Modelling a modern day pandemic

Benji Tigg

December 2020

Abstract

This report aims to aims to discuss the two main types of models used to show the spread of a infectious disease over time: the differential equations model and agent based modelling. It aims to come to the conclusion that the best type of model to use to show the spread of a modern day pandemic depends on your required use case, it does this by looking at the 3 main aspects of each model model: what the model is and how it works, how each model is built and the difficulties that can arise from the process of building it and the final aspect is the results that the model outputs. This report also aims to show how data can be collected and turned into useful constants that can be used within the model.

Contents

1 Introduction

Since the turn of the millennium we have witnessed; multiple outbreaks of the Ebola virus, an outbreak of the Zika Virus, H1NI(the Spanish flu) made its reappearance in 2009 and 3 different coronavirus outbreaks; SARS, MERS and COVID-19 [1], and we are at the point now when it's not a question if a major pandemic will happen it's when it will happen, as was shown by the COVID - 19 pandemic [2]. One thing we can do to reduce the impact of a pandemic is to model the spread of a disease before the disease spreads.

There are 3 main types of disease modelling used in **epidemiology**: statistical based methods, mathematical models and empirical/machine learning models [4]. As this paper is focusing on which method is the best to model the spread of a pandemic before it happens, it will be looking at mathematical modelling as the other two methods predict how a disease will spread after it has started spreading. Mathematical models allows us to test theories about how the spread of a disease will change without testing in the real world which is: time consuming, expensive and also has few ethically gray areas.

1.1 Background information

In the UK we use a couple of models to help predict the spread of SARS - nCov - 2 (COVID - 19). The main model that the country uses is the imperial college model which is a modified influenza model [5]. Modifying pre - existing models is not uncommon this is because lot's of disease have common behaviours. We know from observation that both SARS -nCov - 2 and seasonal Influenza (H1N1) exhibit the same characteristics in how they transmit themselves, but also: who is most at risk, how long you are contagious for before infection and symptoms. This makes a influenza transmission model a perfect candidate to modify. As you progress through the paper you will also find that this can save a lot of time.

1.2 Virus naming conventions

COVID - 19 tells us that the virus is a coronavirus that was fist discovered in 2019, this isn't very helpful as the common cold is a coronavirus. This is why we look at its second name SARS - nCov - 2, this name tells us that COVID - 19 is a second generation SARS based novel coronavirus. SARS stands for severe acute respiratory syndrome and novel tells us that nobody is immune to the disease. Whilst knowing that this disease is SARS what really helps us is knowing that this disease is a novel virus, knowing that nobody is immune will really help with developing the model.

2 Types of model

When it comes to trying to predict the spread of a disease using mathematical or computational techniques we really only have two main methods: A system of differential equations and agent based modelling. Each model uses the standard SIR model: susceptible, infected and removed, however this list can be built upon. For example the imperial college model used SLIRD: susceptible, **latent**, infected, recovered and dead. However for the purposes of this paper we will sticking with SIR and SIRD.

2.1 System of differential equations

2.1.1 What is it?

A model that relies on a system of differential equations is known as a continuum model. In the mid 1920's 3 papers were published William Kermack and Anderson McKendrick that built upon the work by Ronald Ross that showed that a set of equations could show the spread of a disease, when time was the function. They introduced the idea of compartmentalising a population which allows us to produce SIR models. They also suggested that the probability of someone being infected with a disease was proportional to the number of infected people they come into contact with, as this increases with the number of infected people, the probability of being infected is proportional to the number of infected people.

$$\text{Rate of increase in infected} = k \times S \times I \tag{1}$$

With K being the chance of being infected, S being the number of susceptible individuals in the model, and I being the number of infected individuals.

$$\text{Rate of change in loss of susceptible individuals} = -kSI \tag{2}$$

$$\text{Rate of recovery} = \lambda I \tag{3}$$

With λ being the chance of recovery.
This gives us 3 main equations

$$\frac{dS}{dt} = -kS(t)I(t) \tag{1}$$

$$\frac{dI}{dt} = KS(t)I(t) - \lambda I(t) \tag{2}$$

$$\frac{dR}{dt} = \lambda I(t) \tag{3}$$

Equation 1 shows that the rate at which the number of susceptible individuals changes is equivalent to the negative increase in the number of infected individuals. Equation 2 shows that the rate at which the number of infected individuals changes is equal to the increase in the number of infected individuals minus the rate at which individuals recover or die. Equation 3 shows that the rate at which the number of removed individuals changes is the same as the rate at which individuals are removed from the model either by recovering or dying. The equations can then be solved for time and depending on the input parameters will produce a graph like this.

```
Total Population: 500000
Total Number of Infected at time = 0: 1
Total Number of Removed at time = 0: 0
Chance of Infection: 0.2
Fraction of People who recover per day: 0.05
Time for which the model should run: 300
```

Figure 1: Model parameters. K = 0.2, λ = 0.05, S(t = 0) = 49999, I(t = 0) = 1 and R(t = 0) = 0

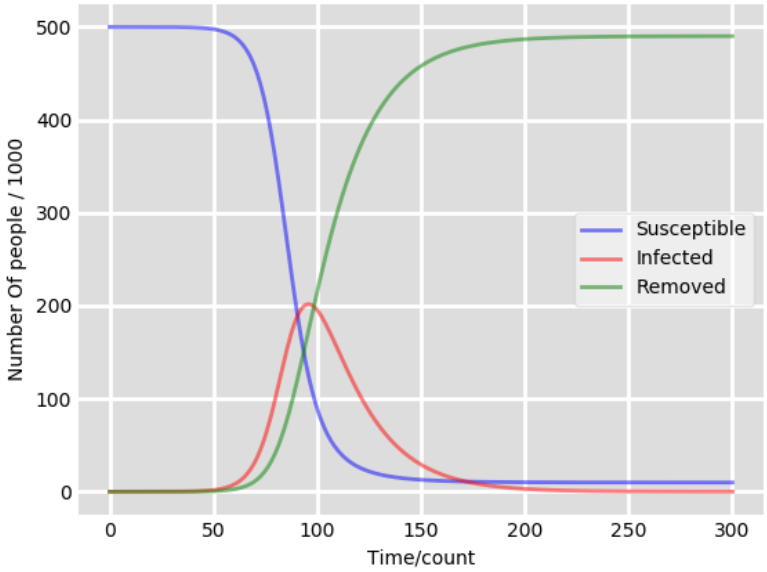


Figure 2: Graph to show the differential equations model with all the parameters shown in Figure (1)

The report will discuss the shape of the graph in more detail in the results section but a brief overview of what the model shows is that as the number of infected individuals increases so will the rate at which it increases by. This will eventually peak when the number of people infected or removed is greater than the number of people who are still susceptible. Both the susceptible and removed graph will plateau when the gradient of the curve showing the number of infected is 0.

2.1.2 Advantages and disadvantages

A benefit of this type of model is that it requires very little computational power due to only type of computational work in this model being lots of number crunching when done by a computer this model can extremely fast, however due to the model not being a stochastic model it means that results between different runs of the model, with the same parameters, will not change. Whilst this allows for people to verify the results produced by the model, the lack of variation in results means no nuances in the way the disease spreads can be identified.

A further issue with this type of model is that, as you add more and more variables the equations will get progressively more complicated to write and to solve, this will increase the development and run time of the model. This type of model also produces very crude and basic results that whilst easy to read and understand don't tell you anything about how it spreads and who it effects the most.

2.2 Agent based modelling

2.2.1 What is it?

In agent based modelling a system is modelled as a collection of agents who make their own decisions based on a set of given rules. In the context of **transmission models** we model every member of the sample population as their own agent. Unlike the system of differential equations model, agent based modelling is very much a stochastic model however it still builds upon that same idea of compartmentalising the population into separate categories. ABM's are traditionally done using computers however one of the first ABM's used to model segregation in 1960's America was done on paper by Thomas Shelling. However unlike modern ABM's Shellings model was a Cellular Automata (CA) model similar to that used in Conway's game of life. ABM's grew from the foundations that CA's laid and like ABM's each agent (cell) in a CA has multiple states it can be in (e.g. susceptible and infected). At the end of each iteration each cell looks at the state of each cell in it's neighbourhood, in Conway's game of life this is a 3 by 3 square centred on the subject cell, and it will determine it's particular state based on that in accordance to the rules set by the model [9]. This is very similar to how a transmission model works.

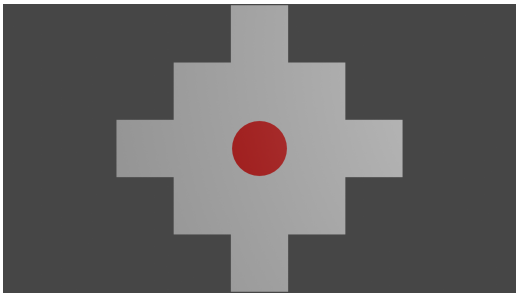


Figure 3: A infected agent casting a infection circle

In figure 1 it shows a infected agent casting out a circle of a given radius, any susceptible agents within that circle would have a chance of being infected determined by the rules set out by the model.
An ABM for the transmission of a disease can be a simple as modelling the interactions between people in a community.

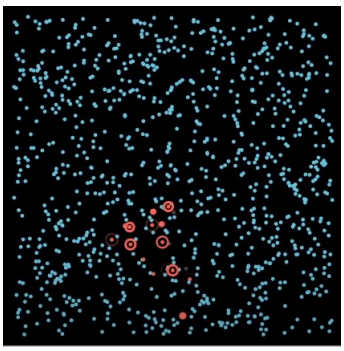


Figure 4: A model to show the interactions between people [8]

To a more complicated model where you agents are: going to school / work, use public transport, visit public places like; restaurants, parks, shopping centers and places of worship and people die and new people get added to the model. For example the ICL model that is used to model the spread of COVID - 19 in the UK is a ABM models: The interactions between agents, agents going to school

and work, agents being at home, agents leaving and entering the country via airports, agents going to hospital and agents moving around by walking but also public transport.

2.2.2 Advantages and disadvantages

Unlike the system of differential equations model, this type of model is a stochastic model the results will be different every time. This can lead to the model showing **emergent phenomena**, this would not be shown in a model using a system of differential equations. This allows for more effective decisions to be made. This unfortunately means that results aren't repeatable and as shown later in the paper it isn't always often clear if the model is working as expected.

The model also has the advantage that your a modelling human behaviour and so the results produced by it will accurately represent what would happen if you introduced the virus to a community, the benefit of using the ABM to do this is that you avoid the ethical gray area that would arise from unloading a potentially deadly disease into a community. However one issue that does arise from this approach is modelling human behaviour. Humans are very complex creatures and trying to simulate how one would behave in this scenario is computationally expensive and often produces unexpected results.

The level of detail that a ABM gives you does come at a cost though, as said in the previous paragraph modelling human behaviour is computationally expensive, but so is increasing the number of variables e.g. the number of agents, this will directly affect the time it takes and the computational power required to compute the model will increase and depending on how well the model is written will determine the scale at which the model increase by. But it's not just a computational cost that exists there is a labour cost as well, like the system of differential variable model as you add more and more systems to your model the more complex it becomes. e.g. adding a transport network system in requires path finding to be added and some many more systems. Sometimes though the detail provided by the results of the ABM aren't all that useful as The emergent phenomena illustrated by ABM's just don't tend to be found in the real world. This is because ABM's are extremely sensitive to the conditions of the environment that the model is in and even the smallest of change can completely change the outcome of the model [10].

2.3 Conclusions

At this point in the comparison it seams as though the ABM's is a much better modelling technique due to the increased depth that the results give. ABM's also allow for emergent phenomena to be observed which gives a greater understanding of how a disease will spread in a community compared to a system of differential equations. However the differential equations model does have some benefits over the ABM, it's fast and doesn't have much room for error. However at this stage the advantages of the ABM out-weight the advantages of the differential equations model in all situations.

3 Building the models

In this section of I am going to discuss the development of each model and the advantages and disadvantages that each model has when it comes to developing them.

3.1 System of differential equations

3.1.1 Development

Using the equations that are in the first section and using the SCIPY library within python, which contains functions for maths, science and engineering. It is possible to solve the system of equations with time as a parameter.

```
import numpy as np
from scipy.integrate import odeint

def Find_Deriv(y, RunTime, N, alpha, beta):
    S, I, R = y
    dsdt = -alpha * S * I/N
    didt = (alpha * S * I/N) - (beta * I)
    drdt = beta * I
    return dsdt, didt, drdt

coord = odeint(Find_Deriv, y_0, RunTime, args = (N, alpha, beta))
```

The rest of the code within the model, is used to plot the values onto a graph produced by matplotlib. The model at the start takes in values for alpha (Chance of infection) and beta (Chance of recovery). The model also takes inputs for the total number of susceptible, the total number of infected and the total number of people removed from the model at time 0. This enables to model to use the differential equations.
The full code can be found in the appendix under System of differential equations model.

3.1.2 Advantages and disadvantages

There isn't really much to discuss about the advantages and disadvantages when it comes to developing the system of differential equations model, as the developing the model took around half and hour to write and the model itself runs really fast due to the small amount of computational power required to produce the graph. And the code base for the model is around 100 lines of python code This makes finding and fixing any bugs within the model easy to do.

3.2 Agent based modelling

3.2.1 Development

Unlike a system of differential equations mode an ABM requires much more planning, this is due to the amount of code that will have to be written and how complex that code can become as you add more and more features to your model.

When building a agent based transmission model, a couple of features need to be included A scenario editor, a school / work system, public transport, the ability for agents to go and visit public buildings and a hospital system. C++ is the models primary language with python being used to analyse the data the model is producing, this is because C++ is a optimised for **object orientated programming** and this meant that the model could support lots of agents but also all the necessary buildings, that would be in the model environment, that would be required for certain features to run. C++ is also used due to the increased control you have over memory management, in a environment where thousands of objects, knowing where they are, via **pointers**, and being able to delete objects and remove them from memory when the I know / the model knows they aren't needed anymore is very useful as it keeps the memory usage down. C++ has other advantages over other languages that have a concept of classes e.g. PYTHON. C++ is a compiled language compared to a language like python that is interpreted. This means that the time taken for the code to run will be less and so it allows for faster simulations, allowing for larger simulations to be run.

With the scenario editor being a feature of the ABM, this allows for user customization and thus this means there is room for user error, this is where the logging system comes in. The logging system is designed to catch errors that would crop up if there was issues with the scenario that the user had created. The logging class had a couple of main features: It could write to a given file that is the log file, during the time that the log class is within scope it records the issues with the model and it can get the current time so the user can see when errors occur.
The Log class has a set of messages, stored within a map that it can write to the log file, whilst I could of had a vector of strings where each string was a different message and that message being a variable that is passed in when you call the log function, the current system saves on the processing that this system would require.The entire code base for the logging system can be found in the appendices.

If I was to talk about every utility within the software this paper would get extremely long so moving on to the scenario system. The scenario system was the second major feature added to the system due to it being a requirement for the rest of the model to run. The scenario system outputs to it's own custom file type **.CVSN** (Coronavirus scenario). However the file type is a text file instead of a binary file, this makes reading the file into the system easier than if it was a binary file and it also allows for the user to edit the file in a traditional text editor. Editing the scenario file outside of **COVIDSIM** is not recommended due to the amount of data a scenario file contains and the syntax of the scenario file.

```
Scenario_Name: test
World_Infomation

Tile_Size(meters)
20

Num_of_tiles
10

Num_of_Unique_Buildings
10

Unique_Buildings
Public_Services
Education_buildings
Num_of_Primary Schools
2

Primary Schools 1
1
1
1
9
50

Primary Schools 2
6
6
9
20
100
```

Figure 5: An extract from a scenario file.

Each unique building stored in the scenario file stores it's location x, y, tile number with the left bottom corner being (0,0). Transport buildings also store type, e.g. airports have 2 different types: Regional and international. The scenario file doesn't just store building locations and types it also includes information on; the number of agents in the model, the number of counts the model should run for, the rules on social distancing and percentages for; different diseases, race, and age groups.

Scenario Import essentially just reads the information from the file and put it into to format that model would be able to use. e.g. the adjacency matrix for the transport network. The entire scenario system has quite a lot of code and lot's of it isn't really that important to the development of the model, a link to the code can be found in appendices.

The school / work system: Every agent within the model during scenario import is assigned a role in the model depending on their age group. A agent that is assigned to the 5 - 17 age category will either be assigned to a primary school on secondary school depending on the number of places left at the schools. The same will happen with people who are eligible to work, they will be assigned a job at a public building and then generic work buildings. The agents then have the the location of that building set as their work location. At a certain time during the day all the agents will make their way to their work building via the public transport network.

The public transport system relies on adjacency matrix that is set up within scenario import and the A* algorithm. When an agent goes to a place it will first work out it if can walk to the destination. It does this shown by the code section below. Get max walk is a function that gets a variable that defines how far an agent can walk.

However as shown by the code, found in **Appendices -> Agent - Go to place function**, if the actor is not in max walk distance from the location it will use the A* star algorithm to work out the quickest route via the public transport network to the destination. If an agent can walk to their location they will use a weighted random walk. This means that whilst the agent won't directly make it's way to the destination it will move in the general direction to the destination location. The weights are determined by the distance to the destination. By having the x and y weights depend on the distance of to the destination in terms of x and y the agent will move in all directions towards the location until it either has the x or y coordinate correct, then it will only move the axis that it does not have the correct coordinate for. e.g. a agent moves from (0,0) to (5,5) it moves randomly to (2,5) it will now only move along the x axis.

In the previous paragraph it stated that the agent will use the A* algorithm to move through the public transport network but what is the A* algorithm. The A* algorithm is a addition to the Dijkstra path finding algorithm, A* star adds a heuristic which is the distance to the end point. In COVIDSIM that heuristic changes depending on if you are on the same tile as your destination. If you are not on the same tile it uses the distance in terms of tiles but if you are on the same tile it will use the distances on that tile, see Figure 5 and 6.

In A* you have nodes, in this case a node represents a transport hub, multiple transport types may be in the same location e.g. buses and trains. Between each node we have a weight or the f cost, this represents the time it takes to get between two different nodes, the node also records what the previous node was. We also have a open list and a closed list, the open list represents all the possible nodes we can visit from out current and previous nodes. The closes list represents nodes we have visited and explored each path extending from them. The algorithm starts by finding the closest node to the agent on the tile that the agent is on, this it the start node, it does the same with the end node, finding the closest node to the destination. The algorithm will then append all the nodes directly next to the start node into the open list and choose the one which has the lowest g cost. The g cost is made up of the both the h cost and the f cost, the h cost is the distance to the end node. The algorithm will then put the start node into the closed list and set the chosen node to be the current node. The algorithm will then add all the nodes around the chosen node to the open list and chose the node with the lowest g cost, it then appends the current node to the closed list. It does this until it has reached the end node. At this point it goes back through the closed list starting with the end node and looks at the previous node and appends it to a vector it repeats this until you get back to the start node. This gives you the quickest route through the public transport system to the destination. The Full code for the A* algorithm can be found in appendices. It will use the weighted random walk to get between it's start location and the start node, also the end node and the end location.

```
double Actor::matrix_distance(node& start_node, node& end_node, Matrix<
    int> tile_matrix)
{
        std::pair<int, int> start_location = { 0,0 };
        int start_tile = std::get<2>(start_node.location);

        std::pair<int, int> end_location = { 0,0 };
        int end_tile = std::get<2>(end_node.location);

        int row = std::floor((double)start_tile / (double)tile_matrix.
            Get_num_col());
        int col = start_tile - (row * tile_matrix.Get_num_col());

        start_location = std::make_pair(row, col);

        row = std::floor((double)end_tile / (double)tile_matrix.
            Get_num_col());
        col = end_tile - (row * tile_matrix.Get_num_col());

        end_location = std::make_pair(row, col);

        double distance = std::sqrt(std::pow((start_location.first -
            end_location.first), 2) + std::pow((start_location.second -
            end_location.second), 2));
        return distance;
}
```

Figure 6: Different tiles

```
int Actor::get_distance(std::tuple<int, int, unsigned int> loc_start, std
    ::tuple<int, int, unsigned int> loc_end)
{
        int d_x = std::get<0>(loc_end) - std::get<0>(loc_start);
        int d_y = std::get<1>(loc_end) - std::get<1>(loc_start);

        double magnitude = std::sqrt(d_x*d_x + d_y*d_y);
        int distance = std::floor(magnitude);
        return distance;
}
```

Figure 7: Same tile

When it came to developing the system that would control the way the actors to behave I turned to a concept that is more commonly found within video games, notable examples would be Left 4 dead and Doom (2016) [11].However my implementation is more in line with the one found in Doom (2016) than Left 4 dead. This is because like Doom, my implementation has a limit on the amount of agents doing things at once. This limit is hard coded into the model and can be changed by the user if they want to.

The way the actors behave relies on a hierarchical finite state machine (HFSM). A HFSM is a state

machine of state machine. A state machine allows the agent to transition between actions depending on whether certain conditions are met, for example in COVIDSIM a Boolean that determines whether a agent has permission to carry out a task and that would act as a condition allowing the agent to transfer between a idle state to an active state. A hierarchical finite state machine follows these same principles however instead of transitioning between states it transitions between state machines e.g. the idle loop and the task loop.

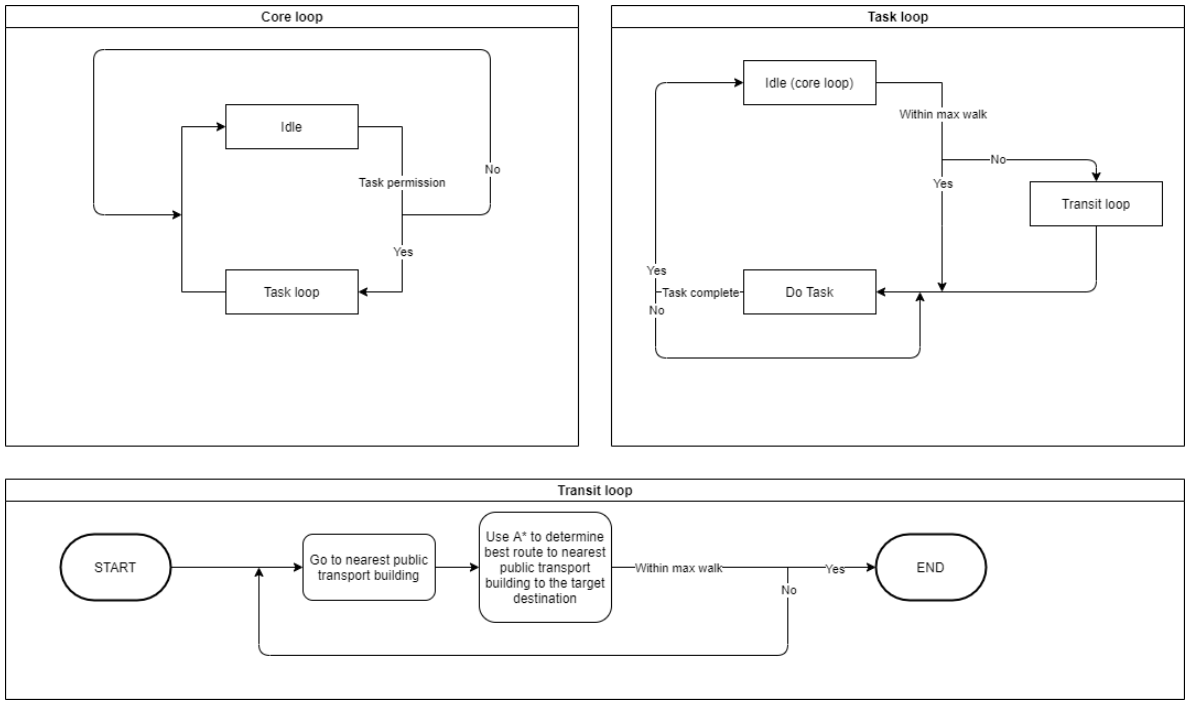


Figure 8: The hierarchical state machine for the agents

The figure above shows the HFSM that I made for the agents. The HFSM contains 3 main FSM's the: core loop, the task loop and the transit loop. The core loop is where most agents will spend there time within the model. Within the core loop is a simple Boolean function that determines whether the agent can do a task or not.

```
bool Actor::ask_director()
{

        std::vector<double> weights = { 1 - get_chance(), get_chance() };
        descision = Random::Discrete_distribution(weights, 1)[0];

        if (descision == 0)
        {
                return false;
        }
        if (descision == 1)
        {
                return true;
        }



bool Director::task_permission(Actor::State state)
{
        if (state == Actor::idle && m_current_tasks.size() <=
            max_actors_not_idle && sleep_active == false)
        {
                return true;
        }
        else
        {
                return false;
        }
}
```

Figure 9: The two functions that are used to determine if a agent can do a task

The first function found within the **actor** class, uses a random discrete distribution engine, determines if an agent is going to ask the director to do a task, this is so the agents further along in the vector that contains the all of the agents have an option to ask the director to do a task. The second function found within the director class, determines if the agent is allowed to do a task given the parameters set by the user and the model. The director assesses the state which the model is currently in and looks at three main parameters: The current state the agent is in, the agent can be in 4 different states: idle, waiting, in transit and doing a task; the number of agents who are currently: in transit, waiting and doing a task; and whether world sleep is active. This stops the model from taking extortionate amounts of time to complete a count, it also makes sure the model doesn't break the transport system as a actor could end up doing two tasks a once and thus never complete a single task and finally it makes sure people aren't running around at middle of the night going to the cinema. An exception to this system would be world tasks. World tasks enforce the entire model to do the same thing within 0 - 10 counts of each other, examples of world tasks would be: going home, going to work / school, going idle and going to sleep. Going to sleep and going idle happen at the same time due them being a simple state change rather than a set task.

A task is an object that contains a set of information on what a current agent is doing, a task contains: the building location, a pointer to the building, the type of building it e.g. is it a public building, how long the task will take and how long the agent has currently been doing that task for.

When a agent is given permission to do a task, the director will randomly choose a public building given two parameters, the agents age group and the time of day within the model. This will help the director choose which task is most suitable for that agent to at that time of day. The director will then go through the list of buildings of that type on the current tile that agent is on. If it can't find one or during random selection the number generated falls outside of the range of the vector it will move to look at the entire model. Also if the building chosen is closed due to staff being off with COVID the director will not allow the agent to go there. Once all the variables in the task object have been populated the task along with the agent that requested to do the task and any delay on the start of the task will be added to a vector so the director can deal with it later in the model.

Executing tasks, once any delay that the task had has passed, the director will follow the task loop shown in figure 7. The director begins the task loop by checking whether the agent is at the location or not, as shown in Figure 9. If not the director will use the code that I talked about school/work system. Once the agent arrives at their destination they are added to the building, this is important for infection loops, and the director carries out the do task function, at this point the agent stays in that building until their time doing that task equals the task length, at which point they get removed from the building, set to idle and can randomly walk around. There is a minimum time an agent has to wait before they can start a new task, this is a to mimic human behaviour and to also allow other agents to request to do a task. In hindsight it would make sense to generate a random number for which agent to ask to do a task and then remove that number from the list of possible numbers it can choose from. However using that system elsewhere in the task system I've noticed it can result in slow downs to the overall execution of that system.

```
if (agent->Get_Location() != std::get<0>(m_current_tasks[i])->location)
{
        agent->go_to_place(std::get<0>(m_current_tasks[i])->location,
            m_transport_net, m_tile_matrix);
        if (agent->A_star_found == false)
        {
                delete std::get<0>(m_current_tasks[i]);
                m_current_tasks.erase(m_current_tasks.begin() + i);
                log_tasks.LogFucntion(Log::LogLevelWarning, 4);
                failed++;
                continue;
        }
        continue;
}
```

Figure 10: If statement to determine if a agent is already at their target destination and if they can actually get to it

But enough about the systems included within the task and actor classes, on to the thing that really makes this a transmission model, the infection system. The infection system has 3 major systems: outdoors infection, transport infection and building infection.

With outdoors infection the model goes through the list of people are infected and are currently outdoors. The model then constructs a circle of a radius specified within the model and all the coordinates are floored due to agents not using float coordinates but integer coordinates.

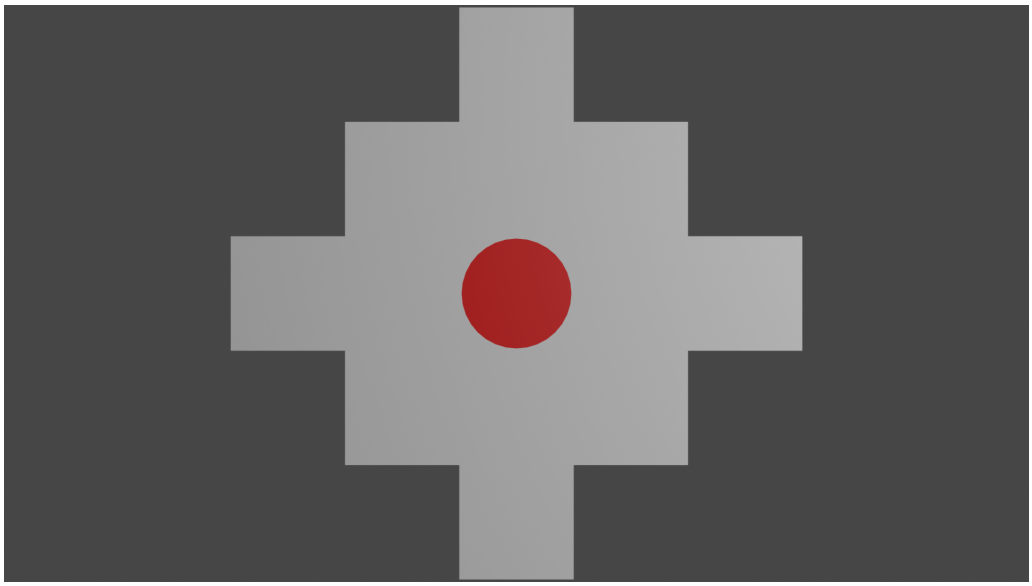


Figure 11: Infection circle

Any agents that exist within this circle on infection then have a chance to become infected based on their chance of being infected, this is determined by how invective the person they come into contact with this is determined by their symptom severity.

The second system is used to see if people are infected whilst on public transport. The system used here calculates the amount of people who are currently infected with COVID 19 and generates a value to modify the chance of being infected by. In this case I decided to use $\log_{100} + 1$, this means that if 100 agents are infectious the chance of being infected would double, at 1000 it would triple and so on. It then runs a infection check on every susceptible person on the transport network using the infection value generated. Limitations of this method would be that it's incredibly crude, i.e all the infectious people could be on a bus on the other side of the network but somebody on the metro could be infected. This is due to a lack of information about where and how the agent is moving around the environment with building location and current location as x, y and tile number being the only information. However it could also be argued that this method shows the effects that the lack of constantly cleaning public transport would have.

The final system relies on the same framework that is used transport system, except this system is used in buildings so instead of using $\text{Log}_{100} + 1$ it uses $\text{Log}_{10} + 1$. This is done to simulate that you are more likely to run into an infected person due to the person being in the same building as you. This system is slightly modified for hospitals due to some of the infected people within hospitals will be there as patients this means that precautions to make sure people don't come into contact with them, this means they do not add so much to the infection modifier compared to somebody who is infected but not a patient.

At this point though it could still be a transmission model to model the spread of any disease. What makes this specifically a transmission model to model the spread of SARS - nCov - 2, The infection constants. How I obtain the infection constants will be discussed in the next section, but essentially I get them by using a technique known as Logistic regression [12] and by obtaining the $\log_e$ of the odds when compared to the control group I can get a coefficient that would be used within the model. When combining multiple factors, e.g.: age, medical conditions and ethnic group, I can use $e^{\text{age coefficient + medical coefficient + ethnic group coefficient}}$ to obtain a value for the odds of

going to hospital, against the control group.

3.3 Advantages and disadvantages

On the other hand when comparing this to the system of differential equations model, the developing the ABM only has disadvantages as the advantages of this type of model come from its output data. The model takes a considerable amount of time to produce as overall development time for the ABM was around 6 months, this is partially due this being my first project of this magnitude in C++ and due to the shear amount of features I wanted to put in my model. This type of model is also very overwhelming as the code base got bigger there comes a point where the code becomes hard to maintain. This can be for many reasons; I don't understand the underlying technology behind the code that I'm writing, I wrote poor code earlier on the model and new sections rely on those sections to function meaning poor code creates more poor code, lack of commenting, the amount of features. This type of model is also quite computationally intensive and required a fair bit of optimization, to keep memory usage down and to decrease the time it takes to complete a count. Memory management was further made more difficult due to lack of **garbage collection** with heap allocated objects in C++, though keeping on top of memory usage was quite easy. However due to the limitations of the system I am running the model on I decided to try and keep memory usage at a minimum this lead to issues as shown in the transport infection system. Also CPU optimization was a bit of a hard one to get right. I ended up using multi threading to optimize my code, whilst this does speed up my code a lot as it allows multiple functions to run at once it also added more code to the model as multiple threads can't edit the same item in memory at the same time meaning I had to add code to the model that allow me to have the same effect however using a different technique.

3.4 Conclusions

From a purely development standpoint the system of differential equations model has only benefits and the ABM has only negatives. This is because unlike the ABM the differential equations model is: easy to debug due it's small code base, relatively easy and quick to produce at a total development time of 30 minutes, very fast so doesn't require any optimization and memory management isn't an issue due to only using a couple of variables compared to thousands of objects. So from this perspective the best one to develop is the system of differential equations model.

4 Data collection and analysis

4.1 Collection and analysis

The data collected is a mixture of data from the World Health organisation and the American CDC. Most of the data collected that was collected from the CDC was to enable the use of logistic regression to allow agents to have multiple factors that will increase their risk of catching COVID / going to hospital due to COVID. The data collected from the CDC is a mixture of data available to the public and restricted data. The restricted data differs from the public data by including more data that is critical to using the logistic regression model. Unlike the publicly available data the restricted data contains information on everyone who has a positive test for COVID 19, this ranged from the symptoms that the patient, their ethnicity, any prior medical conditions and their age category. The information having this level of detail allows for a complex control group to be created, increasing the number of risk factors that can be present within the model which will result in the transmission model producing data that is closer to real world data. The public data contains a more generalised data on those that have been hospitalized with COVID 19.

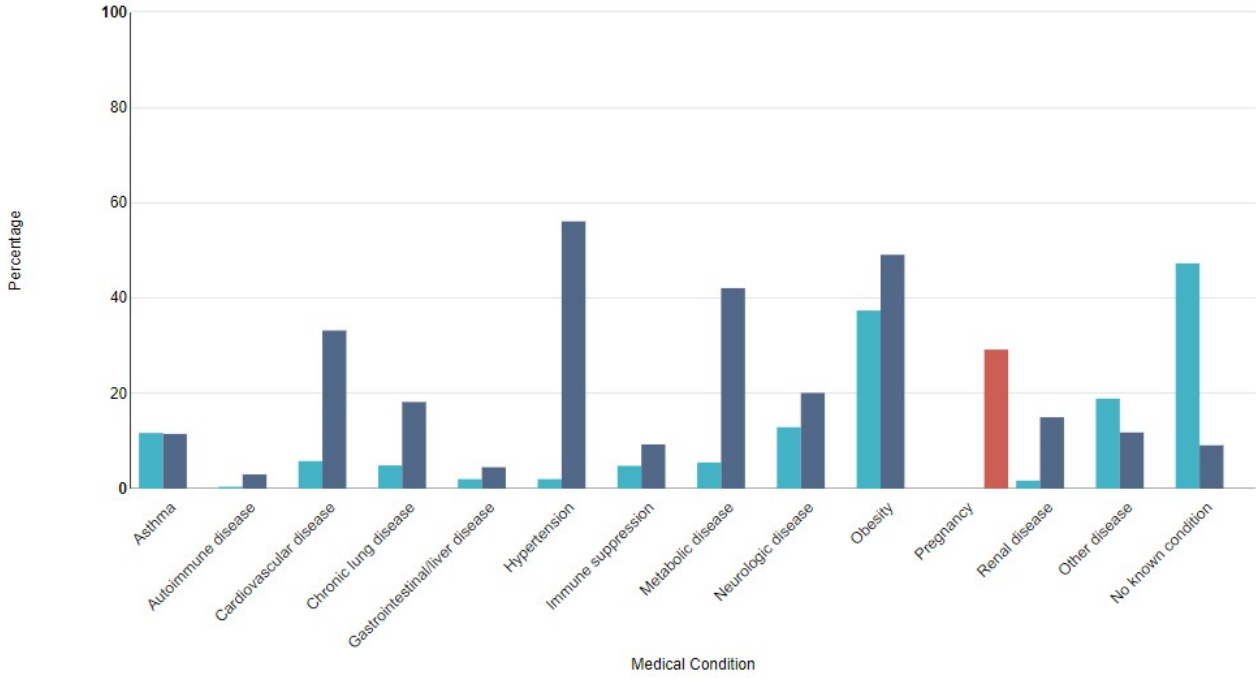


Figure 12: Public medical data

The main use of this data is that it has information on the proportion of hospitalized individuals who have a certain medical condition. for example if 37% of people have asthma this tells you that 37% of all hospitalized individuals who had a prior medical condition on the restricted date set also had asthma. However due to limitations with the data provided there is no values for the percentage of individuals who were not hospitalized who had asthma. This has resulted in the medical conditions being removed from the model and being replaced with just has a medical condition and does not have a medical condition. The public data set unlike the restricted data set is updated every week and unlike the restricted data the public data with the medical data being average that gets updated

every week as more data is added.

The restricted data was passed through a R script and by using the standard control group set by the CDC: Caucasian, no prior medical conditions and in the 18-49 age category. The script obtained the natural log of the odds compared to the odds of the control group. For example if the odds of going to hospital when you have a medical condition is 20 and the control group odds are 0.2.

$$\text{Log of the odds} = \ln \frac{20}{0.2} = \ln 100 = 4.605 \tag{1}$$

This means that in our example, you would be 100 times more likely to go to hospital if you had a prior medical condition.
When calculating the odds of somebody going to hospital you look at all the risk factors that differ to the control group.

$$\text{odds} = e^{(\text{ ethnic + medical + age group })} \times \text{control group odds} \tag{2}$$

In the equation above if the there is no difference in any of the risk factors the value for that risk factor is replaced with zero. For example if the individual has a different ethnic background and has a medical condition but are from the same age category, the age group value would be replaced with 0.

5 Results and conclusion

5.1 Results

In theory if both the models work as intended the data and graphs produced by each model should be similar however the graphs produced by the ABM will differ slightly from the differential equations model due to a ABM being able to pick up the minor details in the way a disease is spread.
Between both models I did try to keep constants the same, however it was found that some constants if kept constant between models would lead to either no infection at all or a infection with a R0 of 200 in the ABM, if that shows anything it serves as a testament for how sensitive the ABM is to a small change in certain values.

5.1.1 System of differential equations

The parameters used for this iteration of the model:

- Chance of infection = 0.01
- Chance of recovery = 0.005
- Starting population = 50000
- Starting infected = 1
- Starting removed = 0

As expected figure 13 shows how the disease spreads on a macro level, and between runs using different parameters the shapes of the curve don't differ much from the graph in figure 13. The gradient of the infected curve will always begin to decrease as the gradient of the removed curve gets bigger than the 0. The infected curve will peak when the chance of recovery is equal to the chance of becoming infected multiplied by the number of susceptible individuals.

$$\frac{dI}{dt} = KS(t)I(t) - \lambda I(t) \tag{1}$$

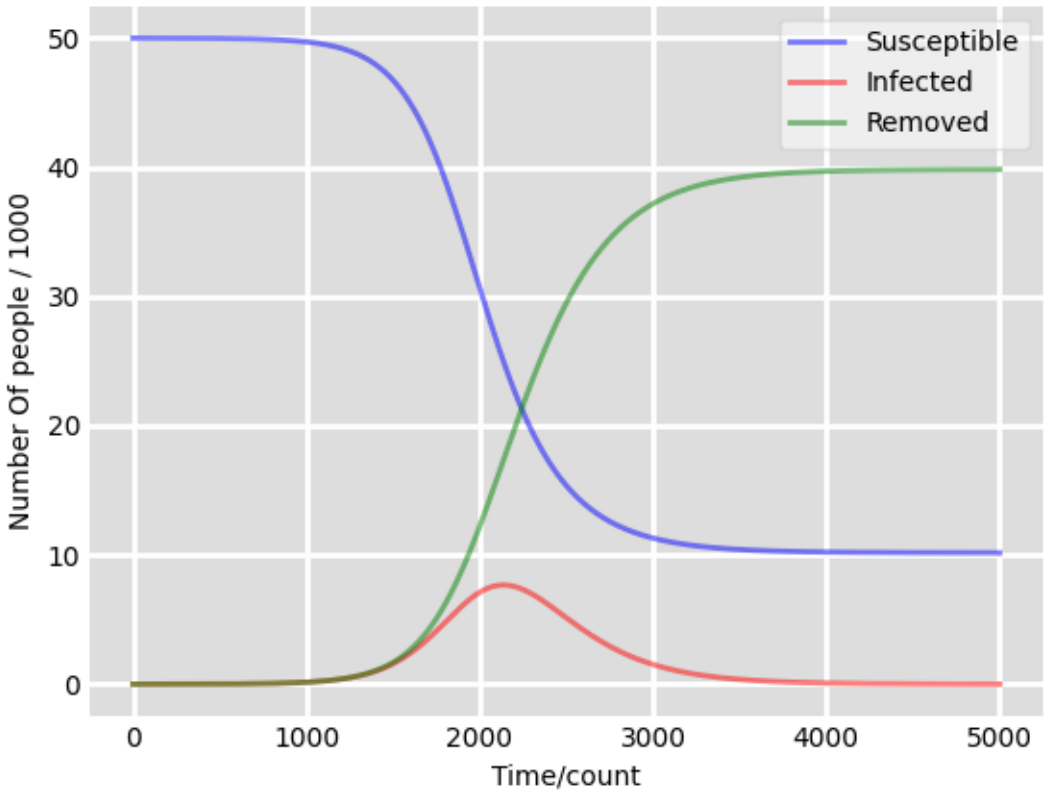


Figure 13: System of differential equations model

$$0 = KS(t)I(t) - \lambda I(t) \tag{2}$$

$$\lambda I(t) = KS(t)I(t) \tag{3}$$

$$\lambda = KS(t) \tag{4}$$

The only effect that changing parameters will have on the shape of the graphs is that: increasing chance of infection increases the gradient of the infected and susceptible graph, increasing the chance of recovery will decrease the gradient of infected graph and increase the gradient of the removed graph and increasing the values of the starting population/ infected/ removed will reduce the time it takes for the graphs to plateau.

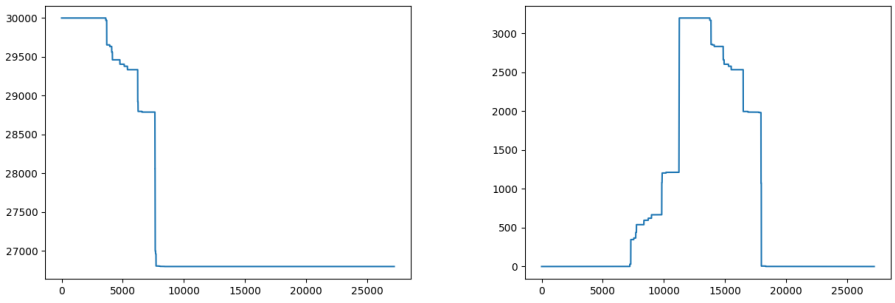
5.1.2 Agent based model

Constants used can be found in section 7.8 of the report.
Unfoutunatly CovidSim did not work as intended and this was due to a combination of issues that resulted in the infection of individuals not happening unless that individual originated from the same tile as the infected individual. I would presume that this has something to do with most people working in some kind of generic workplace for most of the day, this workplace is most likely on the same tile that they live on, and in a model with no concepts of weekends this means agent's rarely leave their own tile, and if they do it's very unlikely that they would infect someone due to the relatively low

chance of infection and the required time period someone has to be in direct contact with a infected individual to catch Covid - 19. e.g. if the chance of infection is 0.1 and the time period is 2 counts the susceptible individual has a 0.01 chance of being infected and that is assuming that they stay infected circle for multiple counts and succeed on both of the infection checks.

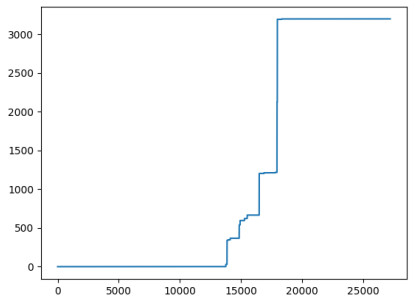
There are multiple ways to fix this, notably to remove the concept of tiles from the scenario system, whilst this would fix the problem in the short term a re wright of the infection system would probably be a better idea. A second addition to the model would to add the concept of weekends, this would be done by adding a 2 day break every 7200 counts, as a count represents 1 minute, this would hopefully encourage more agents to leave their own tiles and investigate the facilities on other tiles.

Whilst CovidSim did not work as intended, it did still produce a some data that follows the same pattern to the one showed in the differential equations model.



(a) number of susceptible individuals

(b) number of infected individuals



(c) number of removed individuals

Figure 14: The graphs shown here are from a version of COVIDSIM that isn't fully functional

As stated in section 5.1 the ABM is highly dependent on it's input values you can see in section 7.8 that the ABM relies on many constant's some of which have been found empirically, others that were assumptions and other ones that were put in there to help CovidSim not overwhelm my system. The constants that were found to have the most effect on the spread of disease where:

- The latent time, a lower latent time meant the infected individual become infectious quicker and in the case where latent time was set to 0 resulted in R0 of nearly 250. This is due to the way outdoors infection works, in that when you are infectious you cast a circle of radius 2 around you and perform a infection check on everyone around you. With a latent period of 0 counts this meant that the circle would just get bigger and bigger until it took up the entire tile.

- Infection probability, by increasing the infection probability the rate at which people become infected increases resulting in a faster exponential growth of the disease.
- Recovery probability, due to this being a stochastic model increasing the chance of recovering has varying effects, it would often results in the infection dying out quicker sometimes the infected individual would recover before becoming infectious. However it would sometimes also have no effect on the how the infection spread, due to it not being a fixed number of people who recover each turn rather a probability of recovering.
- Changing the time people have to be "infected", before becoming infected within the model will also effect the rate at which the number of infected individual at a exponential rate. For example if we take the chance of infection to be 0.5, and the time required to be x.

  $$\text{Chance of infection} = 0.5^x \tag{1}$$

  Decreasing this value greatly increases the infection rate, and increasing it greatly reduces the infection rate.

5.2 conclusion

With both the ABM and differential equations model serving very different purposes within modelling the spread of a disease. As the differential equations model shows the general shape of the curve whilst the ABM shows a more detailed curve that can highlight specific things and shows more detail on how the infection spreads and how it will affect a society due the model being able to output more data on the environment than a differential equations model. Just off the results that each model produces it would be hard to make a firm a decision on which is the better model to use as they both serve for different cases.

6 Final conclusion

Looking at each section of this report it is clear to see that no model is perfect and each model does something better than the other model. What the differential model gains in time to build and run, it looses in the detail of the results it produces, and it's lack of showing nuances in the how that disease spread. Likewise what the ABM gains in detail of the results it produces it lacks in time to build and run. The issue with looking at the two different models and comparing them is that they each exist to serve separate purposes as stated in the previous section, and so comparing them makes no sense as both are used in conjunction with each other to come to a unified decision about which approach makes more sense to combat a disease, and in some cases a ABM just wouldn't be viable as ABM's tend to use large amounts of computational power and requires a lot of time to complete it's task, this means if you were simulating something like a couple of years it would make much more sense to look at the differential equations model as the minor details wouldn't be what you were looking for, instead you would be looking at larger more macro effects. If you were looking at the effects something had over the course of a couple of months then you would look at using a ABM as you would be looking for the smaller details that wouldn't be picked up by the other model. So which one the best one to use, well it depends on your use case.

7 Appendices

7.1 System of differential equations model

PYTHON

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.integrate import odeint

def Find_Deriv(y, RunTime, N, alpha, beta):
    S, I, R = y
    dsdt = -alpha * S * I/N
    didt = (alpha * S * I/N) - (beta * I)
    drdt = beta * I
    return dsdt, didt, drdt

def main():
    N = int(input("Total Population: ")) #N = TotalPopulation
    I_0 = int(input("Total Number of Infected at time = 0: "))
    R_0 = int(input("Total Number of Removed at time = 0: "))
    S_0 = N - I_0 - R_0

    alpha = float(input("Chance of Infection: ")) #Chance of Infection
    beta = float(input("Fraction of People who recover per day: ")) #
        Fraction Of people who recover per count

    MaxTime = int(input("Time for which the model should run: "))
    RunTime = np.linspace(0, MaxTime, MaxTime)

    y_0 = S_0, I_0, R_0
    coord = odeint(Find_Deriv, y_0, RunTime, args = (N, alpha, beta))
    S, I, R = coord.T

    fig = plt.figure(facecolor ='w')
    ax = fig.add_subplot(111, facecolor = '#dddddd', axisbelow = True)
    ax.plot(RunTime, S/1000, 'b', alpha = 0.5, lw = 2, label = "
        Susceptible")
    ax.plot(RunTime, I/1000, 'r', alpha = 0.5, lw = 2, label = "Infected"
        )
    ax.plot(RunTime, R/1000, 'g', alpha = 0.5, lw = 2, label = "Removed")

    ax.set_xlabel('Time/count')
    ax.set_ylabel('Number Of people / 1000')
    ax.yaxis.set_tick_params(length = 0)
    ax.xaxis.set_tick_params(length = 0)
    ax.grid(b=True, which = 'major', c= 'w', lw=2, ls='-')
    legend = ax.legend()
    legend.get_frame().set_alpha(0.5)
    for x in ('top', 'right', 'bottom', 'left'):
        ax.spines[x].set_visible(False)

    plt.show()
```

```
main()
#Most of this code was written by https://scipython.com/book/chapter-8-
    scipy/additional-examples/the-sir-epidemic-model/ and then adapted by
    me
```

7.2 Logging system

C++

```
#pragma once

#include <iostream>
#include <fstream>
#include <time.h>
#include <chrono>
#include <thread>

#include <string>
#include <map>
#include <tuple>


class Log
{
public:
        enum LogLevel
        {
                LogLevelInfo = 0, LogLevelWarning, LogLevelError,
                    LogLevelCriticalError
        };
private:
        LogLevel m_CurrentLevel;

        std::map<unsigned int, std::string> Info{ {1 ,"FILE FOUND
            CORRECTLY"}, {2, "FILE ALREADY EXISTS"}, {3, "FILE READ
            SUCCESSFULLY"}, {4, "WORLD BUILT SUCCESSFULLY"}, {5, "MODEL
            COMPLETE"} };
        std::map<unsigned int, std::string> Warning{ {1 ,"INCORRECT INPUT
            "}, {2, "MORE STUDENTS THAN PLACES AT SCHOOL MIGHT CAUSE
            ISSUES"}, {3, "NO PATH FOUND"}, {4, "CANNOT COMPLETE TASK"} };
        std::map<unsigned int, std::string > Error{ {1 ,"WRONG FILE
            EXTENSION"}, {2, "Error message 2"}, {3, "WORLD BUILT
            UNSUCCESSFULLY"} };
        std::map<unsigned int, std::string> CriticalError{ {1 ,"FILE NOT
            FOUND"}, {2, "INCORRECT DATA INPUT / NO DATA PRESENT"}, {3, "
            END OF FILE ERROR"}, {4, "FILE READ UNSUCCESSFULLY"} };

        std::map<std::pair<LogLevel, int>, std::string> Recorded;

public:
        Log(LogLevel level);
```

```
        ~Log();

        void LogFucntion(LogLevel level, unsigned int ErrorNumber);
        void NewRun();
        void Clear();

        std::string GetCurrentTime();



};

#include "Logging API.h"

Log::Log(LogLevel level)
        :m_CurrentLevel(level)
{
        Recorded.clear();
}

Log::~Log()
{
        if (Recorded.size() != 0)
        {
                std::fstream LogFile;
                LogFile.open("Logfile.txt", std::ios::out | std::ios::app
                    );
                LogFile << GetCurrentTime() << std::endl;

                if (LogFile.is_open())
                {
                        for (auto elem : Recorded)
                        {
                                switch (elem.first.first)
                                {
                                case 0:
                                        LogFile << "[INFO]" << " " << "["
                                            << elem.first.second << "] "
                                           << elem.second << std::endl;
                                        break;
                                case 1:
                                        LogFile << "[WARNING]" << " " <<
                                            "[" << elem.first.second << "]
                                            " << elem.second << std::endl
                                            ;
                                        break;
                                case 2:
                                        LogFile << "[ERROR]" << " " << "[
                                            " << elem.first.second << "] "
                                             << elem.second << std::endl;
                                        break;
                                case 3:
```

```
                                        LogFile << "[CRTICAL ERROR]" << "
                                            " << "[" << elem.first.second
                                             << "] " << elem.second << std
                                            ::endl;
                                        break;
                                default:
                                        LogFile << "[CRITICAL ERROR]" <<
                                            "Seems we have a bit of an
                                            error here, don't worry it
                                            will be fixed soon" << std::
                                            endl;
                                        break;
                                }
                        }
                }
                LogFile.close();
        }
}

void Log::LogFucntion(LogLevel level, unsigned int ErrorNumber)
{
        m_CurrentLevel = level;

        std::pair <LogLevel, unsigned int> ErrorType;
        ErrorType.first = level;
        ErrorType.second = ErrorNumber;

        if (m_CurrentLevel == 0)
        {
                std::map<unsigned int, std::string>::iterator temp = Info
                    .find(ErrorNumber);
                Recorded.insert(std::make_pair(ErrorType, temp->second));
        }
        else if (m_CurrentLevel == 1)
        {
                std::map<unsigned int, std::string>::iterator temp =
                    Warning.find(ErrorNumber);
                Recorded.insert(std::make_pair(ErrorType, temp->second));
        }
        else if (m_CurrentLevel == 2)
        {
                std::map<unsigned int, std::string>::iterator temp =
                    Error.find(ErrorNumber);
                Recorded.insert(std::make_pair(ErrorType, temp->second));
        }
        else if (m_CurrentLevel == 3)
        {
                std::map<unsigned int, std::string>::iterator temp =
                    CriticalError.find(ErrorNumber);
                Recorded.insert(std::make_pair(ErrorType, temp->second));
        }
}
```

```
void Log::Clear()
{
        std::ofstream LogFile;
        LogFile.open("Logfile.txt", std::ios::out | std::ios::trunc);
        LogFile.close();
}

void Log::NewRun()
{
        std::fstream LogFile;
        LogFile.open("Logfile.txt", std::ios::out | std::ios::app);

        LogFile << "----------NEW_RUN----------" << std::endl;

        LogFile.close();
}

std::string Log::GetCurrentTime()
{
        time_t CT = time(NULL); //CT = CurrentTime
        char str[26];
        ctime_s(str, sizeof str, &CT);
        return str;
}
```

7.3 Scenario system

C++

This system is over 2000 lines of code and so it's too much to add the model. It can be found in the COVIDSIM repository: **https://github.com/bptigg/CovidSim**.
Files: Scenario.cpp, Scenario.h

7.4 Agent - Go to place function

C++

```
bool Actor::go_to_place(std::tuple<int, int, int> destination,
    Transport_Net* network, Matrix<int> tile_matrix)
{
        Destination = destination;
        if (location == outside && get_distance(this->Get_Location(),
            destination) <= get_max_walk() && std::get<2>(this->
            Get_Location()) == std::get<2>(destination))
        {
                A_star_found = true;
                location = Actor::outside;
                if (weighted_walk({ std::get<0>(destination), std::get
                    <1>(destination) }) == true)
                {
                        return true;
```

```
                }
        }
        else
        {
                if (m_a_star_path == false)
                {

                        A_star_found = A_star_algorithim(network,
                            tile_matrix);
                        if (A_star_found == false)
                        {
                                return false;
                        }
```

7.5 A* algorithm

C++

```
bool Actor::A_star_algorithim(Transport_Net* network, Matrix<int>
    tile_matrix)
{
        a_star_path.clear();
        Log a_star_log(Log::LogLevelInfo);

        int nearest_node = get_nearest_node(network, this->Get_Location()
            );

        if (nearest_node == 0)
        {
                a_star_log.LogFucntion(Log::LogLevelWarning, 3);
                return false;
        }

        node start_node;

        int b = 0;
        for (auto elem : network->public_transport_network)
        {
                if (elem.first == nearest_node)
                {
                        start_node.g_cost = 0;
                        start_node.node_num = elem.first;
                        start_node.location = elem.second.first;
                        break;
                }

                b++;

                if (b == network->public_transport_network.size())
                {
                        a_star_log.LogFucntion(Log::LogLevelWarning, 3);
                        return false;
```

```
        }

}

b = 0;
int nearest_to_end = get_nearest_node(network, Destination);

if (nearest_to_end == 0)
{
        a_star_log.LogFucntion(Log::LogLevelWarning, 3);
        return false;
}

node end_node;
for (auto elem : network->public_transport_network)
{
        if (elem.first == nearest_to_end)
        {
                end_node.location = elem.second.first;
                end_node.node_num = elem.first;
                break;
        }

        b++;

        if (b == network->public_transport_network.size())
        {
                a_star_log.LogFucntion(Log::LogLevelWarning, 3);
                return false;
        }

}

bool tile_heuristic = false;
if (std::get<2>(start_node.location) == std::get<2>(end_node.
    location))
{
        tile_heuristic = true;
}

if (tile_heuristic == true)
{
        start_node.h_cost = std::floor(get_distance(start_node.
            location, end_node.location));
}
else
{
        start_node.h_cost = std::floor(matrix_distance(start_node
            , end_node, tile_matrix));
}

start_node.previous_node_num = NULL;
```

```
start_node.f_cost = start_node.h_cost;

bool path_found = false;

std::vector<std::pair<int, node*>> open_list;
std::vector<node*> closed_list;

int run = 0;

node* current_node = new node;
current_node = &start_node;
open_list.push_back(std::make_pair(current_node->f_cost, &
    start_node));

while (path_found == false)
{
        std::vector<int> col_weights = network->TransportNetwork
            ->GetCol(current_node->node_num);
        for (int i = 0; i < network->TransportNetwork->GetCol(
            current_node->node_num).size(); i++)
        {
                bool in_closed_list = false;
                if (i == current_node->node_num - 1)
                {
                        continue;
                }
                else
                {
                        node* graph_node = new node;
                        graph_node->g_cost = col_weights[i];

                        for (auto elem : network->
                            public_transport_network)
                        {
                                if (elem.first == i+1)
                                {
                                        graph_node->location =
                                            elem.second.first;
                                        graph_node->
                                            previous_node_num =
                                            current_node->node_num
                                            ;
                                        graph_node->node_num = i
                                            + 1;

                                        if (tile_heuristic ==
                                            true)
                                        {
                                                graph_node->
                                                    h_cost =
                                                    get_distance(
                                                    graph_node->
```

```
                                location,
                                end_node.
                                location);
                        }
                        else
                        {
                                graph_node->
                                h_cost =
                                matrix_distance
                                (*graph_node,
                                end_node,
                                tile_matrix);
                        }

                        break;
                }
        }
        graph_node->f_cost = graph_node->g_cost +
            graph_node->h_cost;

        for (int a = 0; a < open_list.size(); a
            ++)
        {
                if (graph_node->location ==
                    open_list[a].second->location)
                {
                        if (graph_node->f_cost <=
                            open_list[a].first)
                        {
                                open_list[a].
                                first =
                                graph_node->
                                f_cost +
                                open_list[0].
                                first;
                                open_list[a].
                                second->f_cost
                                 = graph_node
                                ->f_cost +
                                open_list[0].
                                first;
                                open_list[a].
                                second->
                                previous_node_num
                                 = graph_node
                                ->
                                previous_node_num
                                ;
                        }
                }
        }
```

```
                for (int a = 0; a< closed_list.size(); a
                    ++)
                {
                        if (graph_node->location ==
                            closed_list[a]->location)
                        {
                                in_closed_list = true;
                        }
                }

                if (graph_node->g_cost != 0 &&
                    in_closed_list == false)
                {
                        open_list.push_back(std::
                            make_pair(graph_node->f_cost,
                            graph_node));
                }
                else
                {
                        delete graph_node;
                }
        }
}
std::sort(open_list.begin(), open_list.end());
closed_list.push_back(current_node);

if (open_list[0].second != current_node)
{
        current_node = open_list[0].second;

        int i = 0;
        for (i < open_list.size(); i++;)
        {
                if (open_list[i].second == current_node)
                {
                        break;
                }
        }
        open_list.erase(open_list.begin() + i);
}
else
{
        open_list.erase(open_list.begin());
        if (open_list.size() == 0)
        {
                a_star_log.LogFucntion(Log::
                    LogLevelWarning, 3);
                return false;
        }
        current_node = open_list[0].second;
}
```

```
        if (closed_list[run]->location == end_node.location)
        {
                path_found = true;
                break;
        }

        if (open_list.size() == 0)
        {
                a_star_log.LogFucntion(Log::LogLevelWarning, 3);
                return false;
        }

        run++;
}

int node_num = closed_list.back()->node_num;

path_found = false;

while (path_found == false)
{
        for (int i = 0; i < closed_list.size(); i++)
        {
                if (closed_list[i]->node_num == node_num)
                {
                        a_star_path.push_back({node_num,
                            closed_list[i]->f_cost, Vehicle::
                            default_vehicle });
                        node_num = closed_list[i]->
                            previous_node_num;

                        if (closed_list[i]->node_num ==
                            start_node.node_num)
                        {
                                path_found = true;
                        }

                        break;
                }
        }
}

for (int i = 0; i < closed_list.size(); i++)
{
        //delete closed_list[i];
}
closed_list.clear();

m_a_star_path = true;
return true;
```

```
        //I need to fix this but don't quite know how prob use a if
            statement
}
```

7.6 Task allocation system

C++

Like the Scenario system the task system is also too long to add to this report. It can be found within the COVIDSIM repository: **https://github.com/bptigg/CovidSim**.
Files: Director.h, Director.cpp, Actor.h, Actor.cpp, Tasks.h, Model.cpp

7.7 R script

R

```
setwd("~/EMS/EMC/EMC_Y13/Restricted_Data/Restricted_data")

data_1.control_group = c("18_-_49_Years","White,_Non-Hispanic","No")

#31-07-2020
analyse_csv = function(data_1.control_group, csv_file){
  data_1 = read.csv(csv_file)
  data_1.control_group_size = data_1$hosp_yn[which(data_1$age_group ==
      data_1.control_group[1]& data_1$ ..race_ethnicity_combined == data_
      1.control_group[2] & data_1$medcond_yn == data_1.control_group[3])]
  data_1.control_group.hospital.yes = data_1$hosp_yn[which(data_1$hosp_yn
       == "Yes" & data_1$age_group == data_1.control_group[1] & data_1$ ..
      race_ethnicity_combined == data_1.control_group[2] & data_1$medcond_
      yn == data_1.control_group[3])]
  data_1.control_group.hospital.no = data_1$hosp_yn[which(data_1$hosp_yn
      == "No" & data_1$age_group == data_1.control_group[1] & data_1$ ..
      race_ethnicity_combined == data_1.control_group[2] & data_1$medcond_
      yn == data_1.control_group[3])]
  data_1.control_group.hospital.missing = data_1$hosp_yn[which((data_1$
      hosp_yn == "Missing" | data_1$hosp_yn == "Unknown") & data_1$age_
      group == data_1.control_group[1] & data_1$ ..race_ethnicity_combined
       == data_1.control_group[2] & data_1$medcond_yn == data_1.control_
      group[3])]

  data_1.control_group.odds = length(data_1.control_group.hospital.yes) /
       length(data_1.control_group.hospital.no)
  return (data_1.control_group.odds)
}

analyse_single_variable = function(variable, csv_file, other){
  data_2 = read.csv(csv_file)

  data_2.group.yes = c()
  data_2.group.no = c()

  if(other) {
```

```
    data_2.group.size = data_2$hosp_yn[which(data_2$ .. race_ethnicity_
        combined == variable)]
    data_2.group.yes = data_2$hosp_yn[which(data_2$hosp_yn == "Yes" &
        data_2$ .. race_ethnicity_combined == variable)]
    data_2.group.no = data_2$hosp_yn[which(data_2$hosp_yn == "No" & data_
        2$ .. race_ethnicity_combined == variable)]
  }
  else {
    data_2.ignore = c("Hispanic/Latino", "Black, Non-Hispanic", "Missing"
        , "NA", "Unknown", "White, Non-Hispanic")
    data_2.group.size = data_2$hosp_yn[which(data_2$ .. race_ethnicity_
        combined != data_2.ignore[1] & data_2$ .. race_ethnicity_combined !
        = data_2.ignore[2] & data_2$ .. race_ethnicity_combined != data_2.
        ignore[3] & data_2$ .. race_ethnicity_combined != data_2.ignore[4]
        & data_2$ .. race_ethnicity_combined != data_2.ignore[5] & data_2$
        .. race_ethnicity_combined != data_2.ignore[6])]
    data_2.group.yes = data_2$hosp_yn[which(data_2$hosp_yn == "Yes" &
        data_2$ .. race_ethnicity_combined != data_2.ignore[1] & data_2$ ..
        race_ethnicity_combined != data_2.ignore[2] & data_2$ .. race_
        ethnicity_combined != data_2.ignore[3] & data_2$ .. race_ethnicity_
        combined != data_2.ignore[4] & data_2$ .. race_ethnicity_combined !
        = data_2.ignore[5] & data_2$ .. race_ethnicity_combined != data_2.
        ignore[6])]
    data_2.group.no = data_2$hosp_yn[which(data_2$hosp_yn == "No" & data_
        2$ .. race_ethnicity_combined != data_2.ignore[1] & data_2$ .. race_
        ethnicity_combined != data_2.ignore[2] & data_2$ .. race_ethnicity_
        combined != data_2.ignore[3] & data_2$ .. race_ethnicity_combined !
        = data_2.ignore[4] & data_2$ .. race_ethnicity_combined != data_2.
        ignore[5] & data_2$ .. race_ethnicity_combined != data_2.ignore[6])
        ]
  }
  return(length(data_2.group.yes) / length(data_2.group.no))

}

analyse_single_variable_age = function(variable, csv_file, other){
  data_2 = read.csv(csv_file)

  data_2.group.yes = c()
  data_2.group.no = c()

  if(other) {
    data_2.group.size = data_2$hosp_yn[which(data_2$age_group == variable
        )]
    data_2.group.yes = data_2$hosp_yn[which(data_2$hosp_yn == "Yes" &
        data_2$age_group == variable)]
    data_2.group.no = data_2$hosp_yn[which(data_2$hosp_yn == "No" & data_
        2$age_group == variable)]
  }
  return(length(data_2.group.yes) / length(data_2.group.no))

}
```

```
analyse_single_variable_medical = function(variable, csv_file, other){
  data_2 = read.csv(csv_file)

  data_2.group.yes = c()
  data_2.group.no = c()

  if(other) {
    data_2.group.size = data_2$hosp_yn[which(data_2$medcond_yn ==
        variable)]
    data_2.group.yes = data_2$hosp_yn[which(data_2$hosp_yn == "Yes" &
        data_2$medcond_yn == variable)]
    data_2.group.no = data_2$hosp_yn[which(data_2$hosp_yn == "No" & data_
        2$medcond_yn == variable)]
  }
  return(length(data_2.group.yes) / length(data_2.group.no))
}

#
    ##############################################################################


data_first = analyse_csv(data_1.control_group, "31-07-2020.csv")
data_second = analyse_csv(data_1.control_group, "31-08-2020.csv")
data_third = analyse_csv(data_1.control_group, "30-09-2020.csv")
data_fourth = analyse_csv(data_1.control_group, "04-12-2020.csv")

odds = c(data_first, data_second, data_third, data_fourth)
control_group_hospital_odds = mean(odds)

#Race values

odds.other = analyse_single_variable("", "31-08-2020.csv", FALSE)
odds.Hispanic_Latino = analyse_single_variable("Hispanic/Latino", "
    31-08-2020.csv", TRUE)
odds.Black_non_hispanic = analyse_single_variable("Black, Non-Hispanic",
    "31-08-2020.csv", TRUE)
odds.White_non_hispanic = analyse_single_variable("White, Non-Hispanic",
    "31-08-2020.csv", TRUE)

odds_2.other = analyse_single_variable("", "31-07-2020.csv", FALSE)
odds_2.Hispanic_Latino = analyse_single_variable("Hispanic/Latino", "
    31-07-2020.csv", TRUE)
odds_2.Black_non_hispanic = analyse_single_variable("Black, Non-Hispanic"
    , "31-07-2020.csv", TRUE)
odds_2.White_non_hispanic = analyse_single_variable("White, Non-Hispanic"
    , "31-07-2020.csv", TRUE)

odds_3.other = analyse_single_variable("", "30-09-2020.csv", FALSE)
odds_3.Hispanic_Latino = analyse_single_variable("Hispanic/Latino", "
    30-09-2020.csv", TRUE)
```

```
odds_3.Black_non_hispanic = analyse_single_variable("Black, Non-Hispanic"
    , "30-09-2020.csv", TRUE)
odds_3.White_non_hispanic = analyse_single_variable("White, Non-Hispanic"
    , "30-09-2020.csv", TRUE)

odds_4.other = analyse_single_variable("", "04-12-2020.csv", FALSE)
odds_4.Hispanic_Latino = analyse_single_variable("Hispanic/Latino", "
    04-12-2020.csv", TRUE)
odds_4.Black_non_hispanic = analyse_single_variable("Black, Non-Hispanic"
    , "04-12-2020.csv", TRUE)
odds_4.White_non_hispanic = analyse_single_variable("White, Non-Hispanic"
    , "04-12-2020.csv", TRUE)

odd.other = c(odds.other, odds_2.other, odds_3.other, odds_4.other)
other.odds = mean(odd.other)
other.LG_value = log((other.odds / control_group_hospital_odds), base =
    exp(1))

odd.Hispanic_Latino = c(odds.Hispanic_Latino, odds_2.Hispanic_Latino,
    odds_3.Hispanic_Latino, odds_4.Hispanic_Latino)
Hispanic_Latino.odds = mean(odd.Hispanic_Latino)
Hispanic_Latino.LG_value = log((Hispanic_Latino.odds / control_group_
    hospital_odds), base = exp(1))

odd.Black_non_hispanic = c(odds.Black_non_hispanic, odds_2.Black_non_
    hispanic, odds_3.Black_non_hispanic, odds_4.Black_non_hispanic)
Black_non_hispanic.odds = mean(odd.Black_non_hispanic)
Black_non_hispanic.LG_value = log((Black_non_hispanic.odds / control_
    group_hospital_odds), base = exp(1))

odd.White_non_hispanic = c(odds.White_non_hispanic, odds_2.White_non_
    hispanic, odds_3.White_non_hispanic, odds_4.White_non_hispanic)
White_non_hispanic.odds = mean(odd.White_non_hispanic)
White_non_hispanic.LG_value = log((White_non_hispanic.odds / control_
    group_hospital_odds), base = exp(1))

#age groups

odds_age_1.zero_to_four = analyse_single_variable_age("0 - 4 Years", "
    31-08-2020.csv", TRUE)
odds_age_1.five_to_seventeen = analyse_single_variable_age("5 - 17 Years"
    , "31-08-2020.csv", TRUE )
odds_age_1.eighteen_to_fourtynine = analyse_single_variable_age("18 - 49
    Years", "31-08-2020.csv", TRUE )
odds_age_1.fifty_to_sixytfour = analyse_single_variable_age("50 - 64
    Years", "31-08-2020.csv", TRUE )
odds_age_1.sixty_five_plus = analyse_single_variable_age("65+ Years", "
    31-08-2020.csv", TRUE )

odds_age_2.zero_to_four = analyse_single_variable_age("0 - 4 Years", "
    31-07-2020.csv", TRUE)
```

```
odds_age_2.five_to_seventeen = analyse_single_variable_age("5 - 17 Years"
    , "31-07-2020.csv", TRUE )
odds_age_2.eighteen_to_fourtynine = analyse_single_variable_age("18 - 49
    Years", "31-07-2020.csv", TRUE )
odds_age_2.fifty_to_sixytfour = analyse_single_variable_age("50 - 64
    Years", "31-07-2020.csv", TRUE )
odds_age_2.sixty_five_plus = analyse_single_variable_age("65+ Years", "
    31-07-2020.csv", TRUE )

odds_age_3.zero_to_four = analyse_single_variable_age("0 - 4 Years", "
    30-09-2020.csv", TRUE)
odds_age_3.five_to_seventeen = analyse_single_variable_age("5 - 17 Years"
    , "30-09-2020.csv", TRUE )
odds_age_3.eighteen_to_fourtynine = analyse_single_variable_age("18 - 49
    Years", "30-09-2020.csv", TRUE )
odds_age_3.fifty_to_sixytfour = analyse_single_variable_age("50 - 64
    Years", "30-09-2020.csv", TRUE )
odds_age_3.sixty_five_plus = analyse_single_variable_age("65+ Years", "
    30-09-2020.csv", TRUE )

odds_age_4.zero_to_four = analyse_single_variable_age("0 - 4 Years", "
    04-12-2020.csv", TRUE)
odds_age_4.five_to_seventeen = analyse_single_variable_age("5 - 17 Years"
    , "04-12-2020.csv", TRUE )
odds_age_4.eighteen_to_fourtynine = analyse_single_variable_age("18 - 49
    Years", "04-12-2020.csv", TRUE )
odds_age_4.fifty_to_sixytfour = analyse_single_variable_age("50 - 64
    Years", "04-12-2020.csv", TRUE )
odds_age_4.sixty_five_plus = analyse_single_variable_age("65+ Years", "
    04-12-2020.csv", TRUE )

odd.zero_to_four = c(odds_age_1.zero_to_four, odds_age_2.zero_to_four,
    odds_age_3.zero_to_four, odds_age_4.zero_to_four)
zero_to_four.odds = mean(odd.zero_to_four)
zero_to_four.LG_value = log((zero_to_four.odds / control_group_hospital_
    odds), base = exp(1))

odd.five_to_seventeen = c(odds_age_1.five_to_seventeen, odds_age_2.five_
    to_seventeen, odds_age_3.five_to_seventeen, odds_age_4.five_to_
    seventeen)
five_to_seventeen.odds = mean(odd.five_to_seventeen)
five_to_seventeen.LG_value = log((five_to_seventeen.odds / control_group_
    hospital_odds), base = exp(1))

odd.eighteen_to_fourty_nine = c(odds_age_1.eighteen_to_fourtynine, odds_
    age_2.eighteen_to_fourtynine, odds_age_3.eighteen_to_fourtynine, odds_
    age_4.eighteen_to_fourtynine)
eighteen_to_fourty_nine.odds = mean(odd.eighteen_to_fourty_nine)
eighteen_to_fourty_nine.LG_value = log((eighteen_to_fourty_nine.odds /
    control_group_hospital_odds), base = exp(1))
```

```
odd.fifty_to_sixytfour = c(odds_age_1.fifty_to_sixytfour, odds_age_2.
    fifty_to_sixytfour, odds_age_3.fifty_to_sixytfour, odds_age_4.fifty_to
    _sixytfour)
fifty_to_sixytfour.odds = mean(odd.fifty_to_sixytfour)
fifty_to_sixytfour.LG_value = log((fifty_to_sixytfour.odds / control_
    group_hospital_odds), base = exp(1))

odd.sixty_five_plus = c(odds_age_1.sixty_five_plus, odds_age_2.sixty_five
    _plus, odds_age_3.sixty_five_plus, odds_age_4.sixty_five_plus)
sixty_five_plus.odds = mean(odd.sixty_five_plus)
sixty_five_plus.LG_value = log((sixty_five_plus.odds / control_group_
    hospital_odds), base = exp(1))

#medical conditons

odd_medical_1.yes = analyse_single_variable_medical("Yes", "31-08-2020.
    csv", TRUE)
odd_medical_1.No = analyse_single_variable_medical("No", "31-08-2020.csv"
    , TRUE)

odd_medical_2.yes = analyse_single_variable_medical("Yes", "31-07-2020.
    csv", TRUE)
odd_medical_2.No = analyse_single_variable_medical("No", "31-07-2020.csv"
    , TRUE)

odd_medical_3.yes = analyse_single_variable_medical("Yes", "30-09-2020.
    csv", TRUE)
odd_medical_3.No = analyse_single_variable_medical("No", "30-09-2020.csv"
    , TRUE)

odd_medical_4.yes = analyse_single_variable_medical("Yes", "04-12-2020.
    csv", TRUE)
odd_medical_4.No = analyse_single_variable_medical("No", "04-12-2020.csv"
    , TRUE)


odd.medical_yes = c(odd_medical_1.yes, odd_medical_2.yes, odd_medical_3.
    yes, odd_medical_4.yes)
medical_yes.odd = mean(odd.medical_yes)
medical_yes.LG_value =  log((medical_yes.odd / control_group_hospital_
    odds), base = exp(1))

odd.medical_no = c(odd_medical_1.No, odd_medical_2.No, odd_medical_3.No,
    odd_medical_4.No)
medical_no.odd = mean(odd.medical_no)
medical_no.LG_value =  log((medical_no.odd / control_group_hospital_odds)
    , base = exp(1))
```

7.8 Parameters used for ABM

```
#include "Constants.h"
```

```
uint32_t MAX_FAMILY_SIZE = 5;
uint32_t get_max_family_size()
{
        return MAX_FAMILY_SIZE;
}

uint32_t NUM_OF_HOUSEHOLDS_PER_BLOCK = 10;
uint32_t get_num_of_households_per_block()
{
        return NUM_OF_HOUSEHOLDS_PER_BLOCK;
}

uint32_t NUM_OF_OFFICES_PER_BLOCK = 2;
uint32_t get_num_of_offices_per_block()
{
        return NUM_OF_OFFICES_PER_BLOCK;
}

uint32_t MAX_OFFICE_SIZE = 150;
uint32_t get_max_office_size()
{
        return MAX_OFFICE_SIZE;
}


double PERCENTAGE_ASYMPTOMATIC = 25;
double get_percentage_asymptomatic()
{
        return PERCENTAGE_ASYMPTOMATIC;
}

double PERCENTAGE_MILD = 50;
double get_percentage_mild()
{
        return PERCENTAGE_MILD;
}

uint32_t INFECTION_DISTANCE = 2;
uint32_t get_infection_distance()
{
        return INFECTION_DISTANCE;
}

uint32_t STARTING_INFECTED = 1;
uint32_t get_starting_infected()
{
        return STARTING_INFECTED;
}

uint32_t LATENT_TIME = 3600; //3600
```

```
uint32_t get_latent_time()
{
        return LATENT_TIME;
}

double ASYMPTOMATIC = 0.75;
double MILD = 1;
double FULL = 1.25;
double get_modifier_value(int value)
{
        switch (value)
        {
        case 0:
                return ASYMPTOMATIC;
                break;
        case 1:
                return MILD;
                break;
        case 2:
                return FULL;
                break;
        default:
                return 1;
                break;
        }
}

uint32_t MAX_IDLE_TIME = 100; //counts are in minutes, This doesn't work
    if the actor is asleep
uint32_t get_max_idle_time()
{
        return MAX_IDLE_TIME;
}

uint32_t MIN_IDLE_TIME = 50;
uint32_t get_min_idle_time()
{
        return MIN_IDLE_TIME;
}

uint32_t COUNTS_PER_DAY = 86400;
uint32_t get_counts_per_day()
{
        return COUNTS_PER_DAY;
}

double TIME_MODIFIER = 0.1;
double get_time_modifier()
{
        return TIME_MODIFIER;
}
```

```
unsigned int MAX_WALK = 1;
unsigned int get_max_walk()
{
        return MAX_WALK;
}

double CHANCE = 0.15;
double get_chance()
{
        return CHANCE;
}

uint32_t MIN_AYSMPTOMATIC_PERIOD = 10080; //10080;
uint32_t get_min_period()
{
        return MIN_AYSMPTOMATIC_PERIOD;
}

uint32_t MAX_AYSMPTOMATIC_PERIOD = 20160; //20160;
uint32_t get_max_period()
{
        return MAX_AYSMPTOMATIC_PERIOD;
}

uint32_t DAY_LENGTH = 1440;
uint32_t get_day_length()
{
        return DAY_LENGTH;
}

std::vector<std::string> NAMES = { "Hugo", "Stephen", "Gage", "George", "
    Ryan", "Ben", "Arthur"};
std::vector <std::string> get_names()
{
        return NAMES;
}

//COVID VALUES (CURRENTLY TEMPORARY STAND IN VALUES)
int INFECTION_TIME = 2;
int get_infection_time()
{
        return INFECTION_TIME;
}

double INFECTIVITY = 0.2;
double get_infect()
{
        return INFECTIVITY;
}

double RECOVERY = 0.1;
double get_recover()
```

```
{
        return RECOVERY;
}

//hostiplization
double CONTROL = -3.731170026;

double Z_FO = 0.5163976;
double FI_SE = 0.08572764;
double EI_FN = 1.322797;
double FIF_SF = 2.62366;
double SF_PLUS = 3.752829;

double get_hospilization_age_co(int num)
{
        switch (num)
        {
        case 0:
                return Z_FO;
                break;
        case 1:
                return FI_SE;
                break;
        case 2:
                return CONTROL;
                break;
        case 3:
                return FIF_SF;
                break;
        case 4:
                return SF_PLUS;
                break;
        default:
                return CONTROL;
                break;
        }
        return CONTROL;
}

double med_con_y = 3.024842;
double med_con_n = 0.8206791;

//child medical

double get_medical()
{
        return med_con_y;
}

double get_ethnicity_co(int num)
{
        switch (num)
```

```
        {
        case 0:
                return 2.177051;
                break;
        case 1:
                return 2.567965;
                break;
        case 2:
                return 2.080266;
                break;
        case 3:
                return 2.597607;
                break;
        default:
                return 2.177051;
                break;
        }
}

/*double get_medical_child (int condition)
{
        switch (condition)
        {
        case 0:
                return med_con_y + log(11.3);
                break;
        case 1:
                return med_con_y + log(0.4);
                break;
        case 2:
                return med_con_y + log(4.5);
                break;
        case 3:
                return med_con_y + log(5);
                break;
        case 4:
                return med_con_y + log(1.1);
                break;
        case 5:
                return med_con_y + log(1);
                break;
        case 6:
                return med_con_y + log(0.4);
                break;
        case 7:
                return med_con_y + log(0.4);
                break;
        case 8:
                return med_con_y + log(0.4);
                break;
        case 9:
                return med_con_n;
```

```
                break;

        case 10:
                return med_con_y + log(0.4);
                break;
        case 11:
                return med_con_y + log(0.4);
                break;
        case 12:
                return med_con_y + log(0.4);
                break;
        default:
                return med_con_y;
                break;
        }
}*/

//adult medical

//Dying
double CONTROL_D = 0.002;
double Z_FO_D = 0.12;
double FI_SE_D = 0.0625;
double FIF_SF_D = 30;
double SF_PLUS_D = 310;

double get_dying_age_co(int num)
{
        switch (num)
        {
        case 0:
                return Z_FO_D;
                break;
        case 1:
                return FI_SE_D;
                break;
        case 2:
                return CONTROL_D;
                break;
        case 3:
                return FIF_SF_D;
                break;
        case 4:
                return SF_PLUS_D;
                break;
        default:
                return CONTROL_D;
                break;
        }
        return CONTROL_D;
}
```

8 Glossary

Epidemiology - is the study of the distribution and determinants of health-related states or events in specified populations and the application of this study to the control of health problems [3]
Latent - A count of all the infected period who are currently in **Latent time** [6]
Latent time - The period of time when you are infected but not yet infectious [6]
Transmission models - a model to predict the spread of a disease
Emergent phenomena - Due to the behaviour of agents depending on the behaviour of the agents around it, the behaviour of a given agent and thus the entire system can change drastically. This is because emergent phenomena can not be determined by a single component of the model, rather the sum of the components of the model this results in the model exhibiting properties that are independent from the components of the model [10]
Object orientated programming - OOP is a computer coding paradigm that is based on the concepts of objects, which contain variables and functions (typically called methods). In OOP objects tend to interact with each other. Most languages that are designed with OOP in mind tend to have the concept of class's this means that objects belong to a particular class.
Actor - A alternative name for agents.
COVIDSIM - The name for my ABM
Pointer - an address in memory that points to a piece of data, be it a vector, integer or a object.
Garbage collection - In other languages like python when a variable is no longer being used the variable will be deleted freeing up memory however with C++ when you allocate memory on the **heap**, you have to manually delete that object within the code as C++ does not do garbage collection on heap allocated objects only **stack** allocated objects.
stack - The stack in C++ refers to a large chunk of memory where in scope (local) variables are stored.
heap - The heap in C++ refers to all of your system memory. A object that is allocated on the heap can be put anywhere in the RAM within your system.

9 Bibliography

- [1] - American Centers for Disease Control and Prevention - https://www.cdc.gov/museum/timeline/
- [2] - National Center for Biotechnology Information - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3710332/
- [3] - American Centers for Disease Control and Prevention - https://www.cdc.gov/csels/dsepd/ss1978/SS1978.pdf
- [4] - National Center for Biotechnology Information - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3710332/
- [5] - https://www.imperial.ac.uk/media/imperial-college/medicine/sph/ide/gida-fellowships/Imperial-College-COVID19-NPI-modelling-16-03-2020.pdf
- [6] - https://nccid.ca/publications/glossary-terms-infectious-disease-modelling-proposal-consistent-language/
- [7] - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3710332/
- [8] - Grant Sanderson (3blue1brown) - https://www.youtube.com/watch?v=gxAaO2rsdIs
- [9] - http://www.geog.leeds.ac.uk/courses/other/crime/abm/general-modelling/index.html
- [10] - https://discovery.ucl.ac.uk/id/eprint/3342/1/3342.pdf
- [11] - Dr Tommy Thompson (ai and games) - https://www.youtube.com/watch?v=RcOdtwioEfI&t=641s , https://www.youtube.com/watch?v=WbHMxo11HcU

- [12] - National Center for Biotechnology Information - https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3936971/