School of Computing

FACULTY OF ENGINEERING AND PHYSICAL SCIENCES

UNIVERSITY OF LEEDS

Final Report

Unity Pandemic Simulation

Luca Smith González

Submitted in accordance with the requirements for the degree of Meng Computer Science

2022/23

COMP3931 Individual Project

The candidate confirms that the following have been submitted*:*

Items	**Format**	**Recipient(s) and Date**
Final Report	*PDF file*	*Uploaded to Minerva (2/5/23)*
Scanned participant consent forms	*PDF file / file archive*	*Uploaded to Minerva (2/5/23)*
Link to online code repository	*URL*	*Sent to supervisor and assessor (2/5/23)*
User manuals	*PDF*	*Sent to client and supervisor (2/5/23)*

[illegible]

Summary:

The task was to create a simulation of a pandemic spread in the game development platform Unity, simulating the spread of disease and human behaviour.

Pandemic Simulation

Chapter 1, project introduction and background

Section 1.1, introduction: The following report details the initial goals, and subsequent process, of creating a pandemic simulation in Unity. The simulation itself is a city-wide infection model, where the user can input different values and experiment to see what parameters change the simulation in what way.

Section 1.2, outline, and goals: The chosen approach to implement the simulation prioritises usability over accuracy, with the intention of creating an application capable of illustrating a pandemic, rather than accurately predicting behaviour. The application described would be suitable in an educational context, but would not be suitable for proper modelling.

Therefore, the project's priorities are as follows, in descending order:

- Efficiency
- Usability
- Simplicity
- Consistency
- Realism
- Aesthetics

To this end, the infection model is based on the SIR mathematical model, created by W. O Kermack and A.G McKendrick. It is a compartmental approach, with 3 classes for a simulated population:

1. **S(t), susceptible:** Used to represent number of individuals not yet infected at time **t**.
2. **I(t), infected:** Used to represent the number of infected individuals at time **t**, who are capable of spreading the infection.
3. **R(t), recovered:** Used to represent the number of individuals who have been exposed and have recovered from the infection, and are therefore incapable of spreading it further.

The model used in the project is a slightly modified version of the SIR model, with an additional compartment: **E(t), exposed.** This compartment is used to represent the number of individuals who have been exposed to the infection but are still unable to spread it. The full hierarchy of the model implemented is as follows:

$$S(t) \rightarrow E(t) \rightarrow I(t) \rightarrow R(t)$$

This model was chosen for its simplicity, as well as the fact that it is easy to visually represent. This makes it ideal for the simulation's purposes.

In terms of scale, the project initially aimed to simulate a small enclosed space, such as a store, but gradually increased in scope, settling on simulating a small "city" with thousands of individual actors. As such, efficiency and consistency are integral. The aforementioned goals are discussed further in their own sections.

Chapter 1

Section 2, background, and research

Chapter 2, method

As mentioned previously, the pandemic simulation was built with primarily simplicity, usability, and efficiency in mind. There were many iterations of the project, with massive changes in the simulated human behaviour and infection model. These will be discussed in greater detail in the efficiency section of the report.

Project iterations are briefly detailed below. Mentioned components will be discussed in section 2.6.

Section 2.1, first iteration: The first somewhat functional iteration of the project made use of Unity's "onCollisionEnter()" method. The idea was very simple; each human had an attached collider and "listened" for any collisions that occurred. The humans wandered aimlessly, constantly moving towards a randomly assigned area of the map. Once they reached their destination, another random location would be chosen, and they would start moving towards it.

If an infected human collided with another object, it would first check whether said object was a human. If so, the infected human would then check the other human was susceptible, and would spread the infection if it was. All the supervisor did in this iteration was create a set number of humans on startup, and infect one of them.

This implementation was as crude as it was inefficient, with a maximum load of about 200 humans before it started slowing down noticeably.

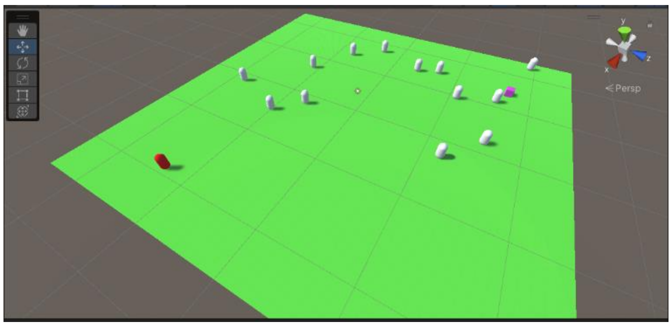


Fig. 1: The first iteration of the project. The capsules represent humans, the red one being an infected individual, and the white capsules being susceptible.

Section 2.2, second iteration: The second iteration was similar to the first, but it implemented the exposed condition. When an infectious human spread the disease to a susceptible human, said human would not become infectious right away. Rather, it becomes "exposed" for a few seconds. Said number of seconds is dictated by a predefined variable in the supervisor (incubation_period), which each human asks for on startup.

When a human initially becomes exposed, it records the time it happened (by asking the supervisor what the current time is) and calculates the moment in time it should become infectious by adding the "incubation period" of the disease to it.

Pathfinding was minimally changed; a method was added to the supervisor which would return a random location on the map. This saved the humans from having to calculate it themselves.

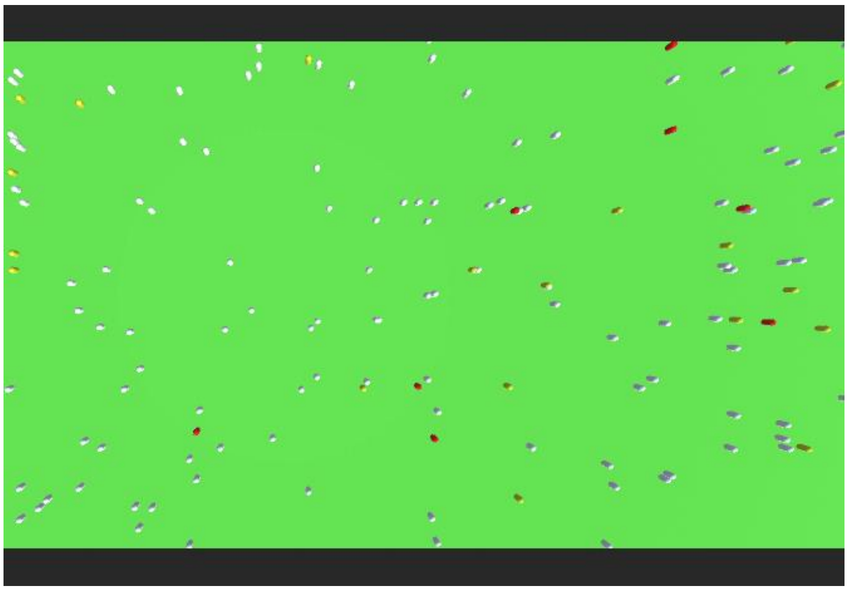


Fig.2: The addition of the exposed condition (yellow capsules)

Section 2.3, third iteration: A graph was added in order to keep track of how many healthy, exposed and infectious individuals there was in the simulation. The supervisor began keeping track of how many humans were in each category, and updating the graph accordingly.

A method called "addGraphValues" was added to supervisor. Each time a human changed its status it would call this method to add and subtract a certain value from the supervisor's totals. For instance, if a human became exposed, it would call this method to subtract 1 from the total amount of healthy people and add 1 to the total amount of exposed people. The supervisor would then update the graph accordingly.

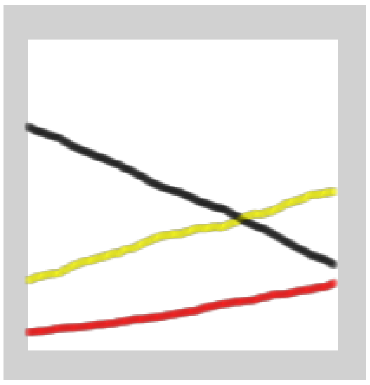


Fig. 3: An example of the graph

Section 2.4, fourth iteration: A pathfinding model was implemented by using "crossroads." Crossroads are trigger zones placed in different locations on startup which tell any humans that step on them where to go. It takes advantage of the fact that the city is laid out in a grid. Each node stores which directions a human is able to move in and relays this information to any human that triggers it.

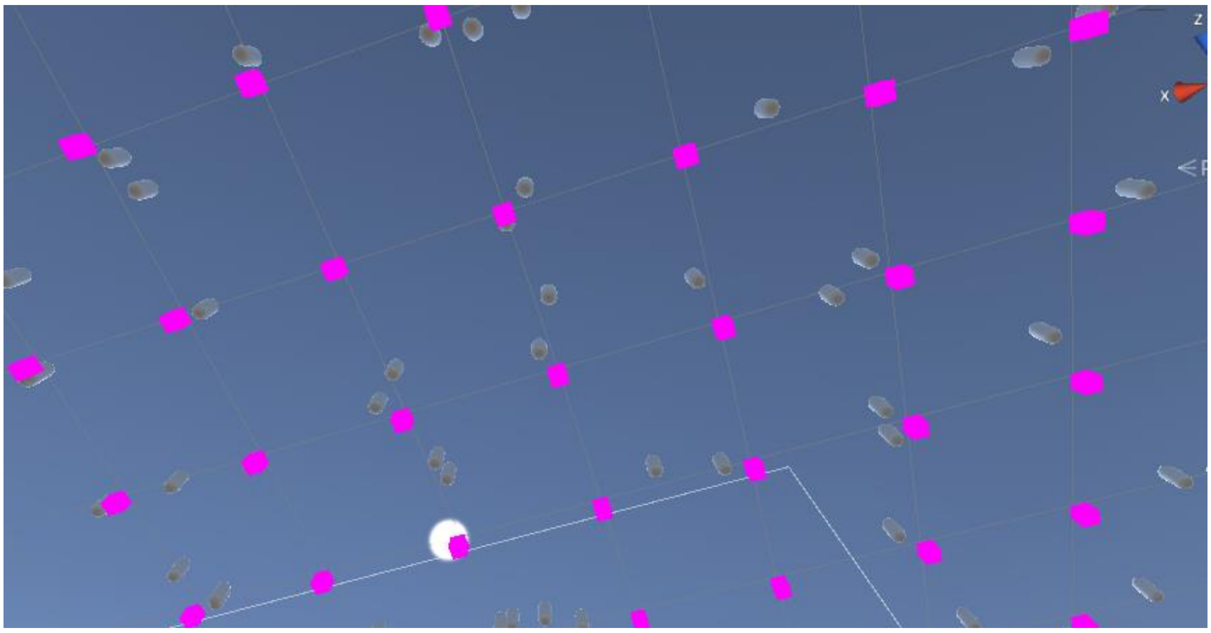
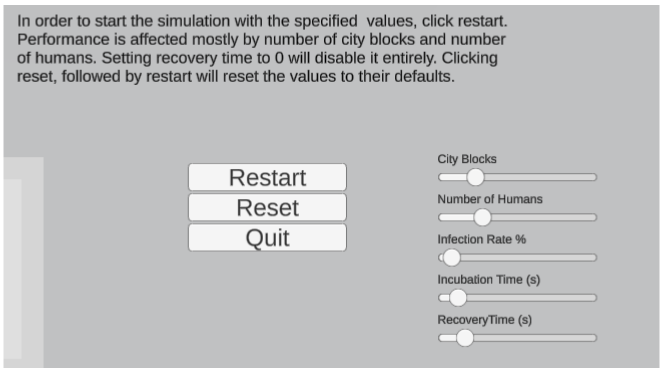


Fig. 4: Nodes are laid out in a grid, forming branching paths that humans can walk down.

Section 2.5, final iteration: In order to create an efficient way of spreading the infection, each crossroad was given the additional functionality of spreading the disease. Every time an infected human walks over a crossroad, there is a small chance (for a very small window of time) that the crossroad infects any subsequent humans that walk on it. Additionally, a menu was implemented, as well as camera control. A user can use this menu to modify data concerning the infection and see how it affects the spread.



In order to start the simulation with the specified values, click restart. Performance is affected mostly by number of city blocks and number of humans. Setting recovery time to 0 will disable it entirely. Clicking reset, followed by restart will reset the values to their defaults.

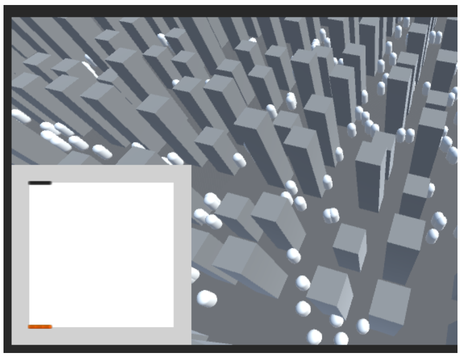


Fig. 5 (left): The menu implemented, allowing the user to modify values in the Supervisor. Fig. 6 (right): an example of the finalised simulation

Section 2.6, project components:

Human: Each human in the simulation has a status attached to it. This status can be any one of, healthy, exposed, infectious and immune. Other objects are able to read this status, and change it if necessary. Each human walks in a specified direction, stored in memory. A human can only walk in one of four directions: North, East, South, West. The direction it walks in can be changed if it encounters a crossroad. Humans will wait for a specified amount of time to go from exposed to infectious, and from infectious to immune.

The supervisor: As the name implies, this class is responsible for overseeing the simulation. It keeps track of the following values:

1. The number of city blocks forming the city (i.e.: how many crossroads to build on startup)
2. The total number of humans in the simulation
3. The chances of an infected person exposing a susceptible person
4. How long an exposed individual takes to become infectious
5. How long an infectious individual takes to recover

The aforementioned values are able to be modified by the user via the use of the menu.

The supervisor is also in charge of building the simulation. It instances the specified number of humans, creates the city with the specified width, places crossroads appropriately (one in each "node" of the grid) and tells each crossroad which directions are available to humans that step on them. That way, most of the heavy lifting of the simulation is taken care of on start-up, minimising how many calculations need to be done per frame.

It also procedurally generates buildings of varying heights, filling out the gaps between nodes in the city.

Moreover, the supervisor also is responsible for relaying information to the other components. Humans will ask it for the amount of time they need to incubate the disease, as well as recovery time. Crossroads will ask it for the infection rate. The supervisor also has an in-built clock which humans can ping.

Crossroads: Arguably the second most integral part of the simulation, after humans. The crossroads are trigger zones responsible for pathfinding and the spread of infection. Having thousands of colliders and trigger zones moving around is extremely computationally expensive.

The crossroads replace the need for these colliders. As mentioned before, the city is simulated as a grid, so these nodes are placed on intersecting roads. They have inbuilt directions (given by the supervisor on startup) that they can provide to humans that trigger them.

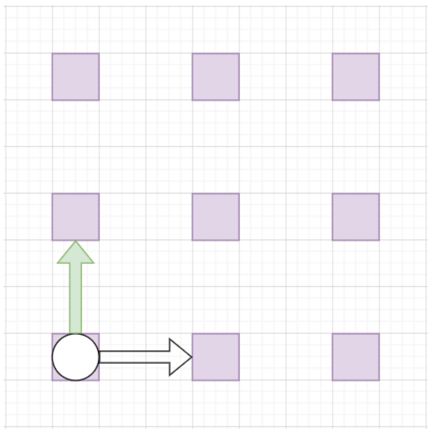
The crossroads take care of the infection spread as well. The infection rate is relayed to them by the supervisor on startup. When an infected human walks over a crossroad, they are flagged as infectious for a fraction of a second. When any subsequent susceptible humans walk over them during this time, there is a chance they are exposed.

Section 2.7, pathfinding: In the simulation the "pathfinding" is more an illusion than an actual calculated path. Having several thousand humans computing paths at the same time takes a heavy toll in performance. Initially, the plan was to have a POI-based pathfinding, where a human would wander aimlessly for a bit, before moving towards a randomly decided point of interest. This is suitable on a small scale, and could feasibly be implemented in the current simulation, but the pursuit was abandoned for the sake of simplicity and performance.

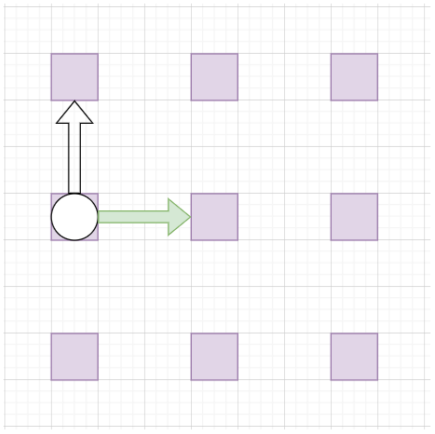
Instead, a human's path is randomly determined by the nodes that they walk on, as previously discussed. Each node has a small list of available directions that they compute on startup. These directions are any that a human could go in and still cross paths with another node. Following this logic, a node at the top of the city only has 3 directions to choose from, and a node in a corner of the city only has 2 (refer to figure 7). When a human walks over the node, the node will pick one of these directions and change the course of the human. A node is not capable of making a human "double back." That is to say, if a human is walking East, and touches a node, said node cannot tell the human to go West. If the node picks out the direction directly opposite to the one the human was walking in, it simply ignores the human and lets it keep walking, since the human is guaranteed to run into another node.

The only nodes that ignore this stipulation are the edge nodes, which are flagged on startup. These have the authority to tell a human to walk in the opposite direction (so as to not walk off the edge of the map). The process of a human walking in a small 3x3 grid is illustrated below.

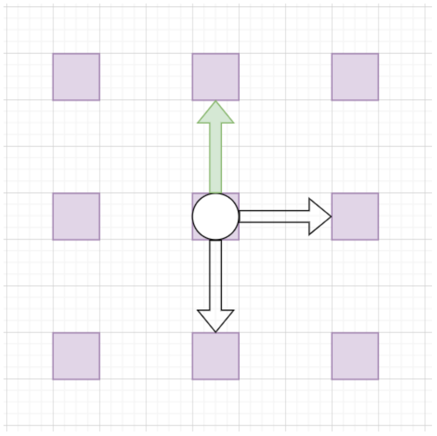
The purple blocks are the crossroad nodes, the circle is the human, white arrows are available directions and green arrows are direction picked.



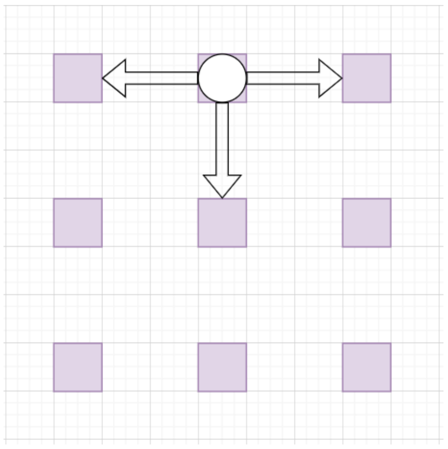
Step 1: The human steps on a node. Since this node is on an edge, it only has 2 available directions that the human can go in. It sends the human North.



Step 2: The human reaches the second node. Once again, this node is on an edge, so it cannot send the human West. It has the authority to make humans going West double back, but not humans going in any other direction. Therefore, it cannot send the human South. It sends the human East.



Step 3: The human reaches the innermost node. Once again, this node is incapable of telling the human to double back, so it only has 3 available directions to choose from. It sends the human North.



Step 4: The human steps on a node. This node is on a North edge, so it can tell the Human to double back, as it is going North.

This way, the simulation has a semblance of recognisable human behaviour, without taking too heavy a toll on performance. This approach was chosen because, as a rule of thumb, the number of humans is far greater than the number of nodes.

However, this approach has many downsides. First of all, it drastically reduces the human object's modularity. It makes humans completely co-dependent on another object. It would be preferrable if humans calculated their own path, rather than having something external do it for them. Moreover, it requires great precision and a deal of care to ensure that they are all placed correctly, with the corresponding directions stored in memory.

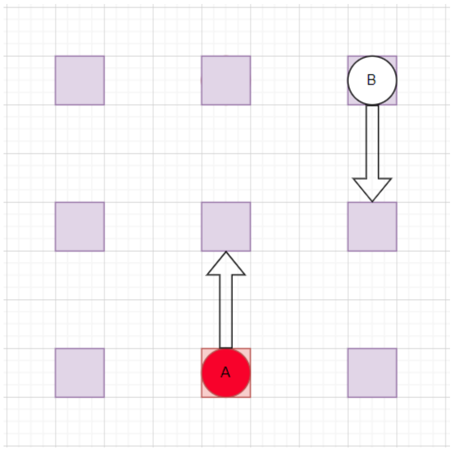
Another disadvantage is its lack of realism. Even if humans are not able to retrace their footsteps generally, there is a chance they may end up going in circles anyway. There is no real human behaviour simulated, just more aimless wandering. Since, generally speaking, there are thousands of humans on the screen at one time, it will hopefully not be immediately obvious to the user.

Section 2.8, infection spread: The infection model, much like the pathfinding, aims to obfuscate its inner workings for the sake of performance. In early iterations of the project, each human had an attached collider, which was very computationally heavy, especially when having to calculate thousands of collisions every second.

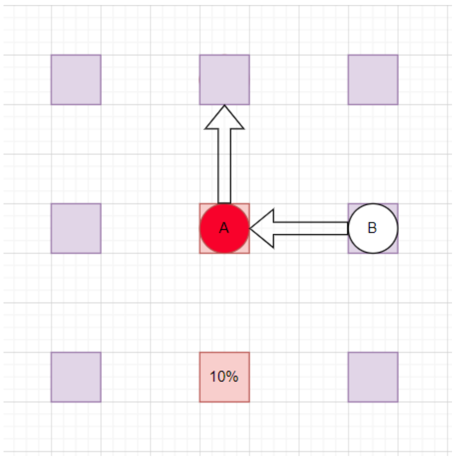
The infection model in the final iteration is built around the pre-existing crossroads "infrastructure". Instead of the crossroads being solely responsible for pathfinding, they also are in charge of spreading the disease. Each crossroads has a flag attached to it, which dictates whether or not it is infectious. When an infected human comes in contact with a crossroad, this flag is set to true.

An infectious crossroad will have a small chance (equal to the infection rate) to infect any susceptible humans it comes into contact with, for a fraction of a second. That way, only humans close to the infectious individual are at risk. If an infected human comes into contact with an already infectious crossroads, the infection rate stored in the node's memory would temporarily increase. Once a small amount of time has passed, the crossroad stops being infectious, and the infection rate is returned to its default.

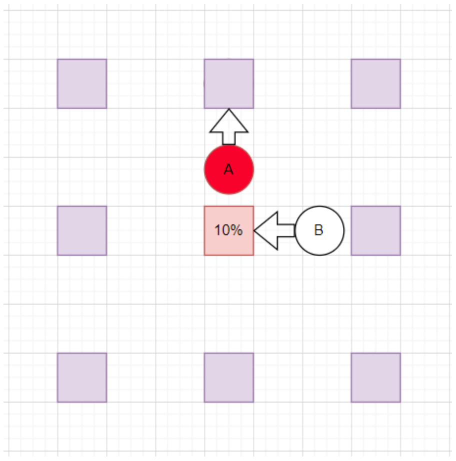
For instance, if an infectious human touches a non-infectious node, the node becomes infectious, with a 10% chance of infecting a susceptible human that touches it. If another infectious human touches this node, the infection chance becomes 20%, 30% and so on. If a susceptible human touches this node, there is a chance it becomes exposed. Once a small amount of time passes from the *last* time the node was touched by an infectious human, it returns to normal. Illustrated below.



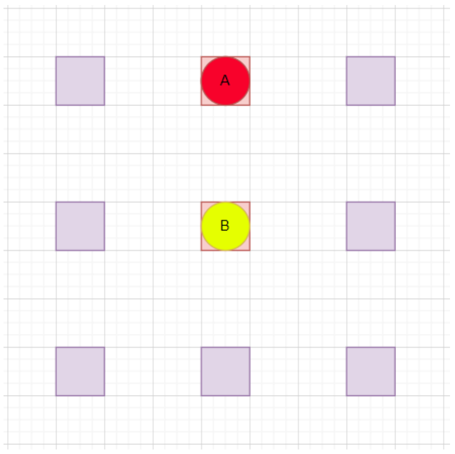
Step 1: Human A (infectious) and human B (susceptible) are following the pathfinding. Human A has made the node it is touching infectious, and has a 10% infection rate.



Step 2: Human A has made the innermost node infectious too.



Step 3: Enough time has passed since human A touched the bottom-middle node, and it has returned to normal. Had an infected human touched it before this happened, the infection chance would have become 20% and the timer would have been reset.



Step 4: human a has infected the top-middle node, and, upon touching the innermost node, human B has become exposed to the infection.

The chances of infection are increased in order to recreate the exponential growth of a pandemic. If infection rates remained constant, so would the spread.

This model removes the need for having thousands of colliders, and is approximately 3 times more efficient than previous models implemented in this project (explored further in the efficiency section). We avoid having moving trigger zones, and instead keep them static for the duration. However, it sacrifices realism for efficiency.

In order for the infection spread to look reasonable, the gaps between nodes must be small, and humans must move very quickly. The time between a node becoming infectious and recovering must be as short as possible, minimising the distance that the infection is able to travel between humans.

Once again, this is not realistic at all. A particularly attentive user may be able to see that humans are only capable of receiving the infection at city junctions. In that way, the program attempts to deceive the user.

Section 2.9, world-building: In order for the crossroads system to work properly, the city must be generated in a grid shape, of width and height determined by a stored value in the supervisor. The user may change this value in the menu. Therefore, the total amount of crossroads equals this value squared.

The supervisor initiates a nested for loop, creating all of the crossroads at a certain distance from each other (this distance is hardcoded and cannot be changed by the user). It then generates buildings where able to, making sure to leave a gap of at least 3 units (in terms of Unity's scale) to avoid humans clipping into the buildings when they walk. This is the simulation's equivalent to streets.

Chapter 3: Results

Section 1: performance analysis

Unity has an in-built performance analysing tool called the profiler. This tool can be used to locate and patch any performance bottlenecks in the project. Below is an example profiler output for the second iteration of the simulation:

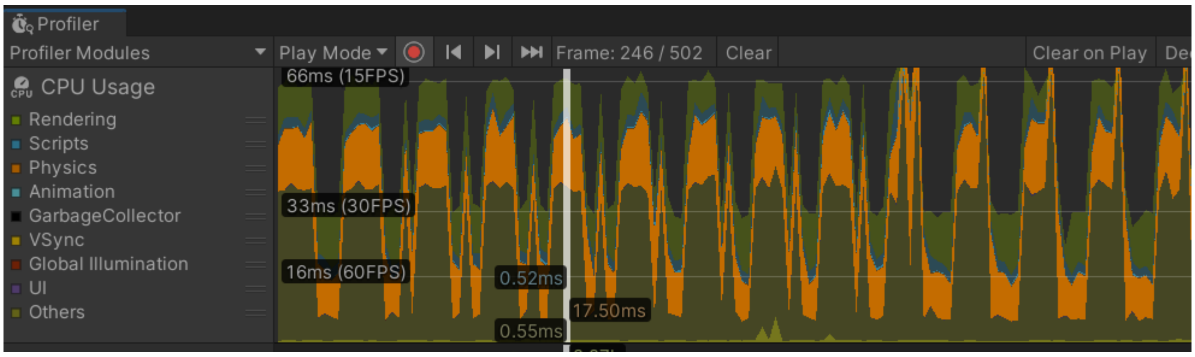


Fig. 7: Profiler output for a legacy version of the project with 2000 humans. Note: the "others" section of the graph is the Unity editor's overhead, which is not an issue once the application is exported.

On this version of the simulation, only around 2000 humans were instanced by the Supervisor. As can be seen in figure 7, the biggest bottleneck in performance consists of Unity's physics updates. This is due to the fact that every human in the simulation has an attached collider.

As mentioned previously, the way the infection spread in this version is via the use of the Unity method OnCollisionEnter(), where each infected human constantly listens for any collisions that occurs and has a chance of spreading the disease on contact. This method has few advantages to offer, aside from its simplicity.

To begin with, moving several hundred objects with colliders is computationally expensive, even if the kinematic property of these objects is disabled (i.e.: if they are physics based or not). Unity has to make many calls per frame just to check what objects are touching. Even with less than a hundred humans, this can add up pretty quickly. Therefore, this approach is inadequate due to its lack of scalability.

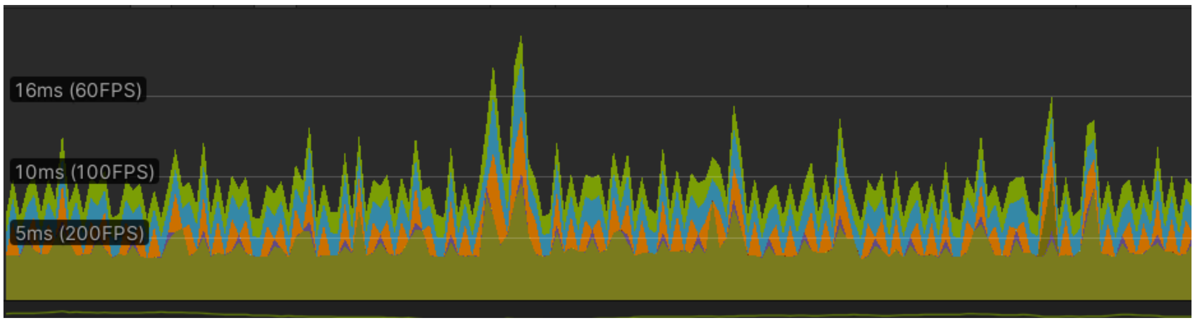


Fig 8: A screenshot of the profiler tool measuring the performance of a version with the crossroads infection model, this instance is running 3000 humans.

Therefore, the goal was to develop a convincing way of spreading the infection utilising a minimal amount of Unity colliders. The first attempt at this was making use of Unity's layers. Each object in Unity has a layer, and can only interact with objects on that same layer. By placing humans on their own layer, it is possible to mitigate the cost of using colliders by ignoring any unnecessary collisions. The impact of this was minimal, however.

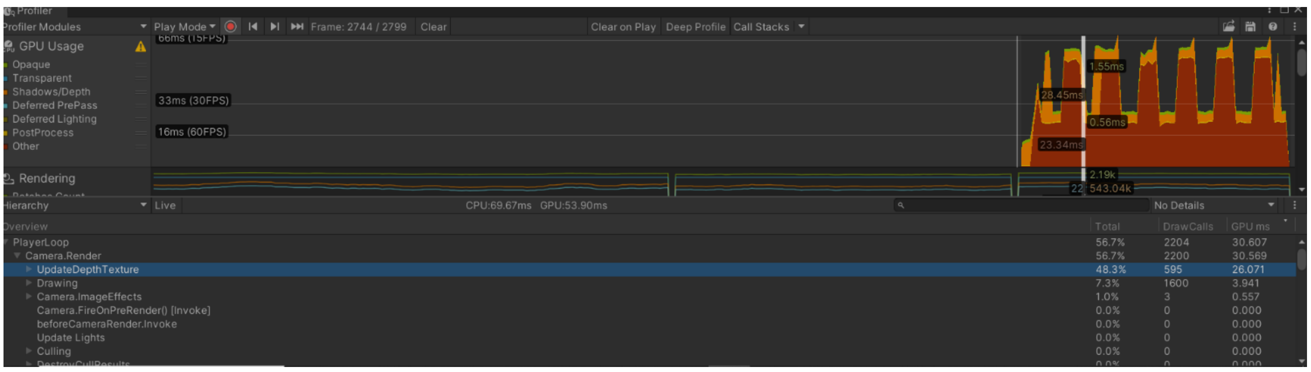
The second, more successful approach was through the use of components. When a human recovers from the infection, they are essentially "useless" to the simulation. They cannot spread or receive the disease, so they are redundant in terms of collisions. With that logic, once a human recovers from the infection, it is safe to remove their collider component.

However, this optimisation only helps the later stages of the simulation, and can take a long time to have any sort of relevance. In addition, since the movement of humans is based on units/frame, it causes a noticeable speed-up of the humans once a certain amount of them recover, affecting the consistency of the project. This issue is discussed in further detail in the consistency section of the report. This optimisation was scrapped due to this fact, in conjunction with its incompatibility with crossroads system.

There were other approaches considered, which included implementing a "radius" around infected humans that they check periodically for objects within it, using the checkSphere() method. The advantage of this would be the lack of colliders, as well as being able to control how many times per second a human checks its surroundings. The idea was discarded in favour of the crossroads approach (discussed in the implementation section).

The crossroads approach avoids using many moving colliders, and instead has a set amount of static trigger zones, which is a lot more efficient. As can be seen in figure 8, framerate is around twice as fast than the project in figure 7, which was running 1000 less humans. The built version of the crossroads infection model can maintain a respectable framerate at up to 8000 humans. This, of course, depends on the machine running the program, but it is a lot better than the older versions.

As discussed in the implementation section of the report, this is not a realistic approach. It is merely an approximation of a realistic infection mode, and relies on the unrealistic parts of the simulation happening fast enough to be passable.



CPU:69.67ms GPU:53.90ms

Overview	Total	DrawCalls	GPU ms
PlayerLoop	56.7%	2204	30.607
Camera.Render	56.7%	2200	30.569
UpdateDepthTexture	48.3%	595	26.071
Drawing	7.3%	1600	3.941
Camera.ImageEffects	1.0%	3	0.557
Camera.FireOnPreRender() [Invoke]	0.0%	0	0.000
beforeCameraRender.Invoke	0.0%	0	0.000
Update Lights	0.0%	0	0.000
Culling	0.0%	0	0.000

Fig. 9: GPU usage with buildings implemented

However, once CPU usage was optimised, the project ran into another bottleneck: GPU usage. Having thousands of individual buildings at a time, all rendered into the same scene puts a strain on the GPU, which affects overall performance. This was solved via the use of Unity's in-built process of batching.

Batching is the process of combining many individuals meshes into a single mesh. This eases the strain on the GPU, and can help a game run many times faster. However, batching has 2 prerequisites: each individual mesh needs to be static, and must share the same material. As consequence the buildings in the project were implemented with these constraints in mind, and kept as simple as possible. This sacrifices aesthetics for performance (figure 10).

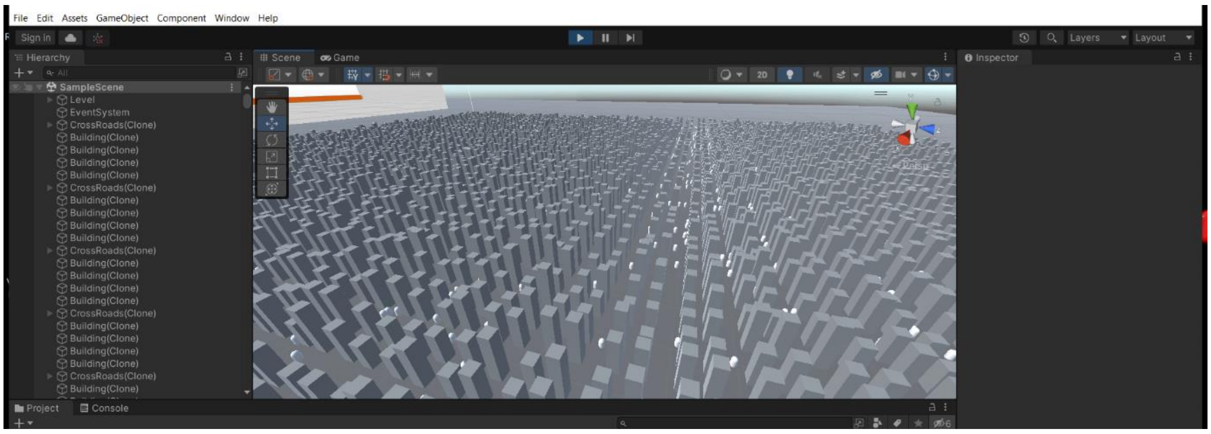


Fig. 10: a very simple city design, the buildings are batched together in order to improve performance.

In the finalised version of the simulation, the biggest performance bottleneck is the behaviour update of humans, or the GPU renderer, depending on how many humans the user chooses to instantiate, and how big the city is. Having an overhead for humans is unavoidable, as moving thousands of objects at one time will be expensive, no matter how optimised the objects are. Moreover, since humans are not static, it is not possible to batch them, which may take a toll on the rendering usage. Even so, the human script is kept as simple as possible, with the only implemented functionality being moving in a stored direction and waiting a few seconds between changing states. This allows for good scalability in terms of simulated population.

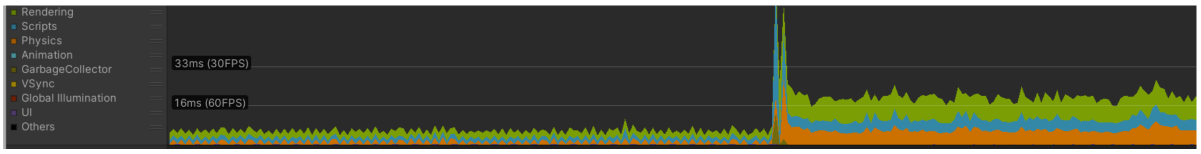


Fig. 11: Performance comparison between 2000 humans and 8000. Unity still has to keep track of collisions with the crossroads objects. More humans means more collisions per frame, hence higher physics based cost. Scripted human behaviour (the great majority of the blue section) starts taking a toll on performance in these numbers.

Chapter 3, section 2: aesthetics

The intention of the project is to illustrate a pandemic to a user, so the project must be aesthetically clear to some degree. That being said, it was the lowest priority out of the goals outlined, so as to not compromise the efficiency and simplicity of the project. When the scope of the project was of a small store, implementing walking animations and particles was feasible, but not with thousands of actors.

The simulated humans were kept as a simple 3D capsule, provided by Unity. The SEIR model was visually implemented through the use of Unity's materials; each category was assigned a distinct primary colour (susceptible is white, exposed is yellow, infectious is red and recovered is green) in order to make it as easy as possible for the user to differentiate them.

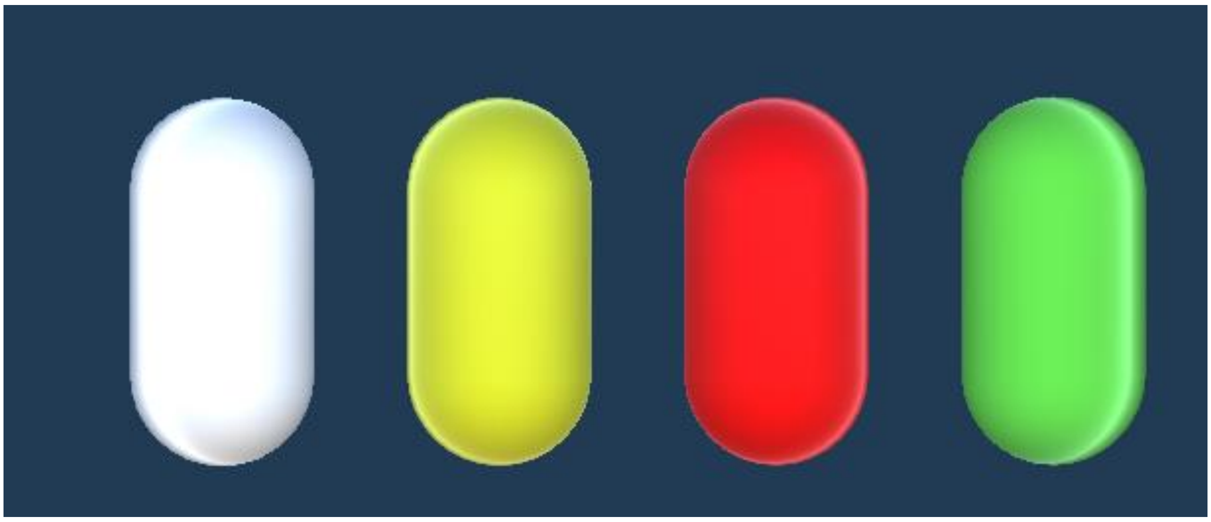


Fig 11: In order of left to right: A susceptible human, an exposed human, an infected human, and a recovered/immunised human.

In order for the simulation to illustrate a city, simple buildings are procedurally generated, filling out the gaps depending on how big the desired city is. These buildings are once again a simple 3D object provided by Unity: a cuboid of varying height. It is important to note that these buildings all share the same material, and are all each composed of a single mesh. This is a prerequisite for batching, which is discussed further in the efficiency and optimisation section.

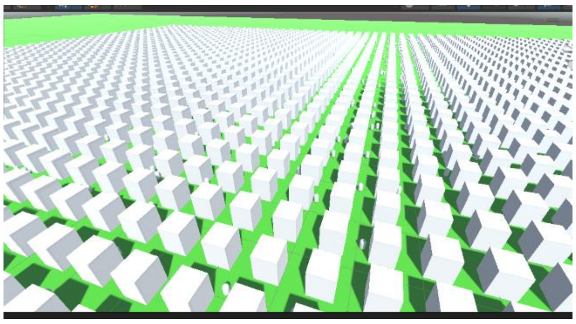
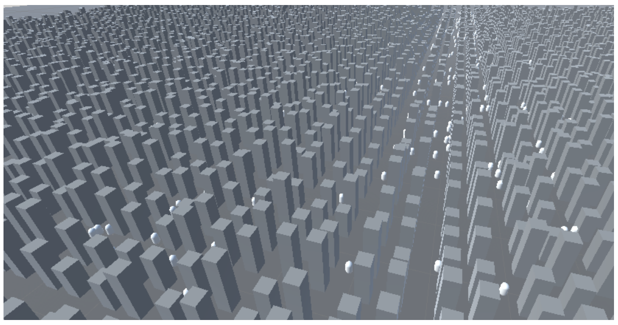



Fig. 12 (left) and fig. 13 (right): Initial implementation of buildings in comparison with the finalised version.

The shape of the city itself is a grid, where gaps between city blocks are left for humans to walk between them, in that way creating city streets. This shape is integral to the pathfinding of the humans, as well as the infection model (discussed in the implementation section, as well as the efficiency section). The project has a predetermined gap between streets in this grid and automatically fits as many buildings as it can in said gap.

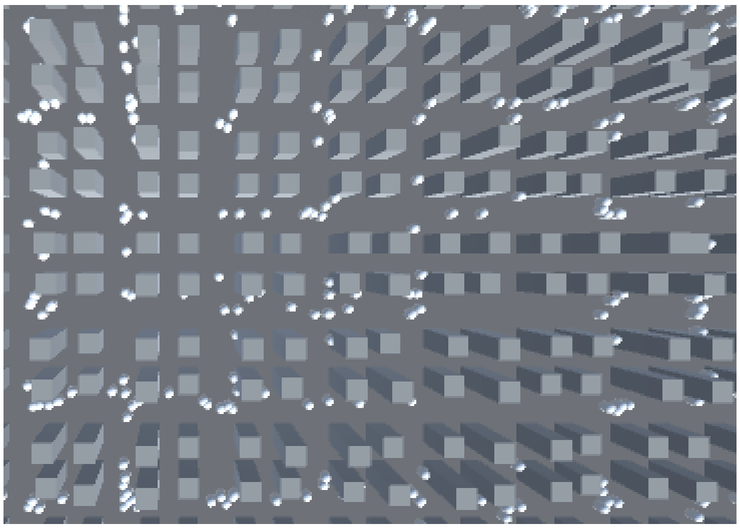
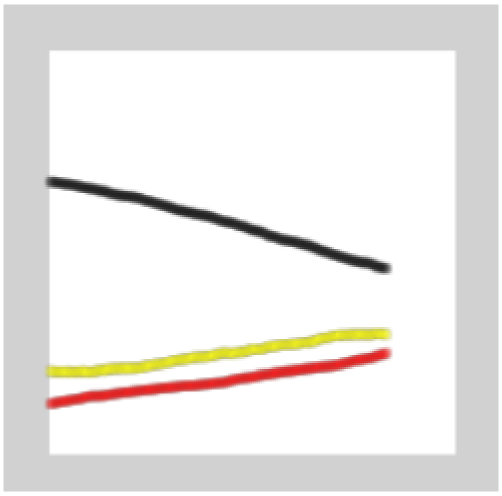


Fig. 14: A top-down view of the city grid, showing how the humans move between buildings.

A simple line graph was implemented to aid in illustrating the pandemic. Three lines, each composed of hundreds of small dots are drawn inside a plane, and are updated each frame and move from left to right. These "dots" will be drawn at a higher or lower level depending on how many individuals are in the category they are representing (as denoted by their colour).

Once the lines reach the far-right side of the plane, the graph is cleared and the process repeats itself.

Fig. 15 (below): An example of the graph. The black line represents the number of healthy individuals, yellow is exposed and red is infected.



Of course, such a simple implementation is not devoid of its downsides. Since the graph is updated and occasionally cleared each frame, implementing accurate axis is difficult. This is due to the fact that framerate tends to be inconsistent, and will often speed up or slow down during runtime.

A possible solution is to implement a graph that updates a certain number of times per second, rather than once per frame. However, this would still require dynamically updating text and many calculations (especially considering that the number of humans will not be consistent).

This would increase the complexity of the graph for a comparably low amount of functionality, and so the graph was left as it is. The goal of the project is to illustrate, not measure, a pandemic.

Chapter 3, section 3: consistency

Due to the way the simulation is implemented, consistency is an area the project struggles in. To start with, in order for the "pathfinding" to work, the movement of each actor must be based in units per frame, rather than units per second. If a human changes their position at an interval that is too big, there is a chance that they may "step over" a crossroads node; their hitbox moves past the node without making contact (figure 16).

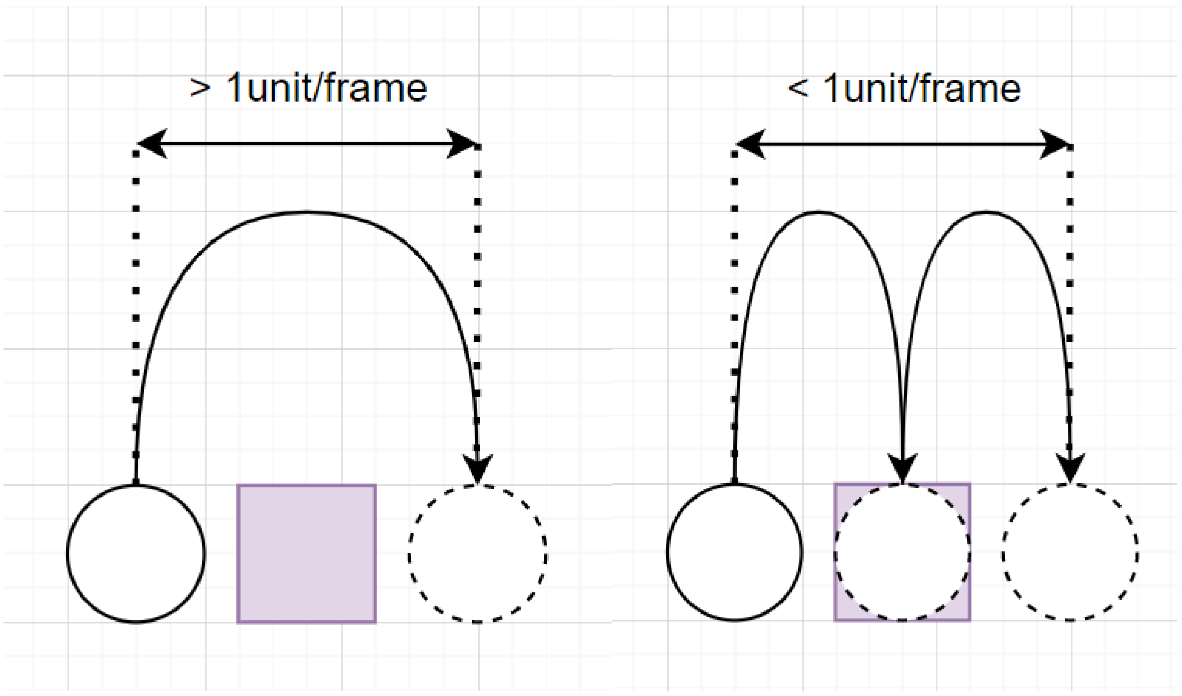
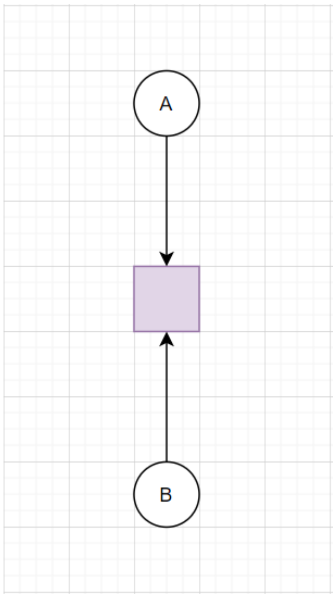


Fig. 16: Comparison between a human taking one large step versus a human taking many small steps. A step exceeding the width of a node (one unit) can lead to some humans not touching them.

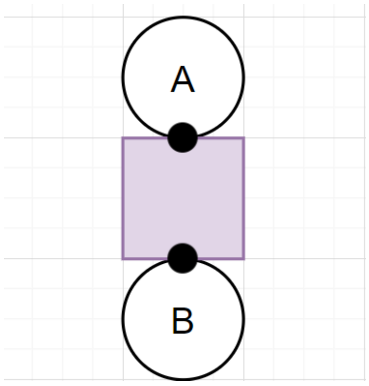
This leads to some humans moving past the intended bounds of the city. This makes it so linking the speed of the human object to the game's clock is unreliable, especially at higher speeds. Moreover, as explained earlier, in order for the spread of the infection to look reasonable, humans must move at a fast pace. This limitation, stemming from the crossroads method, means that humans will walk in sync with the fluctuating framerate, so their speed will not be consistent.

A small remedy for this problem is setting a limit for the framerate. In this case, the project frames per second have been limited to 100.

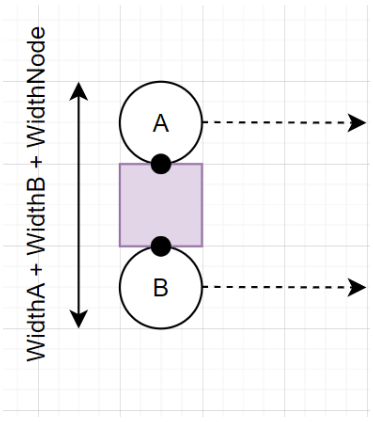
A solution considered for the issue was increasing the size of the trigger zone attached to each node. A larger trigger zone means that humans will not be able to step over it. However, the node system comes with an additional side effect of creating gaps between humans (illustrated below), and increasing node size increases this gap as well.



Step 1: Human A and human B are moving towards the same node. In this case, both human A and human B have matching speeds.



Step 2: Human A and human B touch the node at the same time. A trigger zone will detect the **first** instance that an object's mesh overlaps with its own. In this case, the meshes of both humans do not overlap noticeably with the node, but depending on their speed (which varies per human), their mesh may end up at any point inside the trigger zone (figure 16).

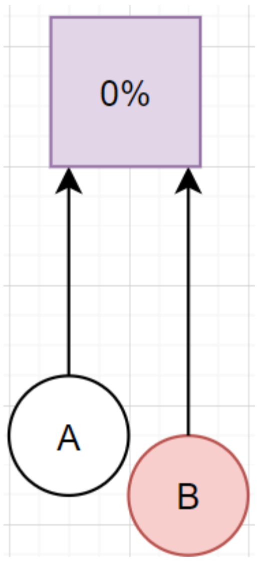


Step 3: The node assigns a direction to both humans, which move with a gap between them, due to the differing "impact zone" with the trigger zone. As mentioned in step 2, a human may end up at any point within the node, so the gap between humans on the same street can be as large as twice the width of the human object + the width of the node object.

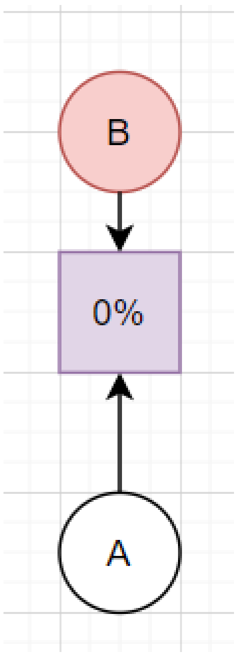
In the case of this simulation, having this gap is suitable, as it forms the likeness of a city street. However, a gap too large sacrifices this.

Yet another drawback of the movement per frame limitation is that it is not possible to optimise movement. Moving a large distance once is a lot more efficient than moving a very short distance many times. Nevertheless, the efficiency of the crossroads implementation still greatly outweighs any other method attempted for this project.

As mentioned before, the infection model of the project lacks in realism and consistency. Consistency was always going to be a problem, since the infection spread is based on chance, but the simulation may also produce unexpected results in very specific situations.



Scenario 1: Human A (susceptible) and human B (infectious) are moving in the same direction, at the same speed. They are virtually next to each-other, but human A is slightly ahead of human B. The node they are walking towards is not infectious. No matter how much time human A and human B spend next to each-other, human A will not be infected. This is because a node only triggers once, on the first frame a human touches it, to save performance. Therefore, human A will touch the node, having no chance of being infected. Shortly after, human B will touch the node, and it will become infectious. Since the node only triggers once, it does not matter if human A is still in the node's mesh, human a will not become infectious.



Scenario 2: Human A (susceptible) and human B (infectious) are walking towards the same node, going in opposite directions. Human B is slightly closer to the node than human A, and is going at a similar speed. When human B reaches the node, it becomes infectious. Once human A reaches it, there is a chance it becomes exposed, even if contact with human B is minimal. Humans move at a very fast rate, and they are moving in different directions, so there would be a tiny window of time where human A and B are close to each-other.

Chapter 3, section 4: usability

A user interacts with the simulation primarily via the use of the menu implemented. The menu is brought up by pressing the 'escape' key, and is capable of modifying many of the key parameters in the simulation (the effects of which have been explained prior). This way, a user can observe a variety of different scenarios, each of which produce different results. The graph in the simulation aides in illustrating these differences.

The menu is composed of three buttons and a few sliders. The sliders are linked to the stored values in the supervisor, and the buttons allow the user to exit the application, reset the parameters to their defaults, and rebuild the simulation. In order for the inputted changes to take effect, the user must restart the simulation.

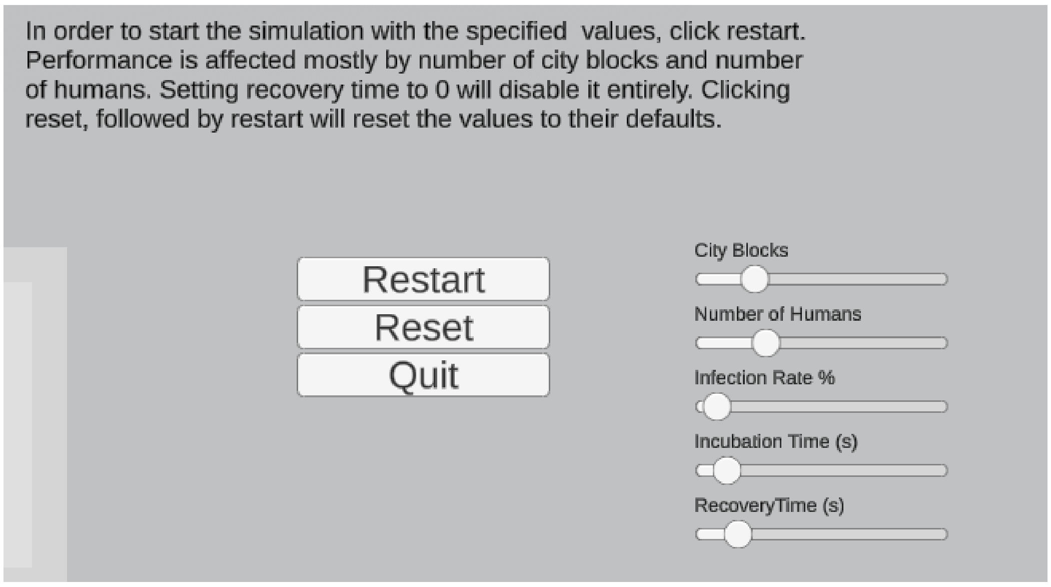

In order to start the simulation with the specified values, click restart. Performance is affected mostly by number of city blocks and number of humans. Setting recovery time to 0 will disable it entirely. Clicking reset, followed by restart will reset the values to their defaults.

Fig. 16: The menu. The sliders allow the user to change many aspects of the simulation, such as the size of the city, how infectious the disease is and so on. The 'restart' button rebuilds the simulation with the inputted values, 'reset' returns these values to their default (the numbers assigned on application startup), and 'quit' exits the application.

A user may also control the bearing and position of the camera, by pressing the 'space' button. WASD controls the camera position, and mouse movement is linked to its position. This way, the user may observe different parts of the generated city and see how the scenario unfolds.

These features allow for increased usability and let the user to interact with the simulation. However, the simulation still has some issues which may seem unintuitive to some. For instance, due to the nature of reloading a scene in Unity (i.e.: rebuilding the simulation) the sliders will be set to their default valuers visually, but not functionally.

Chapter 3, section 5: realism

In the previous section, we explored the lack of consistency that arises within the infection model, stemming from the implementation. In this section, we will see how these inconsistencies measure up to the real world. As mentioned in previous sections, the aim of the project was to illustrate a pandemic rather than collect and predict accurate information. Along development, the scope of the simulation grew, and efficiency became a focal point. As consequence, the realism of the simulation was kept to passable at best.

Section 6.1, human behaviour: The humans in the simulation travel at an accelerated pace, which varies from human to human. This is to simulate the different walking speeds of humans, as well as to introduce a small random element in the simulation. The humans' speed doubles as both a way of hiding the inner workings of the infection spread from the user, and simulating time passing by quickly.

Humans will wander around the city, aimlessly. They will be told by crossroads that they walk on which directions they are capable of moving in, as well as keeping them within the bounds of the city. Additionally, humans can only walk in 4 directions. This is mostly due to the limitation of having thousands of humans at one time.

Obviously, this is not accurate to true human behaviour, especially during a pandemic. The recent COVID pandemic has shown first-hand how social distancing affects society as a whole. In our simulation, humans do not "react" to the spread of the disease, and their behaviour remains constant. Individuals do not practice social distancing, do not frequent specific buildings (such as their home of place of work) and do not interact with their environment in any way.

Moreover, as mentioned previously, humans are only capable of being infected at city junctions, which is wildly inaccurate. In the earlier iterations of the project, the more realistic, but once again, inefficient, approach was contact based. According to the CDC, the majority of contact-based diseases spread through improper hygiene (such as people not washing their hands). In the final iteration of the project, the infection spread is more akin to an indirect transmission disease, as humans can infect each-other at a distance. These ideas have proposed implementations in the 'ideas' section of the report, where they are discussed in-depth.

In the final iteration, the material of the city generated was switched to a dark grey to imitate concrete.

Chapter 4, discussion

Section 1, Future ideas

Since the focal point of the report became optimising the infection spread to handle thousands of humans, implementing many features fell by the wayside.

An idea that was part of the original iteration was adding points of interest for humans. The initial pathfinding for humans was going to be POI based, where a human would travel from one point of interest to another. This idea was later abandoned in favour of the crossroads implementation, but it would still be possible to include some version of it in the project, in a future iteration.

To begin with, create several, randomly placed points of interest throughout the city. The crossroads object, thanks to its relative simplicity, requires few changes to work in this context. Inside each of these nodes, include a stored string, which will hold the tag corresponding to a point of interest on the map, as well as a stored direction. The next step would be to find some way to create a path of nodes, leading to their POI. That is to say, one node would point to the next, which would point to the next node in the chain until it reached their corresponding POI. All a human would have to do is wander randomly until they find a node with the attached tag of the POI they are trying to find.

Expanding on the point of interest idea, one of these POIs could be a hospital, for instance. Humans could try and head to their nearest hospital upon becoming infected, and potentially recover there. However, as the number of infected people increases, so does the amount of people at the hospital, increasing the risk of infection in the area. This would simulate the strain on hospitals during the COVID pandemic, as an example.

As briefly mentioned in the realism section, another possible feature of the simulation could be indirect methods of transmission. Utilising the checkSphere() method for Unity, one could simulate a “sneeze”, where all humans in a certain radius (determined by the user) are at risk of becoming exposed. In order to preserve performance, these “sneezes” could be limited to once every few seconds.

Having humans with different levels of hygiene could also be included in a future version. It would be relatively simple to include a copy of the infection rate inside of each human object, and have it vary between individuals. Some humans would be more hygienic, and therefore have a lesser chance of spreading the infection.

Including some way of illustrating the impact of social distancing in pandemics could be a future improvement of the project. Humans would have an allocated “home” building. Humans would spend time in this building, until they decide to leave and travel around the city. Homes to multiple humans may become infectious if a human leaves and is exposed in the city. Enabling social distancing would reduce the chance of humans leaving their homes, therefore slowing the spread of the infection.

Chapter 4, section 2: Conclusions

The ultimate goal of the project, as it evolved, became to find a method of convincingly spreading the infection, that was efficient enough to support thousands of humans at one time. Even though aesthetically the project was kept as simple as possible, I believe that the crossroads method is a successful solution to this goal. While there are likely many other methods that can achieve the same effect, the crossroads method is (to my knowledge) an original approach. I am satisfied that I have managed to produce a unique answer to a unique problem.

List of references

Appendix A

Self-Appraisal

A.1 Critical self-evaluation: While I am mostly content with how the project turned out, there are no end of small, unpolished details that bother me slightly. For instance, as mentioned in the usability section, the slider defaults are not particularly clear sometimes. The camera also displays some strange behaviour when looking straight down, not to mention the occasional dips in performance. I think it may have been possible to get to these small issues properly, had I started the report earlier.

I left the report for last as I do not enjoy writing as much as I do programming, and, as consequence, the amount of time I had to patch anomalous behaviour was reduced. Moreover, the aesthetics of the project are bland, and the city itself is ugly. While I am no artist, I am sure that I would have had more time to polish it, had I been more organised.

The project itself is functional, and I believe it demonstrates how the crossroads approach implemented could be effective in certain contexts. Grid pathfinding (discussed in the background section of the report) is an established pathfinding method, and the infection model could be repurposed to fit in that context.

The humans ended up being quite simple due to the performance constraints, and I would have liked to see if I could have developed them in some way, and included some sort of behaviour simulated. The project does not have many distinct features implemented, since most of the interesting code occurs “under the hood.”

Another factor that may have slowed down development was the lack of proper version control, such as GitHub. I had different iterations of the project saved locally on my computer, each of them very different from the other, but that was the extent of my version control. Using the GitHub infrastructure may have helped me in refactoring my code, and saved me some time. Moreover, version control is generally good practice when coding.

A.2 personal reflection and lessons learned: I have learned that having a steady routine of work, where one does a bit of work each day, is better than large, unreliable, bouts of productivity. Moreover, planning a project far in advance may have saved me some of the different iterations implemented. Foresight is important.

In general, I am happy with the work I have submitted. I ended up developing a healthy work schedule of spending several hours a day working on the project, without burning myself out. This allowed me to keep getting fresh ideas, without tiring myself out.

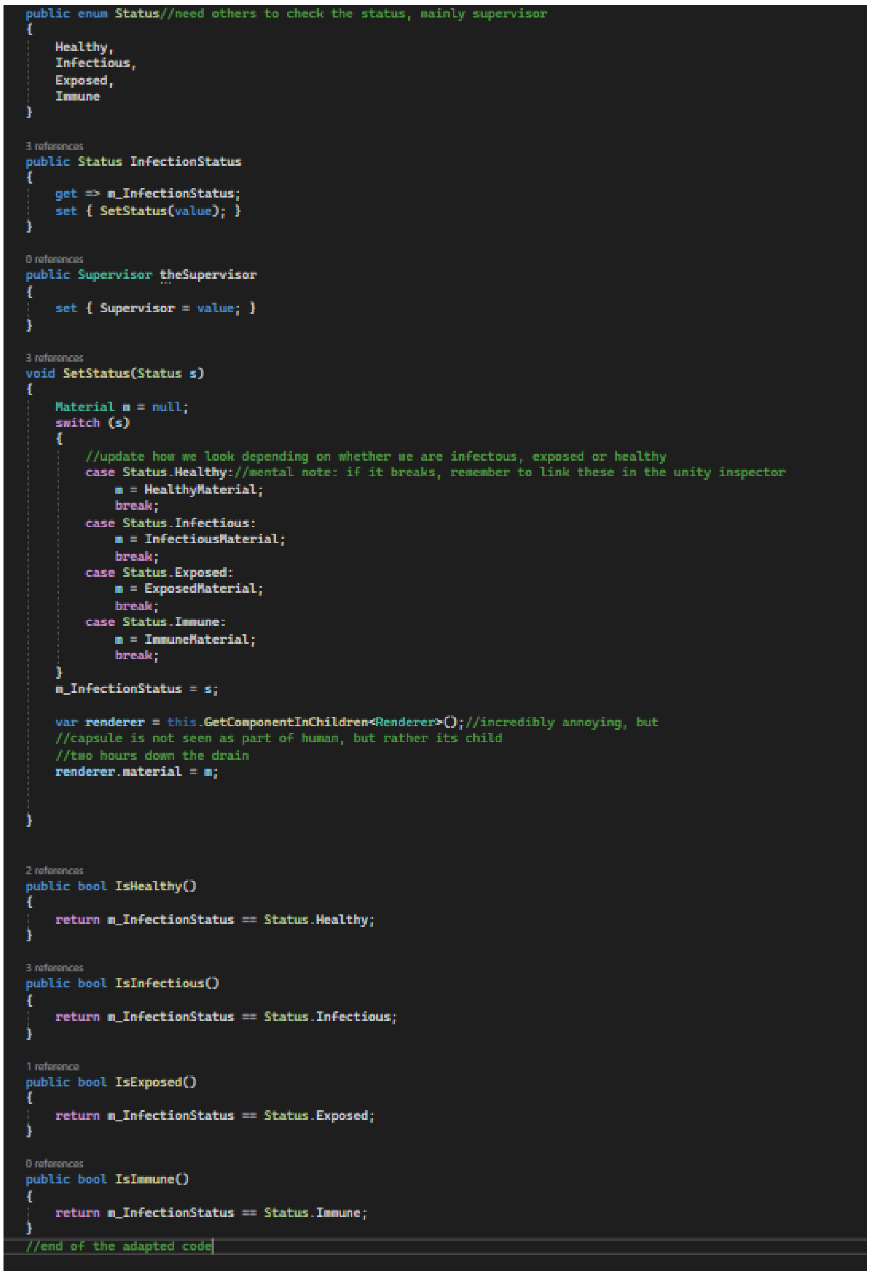
A.3, ethical, legal and social issues: During the course of the project, I did not run into any issues of this nature, as I focused on programming a small, game-like simulation. Any parts of the code which are not entirely mine are properly credited and highlighted within the code.

Appendix B
External Materials

The status code for the human script (from line 28 to line 99) has been adapted from the unity store simulation found at https://blog.unity.com/industry/exploring-new-ways-to-simulate-the-coronavirus-spread

By James Fort, Adam Crespi, Chris Elion, Rambod Kermanizadeh, Priyesh Wani, and Danny Lange.

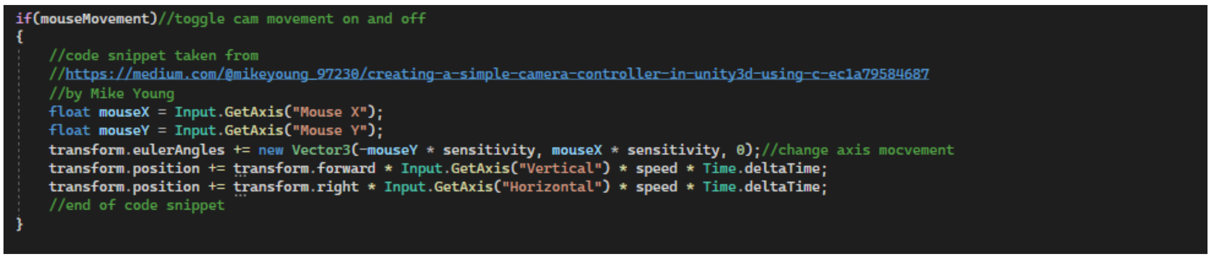
The source code of which can be found at https://github.com/Unity-Technologies/unitysimulation-coronavirus-example



```
public enum Status//need others to check the status, mainly supervisor
{
    Healthy,
    Infectious,
    Exposed,
    Immune
}

3 references
public Status InfectionStatus
{
    get => m_InfectionStatus;
    set { SetStatus(value); }
}

0 references
public Supervisor theSupervisor
{
    set { Supervisor = value; }
}

3 references
void SetStatus(Status s)
{
    Material m = null;
    switch (s)
    {
        //update how we look depending on whether we are infectous, exposed or healthy
        case Status.Healthy://mental note: if it breaks, remember to link these in the unity inspector
            m = HealthyMaterial;
            break;
        case Status.Infectious:
            m = InfectiousMaterial;
            break;
        case Status.Exposed:
            m = ExposedMaterial;
            break;
        case Status.Immune:
            m = ImmuneMaterial;
            break;
    }
    m_InfectionStatus = s;

    var renderer = this.GetComponentInChildren<Renderer>();//incredibly annoying, but
    //capsule is not seen as part of human, but rather its child
    //two hours down the drain
    renderer.material = m;


}


2 references
public bool IsHealthy()
{
    return m_InfectionStatus == Status.Healthy;
}

3 references
public bool IsInfectious()
{
    return m_InfectionStatus == Status.Infectious;
}

1 reference
public bool IsExposed()
{
    return m_InfectionStatus == Status.Exposed;
}

0 references
public bool IsImmune()
{
    return m_InfectionStatus == Status.Immune;
}
//end of the adapted code
```

The camera control code, inside the camera object script, has been adapted from https://medium.com/@mikeyoung_97230/creating-a-simple-camera-controller-in-unity3d-using-c-ec1a79584687

By Mike Young.



```
if(mouseMovement)//toggle cam movement on and off
{
    //code snippet taken from
    //https://medium.com/@mikeyoung_97230/creating-a-simple-camera-controller-in-unity3d-using-c-ec1a79584687
    //by Mike Young
    float mouseX = Input.GetAxis("Mouse X");
    float mouseY = Input.GetAxis("Mouse Y");
    transform.eulerAngles += new Vector3(-mouseY * sensitivity, mouseX * sensitivity, 0);//change axis mocvement
    transform.position += transform.forward * Input.GetAxis("Vertical") * speed * Time.deltaTime;
    transform.position += transform.right * Input.GetAxis("Horizontal") * speed * Time.deltaTime;
    //end of code snippet
}
```