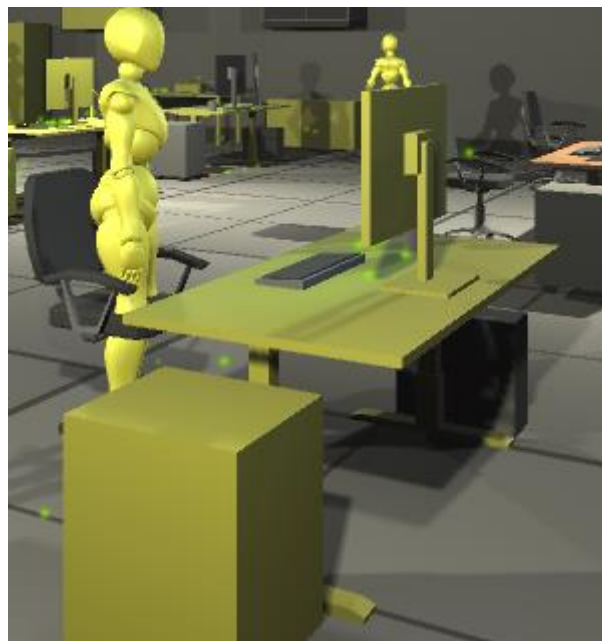
### 4.13. Contaminated Surfaces and Agents

Now that the infection model has been expanded upon, the mechanic of agents getting infected from surfaces was implemented. This was done by first creating an empty game object with a particle system attached and saving it as a prefab. This particle system only spawns a few particles in a small area, with a small amount of noise. The particle collision script was then modified so when a particle collided with an object or surface, this particle system game object was instantiated on the collided object.

```
//Adding particles to represent a contaminated surface  
GameObject particleObject = Instantiate(particleSystemContaminated, other.transform.position, other.transform.rotation, other.transform);
```

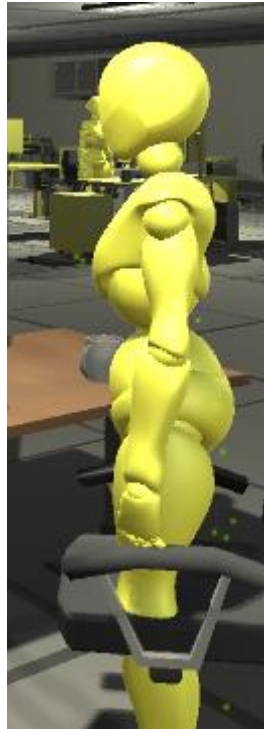
*Figure 54 – Adding contamination particles*

For agents to become exposed from these particles, a new script was added to this particle system game object prefab. This script is practically the same as the particle collision script, although this new script only registers the collisions with agents, so if the particles collide with any other objects, the infection will not be spread. This was decided as it would not make sense for a contaminated surface to be able to contaminate other surfaces.



*Figure 55 – Contaminated surface with particles*

Finally, another empty game object was created with a particle system attached and saved as a prefab. This particle system also only spawns a small number of particles, but they emit in a shape similar to the shape of the character model. This is to visually demonstrate that the agent's clothes are contaminated with viral particles. Both collision scripts were modified so when an agent becomes exposed, this particle system game object is instantiated on them. The same modified collision script was added to this particle system game object prefab, so exposed agents have a small chance of exposing other agents if they touch off each other, but they will not contaminate surfaces.



*Figure 56 – Exposed agent with particles on their body*

The infection can now spread from breathing in infectious particles, interacting with a contaminated surface, or touching off another exposed agent. All of these possible transmission events take into account whether the agent is wearing a mask and/or is vaccinated.

#### 4.14. Finishing UI

Now that all of the core mechanics were implemented, the start screen, end screen and other UI elements were greatly improved, and some bugs were fixed. Some black box testing was carried out which helped indicate problems with the UI and simulation.

The start screen was aligned more symmetrically, a short explanation of the system was added and a panel was added which informs the user that they will be able to adjust the timescale, in the same place that the new timescale panel is. A bug was noted that the player could walk while the start screen is being displayed, so player movement is set to disabled until the start button is pressed.

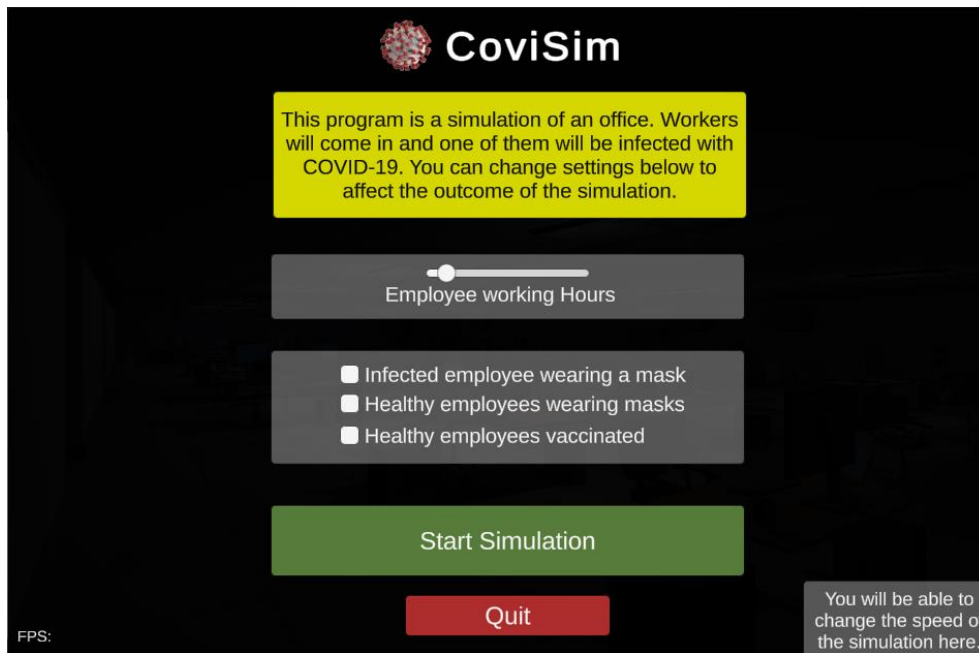


Figure 57- Final start screen

Various UI elements were resized, and a panel for the timescale was added in the corner. This panel also displayed the elapsed time and the time left until the working hours' end. An FPS counter was also added, as during the initial black-box testing the build was run on a very low-end laptop and did not seem to perform as well as other devices. Finally, an "Escape to restart" panel and behaviour were added to allow users to quickly go back to the start screen, as a black box tester noted that in order to restart that had to wait for the simulation to end or close the program.



Figure 58 – Final UI elements

The elapsed time and time left counters were added by displaying the time passed since the user clicked the start button and displaying the difference between the working hours and the above value. The time left value was clamped above 0 to avoid displaying negative numbers at the end of the simulation.

```
//Display the time passed and time left of the workers shift
ElapsedTime.text = "Elapsed Time: " + Mathf.Round(Time.time - startTimeManager.startTime);
TimeLeft.text = "Shift Ends: " + Mathf.Clamp(Mathf.Round(WorkingHours.value - (Time.time - startTimeManager.startTime)), 0, 10000);
```

Figure 59 – Time displays

The FPS counter was implemented by displaying the value of 1 divided by unscaledDeltaTime.

```
//Updating the FPS display
float fps = 1 / Time.unscaledDeltaTime;
fpsDisplay.text = "FPS: " + Mathf.Round(fps);
```

Figure 60 – FPS display

Finally, many counters were added to the end screen, with buttons to restart or quit the simulation.

The values of the 4 interactable components on the start screen are displayed by simply checking what their values were set to. Then the item counts were implemented by searching for all of the items with their respective tags in the scene and displaying the number.

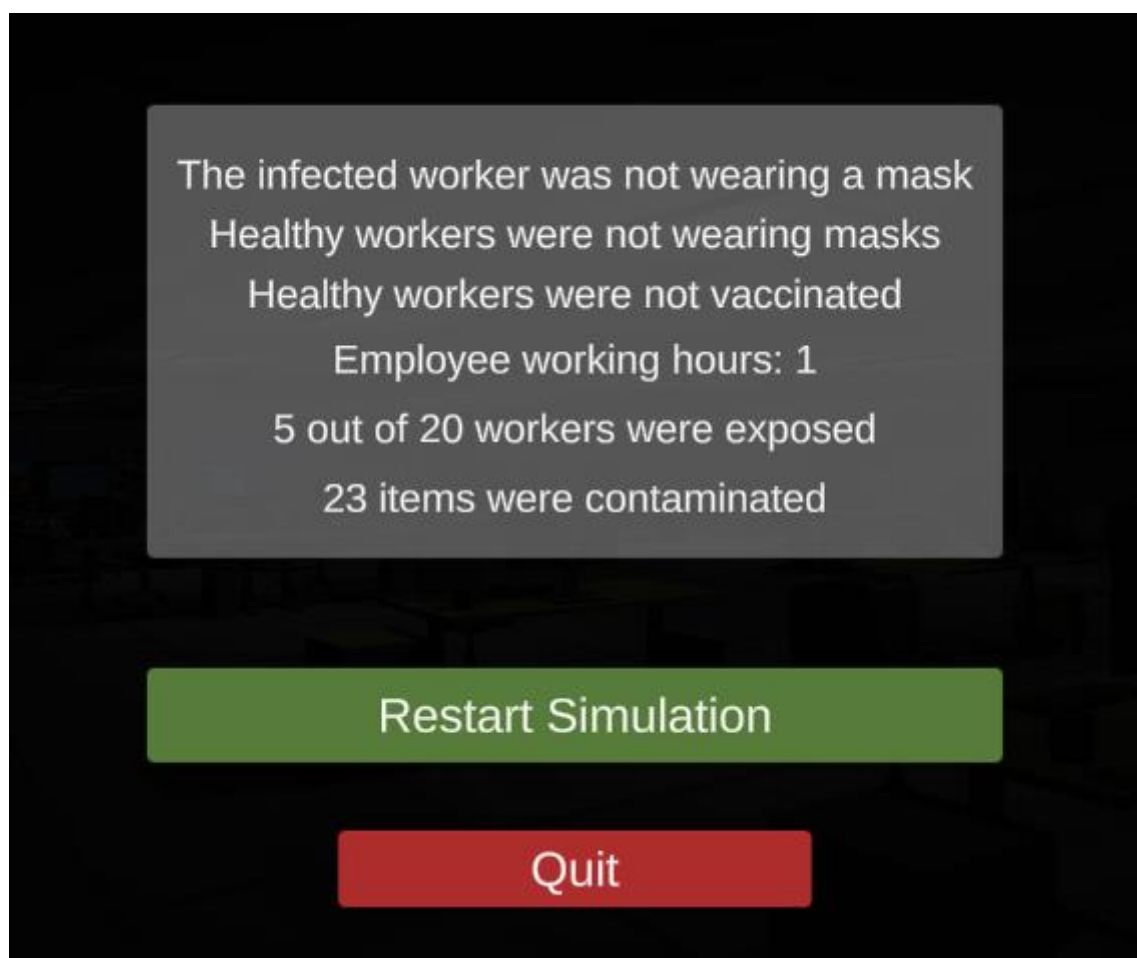


Figure 61 – End screen

#### 4.15. Conclusions

In this chapter, the development process was outlined and discussed. The development went quite well, with the majority of the key features that were planned to be implemented ending up being successfully implemented in some way. Every section of development provided its own set of challenges, but in the end, they were all great learning experiences.

There were more features planned to be added to the simulation, as well as known bugs that were planned to be fixed, but there was not enough time so they will be mentioned in the evaluation and future work chapters alongside the feedback received from the black box and white box testing.



## 5. Testing and Evaluation

### 5.1. Introduction

This chapter outlines the testing the system went through during and after the development process, and how corrections were made, and bugs were spotted. An evaluation of the project is then discussed, evaluating the system, and looking at results from the simulation itself.

### 5.2. System Testing

#### 5.2.1 Testing During Development

During the process of development, the implementation of each feature was followed with various self-tests to ensure the simulation was run as it was supposed to. Some black box testing was also intermittently done with third parties, by making a build of the project and sending it on to other students. This process was useful for finding overlooked solutions and fixing any bugs that weren't easy to spot. At a few points in development, the project was also explained to and evaluated by Dr Zach Tan, who gave his opinion on what features should be prioritised and how to display the methods of transmission.

#### 5.2.2 Testing After Development

Once development was finished, the project was published to <https://kyleheffernan.itch.io/covisim>

Itch.io is a platform that lets independent creators upload their projects and other users can download and run them. A google form with some questions about the simulation was created and added to this page. Other Itch.io users would see this in their feed and possibly be drawn to downloading and testing the project. This link was then also sent to numerous Discord servers focused on game development, college projects, Unity, Irish developers etc. Quite a sizeable amount of people downloaded the simulation and filled out the form.

Question 1 to ascertain if the user could download the project and run the simulation without problems. All but one of the answers were yes, with the outlier stating, "I have an old computer so it ran ok". The project seemed to be able to run completely fine and smoothly on most systems, so the overall performance and optimisation of the system were deemed good. The only issue being low-end systems, which could have done with better optimisation.

Were you able to download and run the simulation?

13 responses

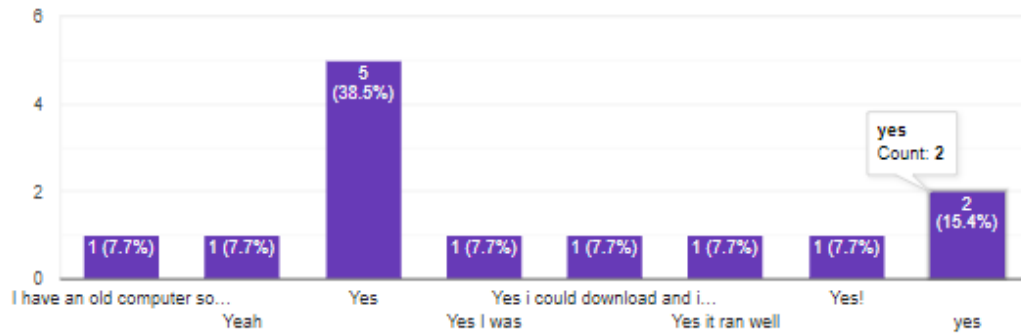


Figure 62 – Form Q1

Question 2 to get the users overall opinion of the project. The responses to this question were all very positive.

Please rate the simulation

13 responses

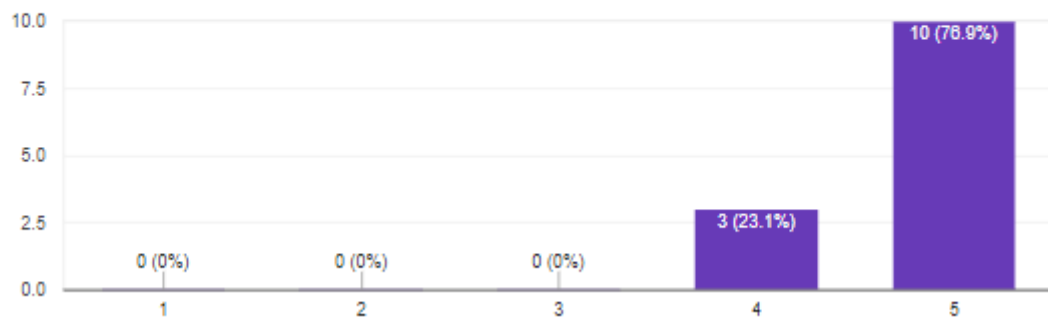


Figure 63 – Form Q2

Question 3 to ascertain whether it was clear what was happening in the simulation, as the simulation aims to make the method of transmission clear and easy to understand. The responses were again all very positive.

Was it clear what was happening?



13 responses

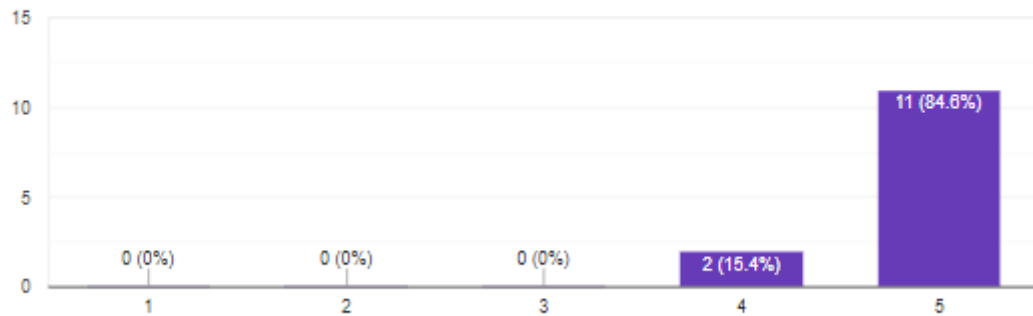


Figure 64 – Form Q3

Question 4 asked about suggestions and features to add, and there were quite a few differing answers. One response suggested adding handwashing, which was one of the lower priority features that was planned to be added but cut due to time and could be taken up in future work. Many of the answers focused on adding more customisation to the start screen, such as the amount of healthy and infectious agents and more office layouts. Both of these were also planned to be implemented but were not due to time restrictions. Users also suggested more complex behaviours, more realistic models and better movement controls, which all could be undertaken in future work.

Would you have any suggestions for features to add?

13 responses

A list of 13 suggestions for features to add, displayed in a scrollable container. The suggestions are:

- An option for regular hand washing or sanitising?
- More complex behaviours and interactions with the NPC's. Ability to simulate different environments. Simulating big crowds
- More realistic people
- Possibly change the movement controls
- It would be interesting to see how the virus would spread in different buildings/office layouts
- nope
- no
- More custimastion
- More office options to choose from would be nice

Figure 65 – Form Q4

Question 5 asked the users if they experienced any bugs. The only bug found was actually a known bug that was not fixed due to time constraints. This bug occurs if the user changes the timescale but does not click off the slider before they try to move again. This will result in player input moving the slider up and down when it should only be controller their character. This may be able to be fixed by some kind of automatic deselection script, or by changing the movement controls.

Did you find any bugs?

12 responses

No
It ran very smoothly for me
No I did not
no
yes the time slider bar will be triggered by the movement keys if you don't click off it
Nope

Figure 66 – Form Q5

Question 6 asked the user how realistic they found the simulation, and the answers were again generally quite positive.

How realistic did you find the simulation?

13 responses

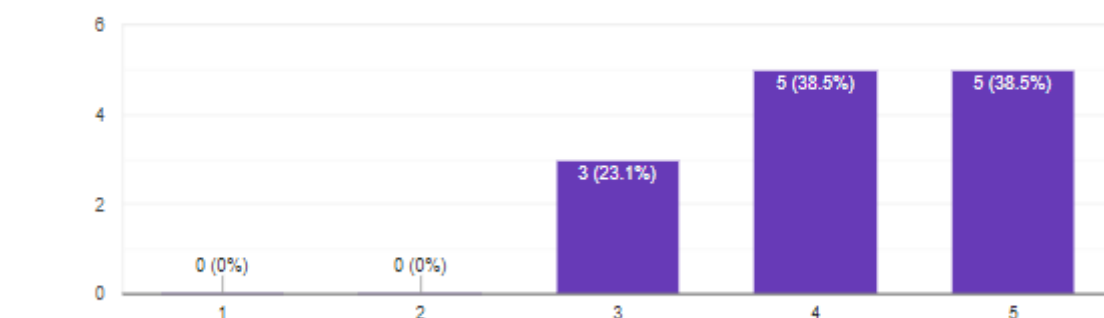
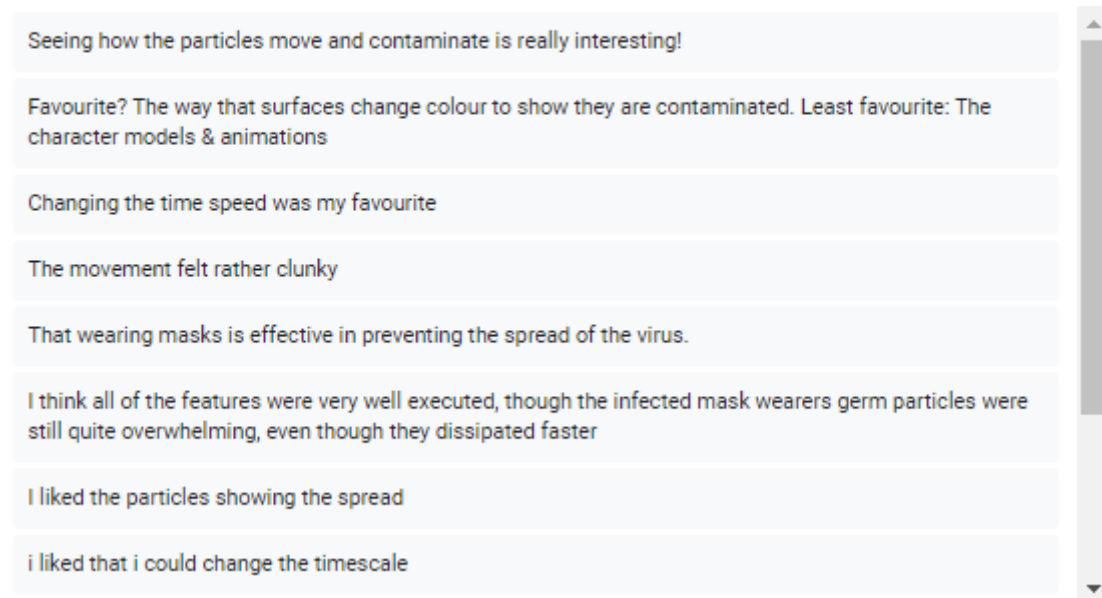


Figure 67 – Form Q6

Question 7 asked about the users favourite and least favourite features. Many users enjoyed being able to change the timescale and liked how surfaces that were contaminated turned a different colour. Many users also liked visually seeing how masks and vaccinations affected the spread of the virus. One user did not like the character models and animations, which are relatively basic and could be improved in future work to add more realism to the simulation. Another user felt that the movement was clunky, which again could be improved in future work for a better user experience.

What was your favourite/least favourite feature?

12 responses



Seeing how the particles move and contaminate is really interesting!
Favourite? The way that surfaces change colour to show they are contaminated. Least favourite: The character models & animations
Changing the time speed was my favourite
The movement felt rather clunky
That wearing masks is effective in preventing the spread of the virus.
I think all of the features were very well executed, though the infected mask wearers germ particles were still quite overwhelming, even though they dissipated faster
I liked the particles showing the spread
i liked that i could change the timescale

Figure 68 – Form Q7

Question 8 and 9 asked if the user ran the simulation multiple times and if they did, whether they noticed differences in results. Other than a single user that only ran the simulation once, every user noticed changes and commented on how the masks and vaccinations lower the infection rate.

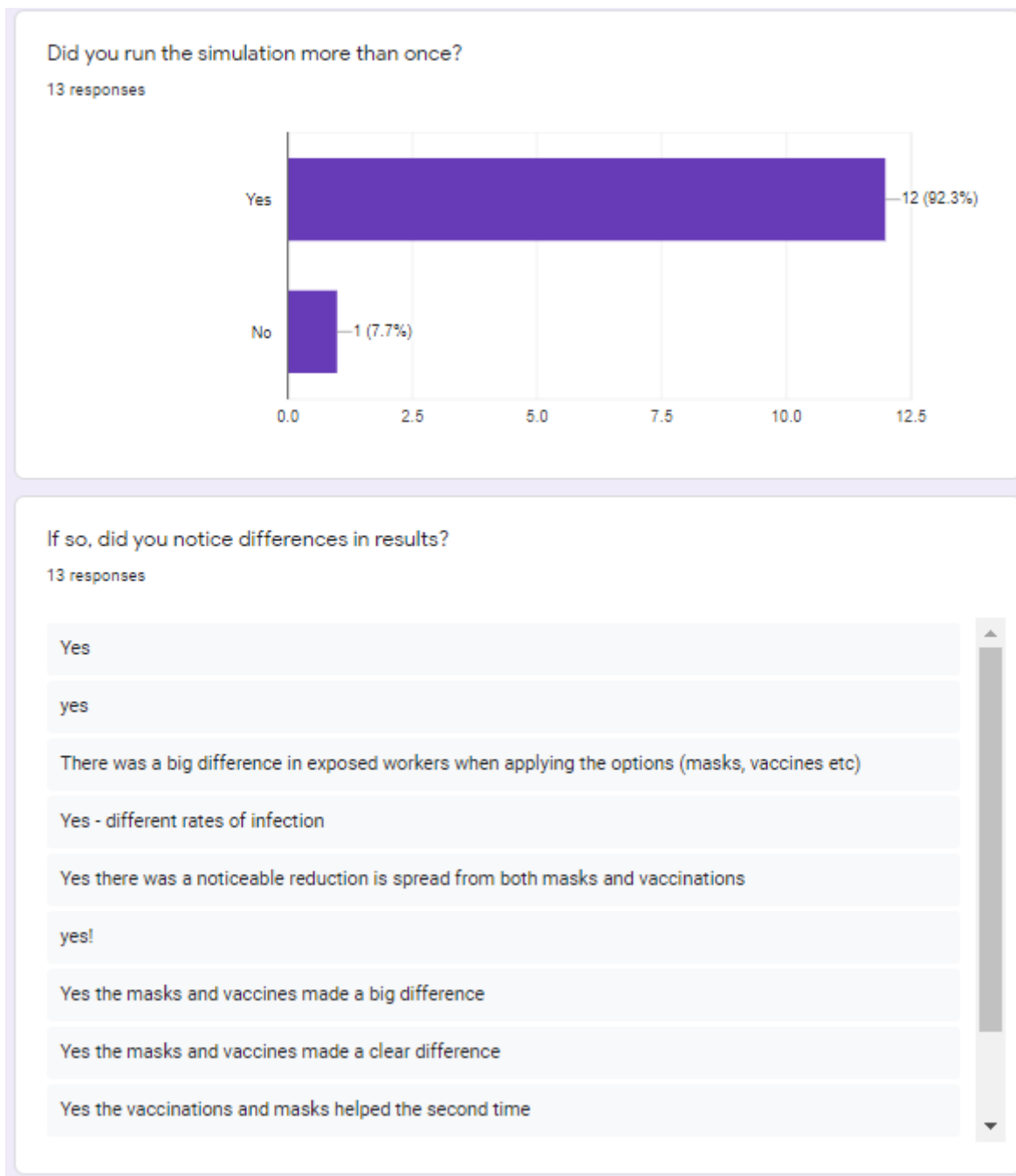


Figure 69 - Form Q8, Q9

Finally, question 10 asked if the user felt they better understood the methods of transmission after using the simulation. The answers to this question were overwhelmingly positive, with most users choosing the maximum value.

Do you feel like you now better understand the methods of transmission?

13 responses

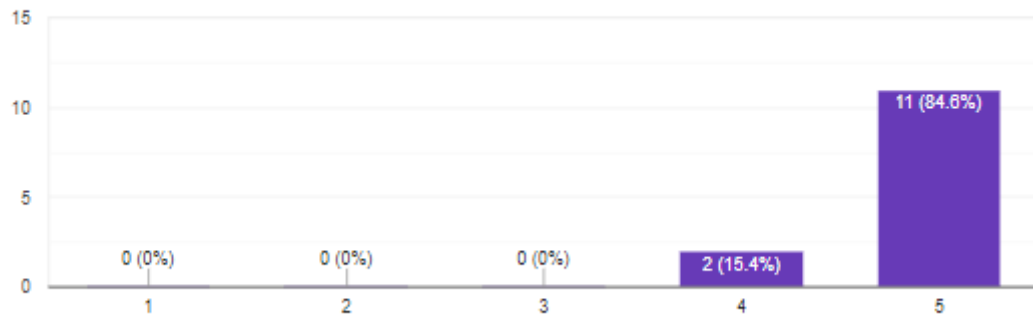


Figure 70 – Form Q10

### 5.3. System Evaluation

#### 5.3.1 Evaluating Simulation Results

For the purpose of system evaluation, the project was run sixty times and the number of exposed agents out of 20 were recorded in rounds of 12. The working hours were set to three for all tests to allow a reasonable amount of to pass.

The first twelve times were run with no settings altered, which is marked as **A**.

The next twelve times with infectious agents wearing masks, marked as **B**.

The next twelve times with healthy agents wearing masks, marked as **C**.

The next twelve times with healthy agents vaccinated, marked as **D**.

Finally, another twelve times with all settings altered, marked as **E**.

	A	B	C	D	E
Round 1	13	7	6	1	0
Round 2	15	1	5	0	0
Round 3	17	1	1	1	0
Round 4	15	4	1	0	0
Round 5	12	0	5	0	0
Round 6	15	5	1	1	0
Round 7	14	6	4	0	1
Round 8	16	7	2	1	0
Round 9	14	12	7	1	0
Round 10	15	4	3	0	0
Round 11	15	2	2	0	0
Round 12	12	3	4	1	0
Average	14.5	4.1	3.4	0.5	0.08

Both masks and vaccines are shown to have a tremendous effect on the transmission rates, and when every agent is both wearing a mask and vaccinated the transmission rate is negligible.

### 5.3.2 Evaluating Whole System

The results from running the simulation match up with the values that were expected, similar to values found in medical papers and similar existing systems. These expectations were that both masks and vaccines have a clear effect on transmission.

The overall user feedback from the testing was also overwhelmingly positive, with the majority of users saying the simulation was realistic, easy to understand, and helped them better understand the methods of transmission.

## 5.4. Conclusions

In this chapter, the process of testing the simulation during and after development was outlined and the feedback received from users through the google form was discussed. An evaluation of the project is then discussed, evaluating the system, and looking at results from the simulation itself.



## 6. Conclusions and Future Work

### 6.1. Introduction

In this chapter, a discussion of the future of CoviSim is presented, with features that could be implemented, sections that could be expanded upon and bugs that could be fixed. The gant chart of the development cycle is then shown and followed by some overall reflections on the project.

### 6.2. Future Work

In the case of future development of CoviSim, there are many features that could be added, which have been gathered from user feedback as well as some features that were planned but cut out due to not being viable within the development time of this project.

A feature that was originally planned for this project but cut is the option to let the user choose from a list of office layouts at the start of the simulation. This would have added another level of complexity and user intractability and could be implemented by having multiple scenes set up and having a script that loads the scene selected by the user.

Another feature similar to this is to let the user alter the amount of healthy and infectious agents that enter the scene. This was also planned but left out of the project due to time constraints but would greatly improve user intractability.

More agent variables such as handwashing behaviours and age were also planned for the system and were planned to have an effect on the infection model, but these features were cut as they would not have an effect on the results as much as masks and vaccines would, so they were set as a lower priority and ended up not being implemented. These features would add much more complexity to the infection model and could use numbers from medical studies in their calculations.

To improve the realism of the system, the behaviour trees could be greatly expanded upon, adding many more behaviours and interactions such as team meetings or toilets. On top of this, more realistic human models and animations could be designed and implemented.

The movement behaviour of the player could also be updated and improved to a more intuitive and interactable behaviour, as some users felt it was a bit clunky.

A bug in which the timescale slider value is changed by keyboard controls if the user forgets to click off of it would be fixed in future project work, possibly by changing the movement controls or by writing a script for automatically deselecting the slider.

Finally, porting the project over to or remaking it in the entity component system could be undertaken as future work as if it is done right, it could greatly optimize performance giving the option to have an extremely large number of agents in a much bigger scene, although this may be a difficult challenge as the entity component system is still in beta.

## 6.3. Gant Chart

Below, a gant chart is presented, which contains all of the stages of development, most of which were completed with a few features cut due to time restrictions.

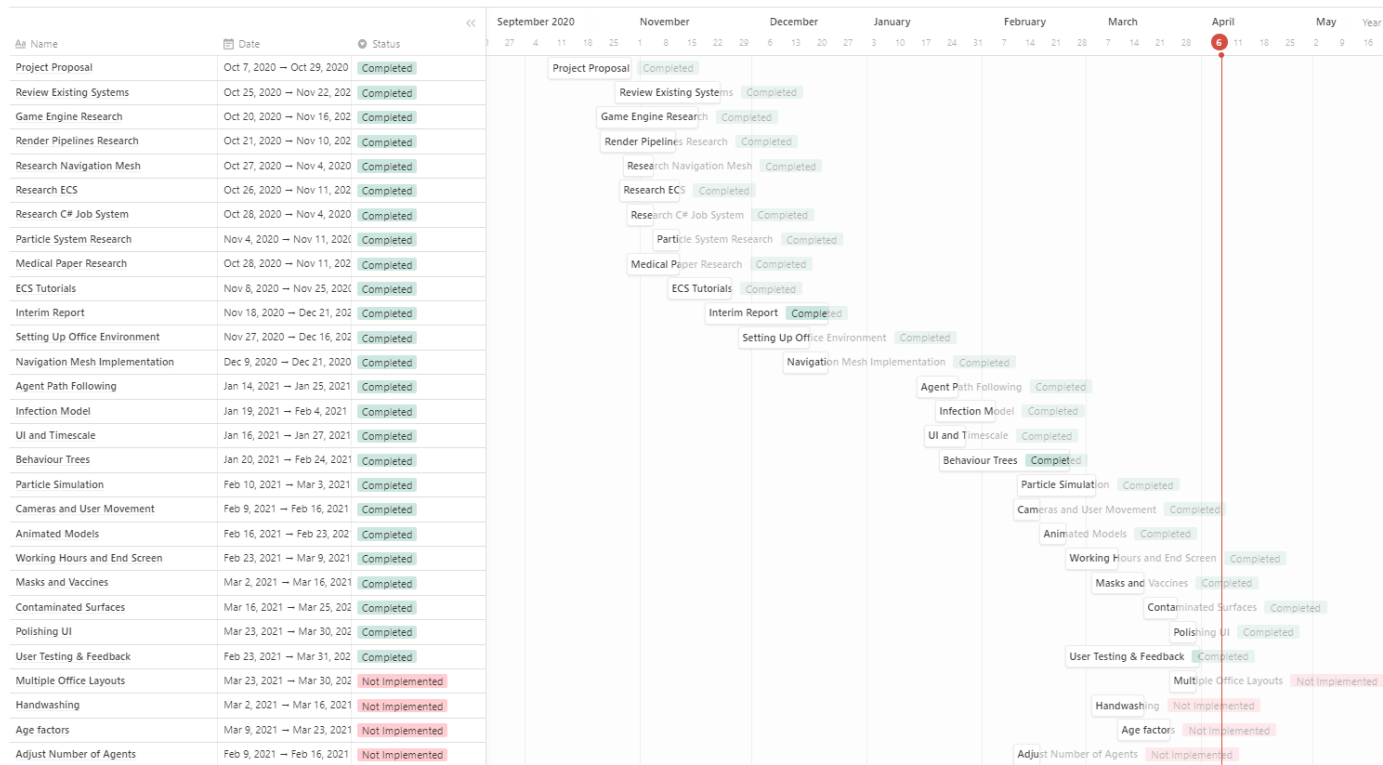


Figure 71 – Gant chart

## 6.4. Project Conclusions

### 6.4.1 Literature review

During the literature review chapter, similar existing systems were investigated and considered, and many relevant technologies were reviewed and discussed. Finally, similar fourth-year projects from the last few years were reviewed. This chapter helped with deciding on which technologies to use for the system, as well as giving an idea of the system requirements. The main topics learned from this chapter were the capabilities of the Unity game engine and the knowledge that some of the most important items that could be implemented are masks and vaccines as they had some of the greatest effects on transmission rates.

### 6.4.2 Experiment Design

In the experiment design chapter, the methodology used during development was discussed, and an overview of the project was presented. The front and back end of the project was then described with the use of key screens and diagrams. The main points learned from this chapter were the pros and pros of the agile scrum methodology, good ways to layout the user interface in a 3d simulation, and finally the structure and planning of behaviour trees.

### 6.4.3 Experiment Development

During this chapter, the entire development process was outlined and the implementation of each feature was discussed, with the aid of many diagrams and code snippets. During the development process, an extremely large amount of invaluable Unity knowledge was gained, and many coding and program design skills were improved upon, especially during the implementation of behaviour trees. A lot of knowledge was also gathered about the entity component system which will be critical in the development of future projects involving ECS.

### 6.4.4 Evaluation

Finally, in the evaluation chapter, the system testing during and after development was described, and the responses to the google form questions were laid out and considered. An evaluation was then done on the simulation to analyse and compare the results. Both the google forms and system evaluation showed that the project successfully demonstrated the methods of transmission of COVID-19, also helping users develop a better understanding of factors that can affect transmission rates.

#### 6.4.4 Final Reflections

Overall, this project was researched, designed and developed successfully as it met its initial aim of helping users understand the transmission methods of COVID-19 and how certain variables can affect them.

While the project development started off slow due to the decision to abandon ECS, overall, the development process was a great and invaluable learning experience that helped greatly develop many project management skills. Many of the concepts involved with this project, such as covid transmission methods and behaviour trees proved to be extremely interesting topics to learn and develop in.

The development of CoviSim was an enjoyable experience, with many challenges along the way that forced certain implementation plans to be thought of in different interesting ways.

## Bibliography

1.  
Fort J. Exploring new ways to simulate the coronavirus spreadUnity Simulation을 이용한  
코로나바이러스 확산 시뮬레이션 コロナウイルスの拡散をシミュレーションする新し  
い方法を探る - Unity Technologies Blog [Internet]. Unity Technologies Blog. 2020 [cited  
2020 Dec 13]. Available from: <https://blogs.unity3d.com/2020/05/08/exploring-new-ways-to-simulate-the-coronavirus-spread/>
2.  
Evershed N, Ball A. See how coronavirus can spread through a population, and how  
countries flatten the curve [Internet]. the Guardian. The Guardian; 2018 [cited 2020 Dec 14].  
Available from: <https://www.theguardian.com/world/datablog/ng-interactive/2020/apr/22/see-how-coronavirus-can-spread-through-a-population-and-how-countries-flatten-the-curve>
3.  
About — Godot Engine (stable) documentation in English [Internet]. Godotengine.org. 2020  
[cited 2020 Dec 14]. Available from: <https://docs.godotengine.org/en/stable/about/index.html>
4.  
Unity Essentials - Unity Learn [Internet]. Unity Learn. Unity Learn; 2020 [cited 2020 Dec  
15]. Available from: <https://learn.unity.com/pathway/unity-essentials>
5.  
Unity Technologies. Unity - Manual: Render pipelines [Internet]. Unity3d.com. 2019 [cited  
2020 Dec 15]. Available from: <https://docs.unity3d.com/Manual/render-pipelines.html>
6.  
Unity Technologies. Unity - Manual: Building a NavMesh [Internet]. Unity3d.com. 2019  
[cited 2020 Dec 15]. Available from: <https://docs.unity3d.com/Manual/nav-BuildingNavMesh.html>
7.  
Unity Technologies. Unity - Scripting API: ParticleSystem [Internet]. Unity3d.com. 2019  
[cited 2020 Dec 15]. Available from:  
<https://docs.unity3d.com/ScriptReference/ParticleSystem.html>
8.  
Unity 2017 Game AI programming - Third Edition | Packt [Internet]. Packt. 2017 [cited 2021  
Apr 4]. Available from: <https://www.packtpub.com/product/unity-2017-game-ai-programming-third-edition/9781788477901>
9.  
Entity Component System | Entities | 0.16.0-preview.21 [Internet]. Unity3d.com. 2020 [cited  
2020 Dec 15]. Available from:  
<https://docs.unity3d.com/Packages/com.unity.entities@0.16/manual/index.html>
10.  
BillWagner. A Tour of C# - C# Guide [Internet]. Microsoft.com. 2020 [cited 2020 Dec 15].  
Available from: <https://docs.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/>
11.  
Unity Technologies. Unity - Manual: C# Job System Overview [Internet]. Unity3d.com.  
2018 [cited 2020 Dec 15]. Available from:  
<https://docs.unity3d.com/2018.4/Documentation/Manual/JobSystemOverview.html>
- 12.

- Tuite AR, Fisman DN, Greer AL. Mathematical modelling of COVID-19 transmission and mitigation strategies in the population of Ontario, Canada. Canadian Medical Association Journal [Internet]. 2020 Apr 8 [cited 2020 Dec 15];192(19):E497–505. Available from: <https://www.cmaj.ca/content/192/19/E497.full>
- 13.
- Bourouiba L. A Sneeze. New England Journal of Medicine [Internet]. 2016 Aug 25 [cited 2020 Dec 15];375(8):e15. Available from: <https://www.nejm.org/doi/full/10.1056/NEJMicm1501197>
- 14.
- Smith C. WHO-backed study shows face masks can reduce coronavirus spread significantly [Internet]. BGR. BGR; 2020 [cited 2021 Apr 5]. Available from: <https://bgr.com/2020/06/03/coronavirus-face-masks-reduce-risk-of-transmission-infection-study-says/>
- 15.
- Horne M. Coronavirus in Scotland: Vulnerable will receive vitamin D supplements [Internet]. Thetimes.co.uk. The Times; 2020 [cited 2020 Dec 15]. Available from: <https://www.thetimes.co.uk/article/coronavirus-in-scotland-vulnerable-will-receive-vitamin-d-supplements-zc8stdpkh>
- 16.
- Pfizer and BioNTech Conclude Phase 3 Study of COVID-19 Vaccine Candidate, Meeting All Primary Efficacy Endpoints | pfpfizeruscom [Internet]. Pfizer.com. 2020 [cited 2021 Apr 5]. Available from: <https://www.pfizer.com/news/press-release/press-release-detail/pfizer-and-biontech-conclude-phase-3-study-covid-19-vaccine>
- 17.
- Nishiura H, Oshitani H, Kobayashi T, Saito T, Sunagawa T, Matsui T, et al. Closed environments facilitate secondary transmission of coronavirus disease 2019 (COVID-19). 2020 Mar 3 [cited 2020 Dec 15]; Available from: <https://www.medrxiv.org/content/10.1101/2020.02.28.20029272v2>
- 18.
- Few S. Data Visualization - Past, Present, and Future [Internet]. 2007. Available from: [https://www.perceptualedge.com/articles/Whitepapers/Data\\_Visualization.pdf](https://www.perceptualedge.com/articles/Whitepapers/Data_Visualization.pdf)
- 19.
- Kiryakova G, Angelova N, Yordanova L. Gamification in education. Proceedings of 9th International Balkan Education and Science Conference.
- 20.
- Snaps Prototype | Office [Internet]. @UnityAssetStore. Unity Asset Store; 2019 [cited 2021 Apr 5]. Available from: <https://assetstore.unity.com/packages/3d/environments/snaps-prototype-office-137490>
- 21.
- Mixamo [Internet]. Mixamo.com. 2021 [cited 2021 Apr 5]. Available from: <https://www.mixamo.com/#/>
- 22.
- Principles behind the Agile Manifesto [Internet]. Agilemanifesto.org. 2020 [cited 2020 Dec 16]. Available from: <https://agilemanifesto.org/principles.html>
- 23.
- Atlassian. What is Scrum? [Internet]. Atlassian. 2018 [cited 2020 Dec 16]. Available from: <https://www.atlassian.com/agile/scrum>