



# **Agent-based Simulation of COVID-19 Transmission**

**Author:** Tian Zhang (c1912705)

**Supervisor:** Bailin Deng

**Module:** CMT400

MSc Computing

School of Computer Science and Informatics

Cardiff University

October 16, 2020

## **Abstract**

Agent-based modelling has been widely adopted in many disciplines, such as economics and social psychology. It shows high practicability in forecasting trend of units who possess certain pattern of behaviour. This project aims to develop an agent-based model using the ABM framework named Mesa to simulate the transmission of COVID-19 in a small-scale environment. The model contains a visualisation module and can run a single simulation configured using a GUI. It is also capable of running a mass of simulations in batches. The dissertation introduces some relevant background materials about COVID-19 and agent-based modelling. Then it goes through the development of the model and the instruction of each module. It also reviews the results and analysis of the simulations mentioned above. The project aims to offer a primary direction for the reader to understand how COVID-19 works and what we can do to control the epidemic situation.

## **Keywords**

Agent-based modelling; Computational simulation; COVID-19; Epidemic control

## **Acknowledgments**

I would like to express my deep gratitude to my supervisor, Dr Bailin Deng, for his dedicated advice and guidance. Dr Deng continuously encouraged me and was always willing and enthusiastic to help in any way he could throughout the project.

I would also like to extend my thanks to my dear friends, Ziyuan Qin and Chao Ye, for their useful and constructive advice on the project.

Finally, I wish to thank my parents for their support and encouragement throughout my study.

# Table of Contents

<b>Abstract.....</b>	<b>i</b>
<b>Acknowledgments .....</b>	<b>ii</b>
<b>Table of Contents .....</b>	<b>iii</b>
<b>List of Figures.....</b>	<b>vi</b>
<b>Chapter 1 – Introduction.....</b>	<b>1</b>
<b>Chapter 2 – Background .....</b>	<b>2</b>
2.1 Overview .....	2
2.2 Modes of transmission .....	2
2.3 Asymptomatic and symptomatic cases .....	3
2.4 Demographic characteristics .....	4
2.5 Transmission risk in different settings .....	5
2.6 Prevention and isolation.....	6
2.7 Agent-based model and COVID-19 .....	7
<b>Chapter 3 - Aim and objectives.....</b>	<b>9</b>
<b>Chapter 4 - Problem .....</b>	<b>10</b>
4.1 Framework selection.....	10
4.2 Factor selection .....	11
4.3 Simulation customisation.....	11
4.4 Visualisation design .....	11
<b>Chapter 5 – Approach.....</b>	<b>13</b>
5.1 Framework selection - Mesa .....	13
5.2 Factor selection .....	13
5.2.1 Fatality rate by age group .....	13
5.2.2 Face mask .....	13
5.2.3 Incubation and symptomatic stage.....	14
5.2.4 Hospital.....	14

5.2.5 Social distancing policy .....	14
5.2.6 Immunity loss .....	15
5.3 Simulation customisation - GUI configuration.....	15
5.4 Visualisation design .....	16
5.4.1 Single trials of the simulation .....	16
5.4.2 Multiple simulation and data analysis .....	17
<b>Chapter 6 – Implementation .....</b>	<b>18</b>
6.1 Code structure .....	18
6.2 Covid agent module .....	19
6.2.1 __init__() .....	19
6.2.2 set_age().....	20
6.2.3 check_quarantine_status() .....	21
6.2.4 hospital_treatment() .....	22
6.2.5 move().....	22
6.2.6 pass_covid() .....	23
6.2.7 get_infected() and infection_count() .....	24
6.2.9 infection_end().....	25
6.2.10 immunity_loss_check() and immunity_loss() .....	26
6.3 Covid model module.....	27
6.3.1 __init__() .....	27
6.3.2 check_all_agents() and get_end_time() .....	28
6.3.3 auto_quarantine_get/update/reset_quarantine_list() .....	29
6.3.4 manual_quarantine() .....	30
6.4 Configuration module .....	31
6.4.1 config.py .....	31
6.4.2 gui_config.py .....	32

6.5 Visualisation module.....	34
6.6 Batch run module.....	36
<b>Chapter 7 - Results and analysis.....</b>	<b>39</b>
7.1 Overview.....	39
7.2 Batch run 1 - Without additional conditions.....	39
7.3 Batch run 2 - Wearing face masks.....	42
7.4 Batch run 3 - Hospital activated.....	45
7.5 Batch run 4 - Flexible social distancing policy.....	47
7.6 Batch run 5 - strict social distancing policy.....	50
7.7 Batch run 6 - All measures adopted.....	52
<b>Chapter 8 – Conclusion .....</b>	<b>55</b>
<b>Chapter 9 – Reflection .....</b>	<b>56</b>
<b>References.....</b>	<b>58</b>

## List of Figures

Figure 1: Visualisation that is hard to comprehend.....	12
Figure 2: The GUI for simulation configuration.....	16
Figure 3: Stick icons representing different states of the agent.....	17
Figure 4: Code structure of the project.....	19
Figure 5: Constructor of the CovidAgent class .....	20
Figure 6: Code of set_age().....	21
Figure 7: Population of the UK in 2019 by age group .....	21
Figure 8: Code of check_quarantine_status() .....	22
Figure 9: Code of hospital_treatment().....	22
Figure 10: Code of move().....	23
Figure 11: Code of pass_covid().....	24
Figure 12: Code of get_infected() and infection_count().....	25
Figure 13: Code of infection_end().....	26
Figure 14: Code of immunity_loss_check() and immunity_loss() .....	27
Figure 15: Constructor of the CovidModel class .....	28
Figure 16: Code of check_all_agents() and get_end_time() .....	29
Figure 17: Code of quarantine methods .....	30
Figure 18: Code of manual_quarantine() .....	31
Figure 19: Instantiate ConfigParser as config.....	31
Figure 20: Syntax of ConfigParser in Python .....	32
Figure 21: GUI of the configuration module.....	33
Figure 22: Success message box.....	34
Figure 23: Error message box as exception handling.....	34
Figure 24: Method agent_portratal() .....	35
Figure 25: Visualisation of the simulation .....	36
Figure 26: Basic syntax for BatchRunner.....	37
Figure 27: Scatter plot generated by matplotlib .....	38

Figure 28: Other configurations for the tests .....	39
Figure 29: Parameters for test 1 .....	40
Figure 30: Scatter plot of fatality rate of test 1 .....	41
Figure 31: Scatter plot of steps taken for test 1 .....	41
Figure 32: Scatter plot of the highest morbidity rate of test 1 .....	42
Figure 33: Parameter for test 2 .....	43
Figure 34: Scatter plot of fatality rate of test 2 .....	43
Figure 35: Scatter plot of steps taken for test 2 .....	44
Figure 36: Scatter plot of the highest morbidity rate of test 2 .....	44
Figure 37: Parameters for test 3 .....	45
Figure 38: Scatter plot of fatality rate of test 3 .....	46
Figure 39: Scatter plot of steps taken for test 3 .....	46
Figure 40: Scatter plot of the highest morbidity rate of test 3 .....	47
Figure 41: Parameters for test 4 .....	48
Figure 42: Scatter plot of fatality rate of test 4 .....	48
Figure 43: Scatter plot of steps taken for test 4 .....	49
Figure 44: Scatter plot of the highest morbidity rate of test 4 .....	49
Figure 45: Parameters for test 5 .....	50
Figure 46: Scatter plot of fatality rate of test 5 .....	51
Figure 47: Scatter plot of steps taken for test 5 .....	51
Figure 48: Scatter plot of the highest morbidity rate of test 5 .....	52
Figure 49: Parameters for test 6 .....	53
Figure 50: Scatter plot of fatality rate of test 6 .....	53
Figure 51: Scatter plot of steps taken for test 6 .....	54
Figure 52: Scatter plot of the highest morbidity rate of test 6 .....	54



## **Chapter 1 – Introduction**

The epidemic of COVID-19 has drawn everyone's attention since December 2019. It has become the most momentous global event that caused a long-term impact on all human beings in the world. Under this circumstance, people must learn to understand what it is, how it works and thereby stop it from spreading. The research regarding this matter has been taken in several industries like epidemiology, economics and manufacturing. This project concentrates on social psychology and proposes research using the Agent-based Modelling framework named Mesa to create a simulation environment. The environment simulates transmission among people who live in a small community and offers a few optional configurations, such as wearing face masks and social distancing policy to monitor the effect caused during the simulation process. This dissertation first introduces some background knowledge required about COVID-19 and ABM. Then it discusses the aim and objectives of the project; the problem engaged in the construction of the model and the approach adopted to overcome the challenges. It also reviews the code structure and every feature of the model. The last chapter acquaints the reader with the tests created using the model and analyses the generated results: fatality rate, number of steps taken to eliminate the epidemic situation and the highest morbidity rate to get to the conclusion.

## **Chapter 2 – Background**

### **2.1 Overview**

In early December 2019, China recorded a cluster of cases of pneumonia in citizens of Wuhan, Hubei province, People's Republic of China. The responsible pathogen soon is recognised as a novel coronavirus. This virus was named SARS-CoV-2 since it can cause severe acute respiratory syndromes. Due to the highly infectious nature of this virus, the first confirmed infection case was found in Europe on January 24, 2020. In Europe, they named the virus as coronavirus disease 2019(COVID-19). The World Health Organization declared the outbreak a Public Health Emergency of international concern on January 30, 2020. There is plenty of evidence to show that COVID-19 is most closely related to known beta coronaviruses in bats. However, the role of an intermediate host in the infection of the earliest known human cases remains unclear(Andersen et al. 2020).

### **2.2 Modes of transmission**

According to some early-stage research studies, the basic reproductive number of the coronavirus disease 2019(COVID-19) estimated to be from 2.2 to 3.3, and the mortality rate is around 2.3%(Correction to Lancet Infect Dis 2020; published online March 27. [https://doi.org/10.1016/S1473-3099\(20\)30200-0](https://doi.org/10.1016/S1473-3099(20)30200-0). 2020).

By the end of June 2020, using currently available evidence, researchers from different nations in the world have concluded several common routes that COVID-19 relies on to transmit from person to person. The first human coronaviruses primary transmission mode is person-to-person contact through respiratory droplets generated by breathing, sneezing, coughing, etc., as well as direct contact with an infected subject or surface or through the hand-mediated transfer of the virus from contaminated fomites to the mouth, nose, or eyes(La Rosa et al. 2020). For the indirect contact cases, researchers found that thought the COVID-19 could not survive outside the human body for too long, but it

can spread via fomites(e.g., elevator buttons, door handles, or water hose taps). The virus aerosolisation in a public space with poor ventilation or crowded indoor areas could spread the virus without person-to-person contact(Cai et al. 2020). Some research studies also suggested that the COVID-19 can spread by airborne transmission.

Nonetheless, this transmission method is considered less robust comparing to others. The word 'Airborn' in public health area has a specialised meaning used to describe infections capable of being transmitted through exposure to those infectious, virus-containing respiratory droplets comprised of smaller droplets and particles that can remain suspended in the air over long distances and time. However, the main transmission mode is still person-to-person contact.

### **2.3 Asymptomatic and symptomatic cases**

Not all populations tested positive for COVID-19 are symptomatic. There is a review suggested that the proportion of positive cases that remained asymptomatic was estimated at 16%(La Scola et al. 2020). There are many other asymptomatic cases reported from different countries. This fact partially explained why the isolation of confirmed cases is vital in the prevention of COVID-19. What more dangerous to public health than an asymptomatic case is a pre-symptomatic case. While quantifying the asymptomatic transmission is challenging to researchers, the pre-symptomatic cases could be the major reason for transmission, according to some research. From the inferred data through modelling, pre-symptomatic transmission contributed to 48% and 62% of transmissions in Singapore and China(Ganyani et al. 2020).

As we already knew, people infected with COVID-19 can be infectious and be able to transmit the coronavirus to other individuals. However, when infected people become infectious is not easy to answer. There are many factors togetherly to decide when will an infected person become infectious to others. COVID-19 RNA could be detected in patients 1-3 days before symptom onset. Briefly, the infected individuals with mild to moderate symptoms could transmit the virus up to 8-9 days after symptom first emerged, and the patients with severe symptoms could be infectious for even longer(Criteria for

releasing COVID-19 patients from isolation. 2020). Thus, the WHO published a guideline on the isolation of people tested positive.

- For symptomatic patients, 10-days long isolation after symptom onset, plus at least three additional days without symptoms.
- For asymptomatic patients, ten days after a positive test for COVID-19.

## **2.4 Demographic characteristics**

Though the COVID-19 has a relatively moderate death rate, the range of symptoms reported is quite wide, ranging from mild symptoms like coughing, headache, fatigue, etc., to severe illness, such as fever or diarrhoea(CDC 2020). In one of the most significant reviews on symptoms of COVID-19, a study conducted by five universities, researchers combined data from 148 independent studies and 24,000 patients from nine countries to figure out the most common symptoms of this disease. The study found that 78 per cent of people had a fever; 57 per cent reported a cough; 31 per cent said they had suffered fatigue; 25 per cent lost the ability to smell; 23 per cent reported shortness of breathing(Grant et al. 2020). While the COVID-19 infection has been confirmed from almost every demographic group, the incidence of COVID-19 infection is mostly seen in adult male patients with the median age between 34 to 59 years old(Huang et al. 2020). The COVID-19 disease is also deemed to impact people with pre-existing respiratory, cardi-vascular, or diabetes conditions.

Furthermore, among all demographic groups, patients older than 60 years old tend to develop severe symptoms. Comparing to the old age group, the impact of COVID-19 seems less significant in the age group of less than 15. Fewer cases have been recorded from this age group (Huang et al. 2020). In conclusion, children under 15 may be less infected, and even if infected, they tend to have milder symptoms comparing to other age groups. There are many other studies showing evidence to support the conclusion we made before. In one of the research on mortality among different countries, the researcher found that even though the definition and counting approaches varied from nation to nation, the death rate increase with age(Goldstein and Lee 2020). This

research found that in the United States, people older than 70 contribute about 70% of the deaths related to COVID-19.

Moreover, researchers found the same pattern in other countries; all of this suggests that this global epidemic is much more fatal for the elderly age group than for the young age group. Similar evidence from Italy also supports the former conclusion. From one of the research conducted in Italy, researchers have found that children are most probably contact COVID-19 from their family members. Most of these children have developed mild symptoms(Parri et al. 2020).

## **2.5 Transmission risk in different settings**

After the outbreak, many doctors and researchers dedicated to figuring out the transmission model of the COVID-19. Nevertheless, there are still many questions related to the transmission method of the COVID-19 remained. After many independent research studies in different countries, researchers gradually understand the different settings of the transmission. Several reviews of outbreak reports show that COVID-19 could transmit effectively in a crowded space with poor ventilation. We could treat such a place as the ideal incubator of the COVID-19 transmission(Leclerc et al. 2020). Some researchers also find that some specific activities could also contribute to the transmission. Those activities include a close door choir session or a dancing practice(Hamner et al. 2020).

Another most vulnerable public places are health care facilities or nursing homes. Residents in nursing homes usually need close contact services, and most of them are from the elderly population, which means they tend to develop severe symptoms if they were infected. Front-line health works are the other most vulnerable group since they exposure to the contaminated environment for a longer time than other people. In a study targeted on front-line health-care workers, researchers found that the health-care workers were at increased risk for reporting a positive test result, comparing with the general population(Nguyen et al. 2020). They also found that adequate PPE supplements, clinical environment, and even ethnic background are the contributing

factors to this research. People's occupation background can also determine the chances they get infected by the COVID-19. There were multiple outbreaks reported in UK, EU, or the United States are from slaughterhouses, mines, building sites, or meat processing plants. So, some scientists believe that the working environment could also contribute to COVID-19 transmission (Leclerc et al. 2020). The possible factors including but not limited to are working in confined indoor spaces, lack of social distancing, close/direct contact with COVID-19 cases, and Insufficient or incorrect use of personal protective equipment.

## **2.6 Prevention and isolation**

Up to date, there is still no specific treatment for COVID-19; doctors could only using isolation and supportive care methods to treat the patients. Therefore, it is vital to timely recognise the suspect pathogen carrier and to adopt immediate actions to control the potential outbreak. In a community setting, isolating a known tested positive person is essential to prevent a massive outbreak. So, a large-scale test and early intervention are vital to stop spreading in the local community. In many communities, a 2-meter social distance is required to prevent COVID-19 transmission. Many authorities and governments also command to shut down all the crowded places for entertaining use. For example, many film theatres, live houses, and restaurants are required to keep close to prevent transmission of COVID-19. If large-scale community transmission has occurred, a temporal shutdown of schools and other public places, home isolation, personal hand hygiene, or even wearing face masks are the right strategies to adopt during the special situation.

However, for the health care settings, only isolation or recognising is not enough. Adequate supplement of PPE is essential to keep the front-line health-care workers safe, and an effective ventilation system is required to prevent large outbreaks.

Except for vaccine, specific cures, and isolations, there is another tool we can count on in terms of stopping the propagating of COVID-19. That tool is the immune system of human bodies, and the antibody produced after infected. In some early research,

scientists believed that most people's immune systems would respond to the COVID-19 by producing antibodies to prevent second-time infection(Wajnberg et al. 2020). If the result of this research holds, we would expect a plateau period for the pandemic transmission. Eventually, there might be a chance we defeat this virus since the majority of the population has developed antibodies to this virus. However, some recent studies challenged that finding. In a recent study, researchers found that while most patients will develop antibodies, this immunity might not last long(Liu et al. 2020).

## **2.7 Agent-based model and COVID-19**

COVID-19 has affected more than 170 countries and millions of individuals around the world. The number of confirmed cases and deceased patients is still increasing at an alarming rate in many nations. As we mentioned before, neither specific treatment nor effective vaccine is available for COVID-19 at this moment. Therefore, forecasting and tracing the potential risk is quite important. A good quality forecasting model can help the authorities in the world to make better decisions to prevent and control this novel virus. Generally speaking, we can coarsely categorise forecasting models into two categories; the first one is the traditional mathematical models based on stochastic theory; the second method has been evolving rapidly in the past decade, it is the machine learning techniques. Agent-based modelling is one of the models that cross statistical modelling and machine simulation. Agent-based models(ABMs) can provide researchers with statistical power and control tools to observe the interactions and behaviours of any number of agents(Jackson et al. 2017). Agent-based models are computational simulations that can simulate the interactions and behaviours of agents under a customised setting. ABMs are widely used in many disciplines, such as economics, sociology, and political science(Jackson et al. 2017). Agent-based models have been used in a pandemic simulation too. Agent-based models designed for infectious disease rely on a realistic population comprised of different demographic groups, a social network among the individuals in the population, and the disease model itself(Hoertel et al. 2020). For our research, we can simulate the transmission of

COVID-19 in different settings. For example, we include changeable rules in our simulation model so that people can see different simulation outcomes under a specific combination of controls. The benefit of using ABMs to simulate the transmission of COVID-19 is that it allows policymakers to measure the social impact when they have a limited understanding of this disease. However, all the forecasting techniques have their limits and challenges, which will be detailed later.



## **Chapter 3 - Aim and objectives**

This project is aiming to develop an agent-based model to simulate and analyse the transmission between a certain amount of people under different conditions. Thus there are a few objectives set below to regulate the development:

1. The model should be set up using a highly-scalable environment.
2. All the components of the model should be established with explicit attributes and functions.
3. The model should include as many factors as possible to form a fully-integrated simulation.
4. The model should provide the user with a lucid visualisation to understand the progress of the simulation.
5. There should be a user-friendly approach for the user to configure each simulation.
6. The model should be able to conduct multiple tests running in batches for data collection and generate appropriate diagrams for data analysis.

## Chapter 4 - Problem

### 4.1 Framework selection

Currently, there are many ABM simulation frameworks available on the Internet, which are written in different programming languages and have various capability. It is essential to find a proper one that could meet the requirement while being simple enough to learn considering the limited time given. There are some dominant frameworks listed below for the reader's reference:

**NetLogo:** NetLogo(Wilensky 1999) is a multi-agent programmable modelling environment written in Lisp. Students, teachers and researchers have widely used NetLogo for years, so the environment possesses well-developed documentation and loads of existing scientific models.

**FLAME:** The FLAME framework(Chin 2013) concentrates on creating agent-based models that run on high-performance computers (HPCs). Models of this framework are created based upon a model of computation called (extended finite) state machines, which determines the behaviour of the software execution.

**MASON:** MASON(Luke et al. 2003) is a discrete-event multi-agent simulation library core in Java contributed by the George Mason University's Evolutionary Computation Laboratory and the GMU Center for Social Complexity together, designed for both light-weight and large-scale simulations. It contains both a model library and an optional suite of visualisation tools in 2D and 3D.

**SPARK:** SPARK (Simple Platform for Agent-based Representation of Knowledge) (Solovyev 2010) is a cross-platform, free software for multi-scale agent-based modelling (ABM) by the team at CIRM at the University of Pittsburgh. It has advantages for the field of biomedical model development at the systems level. SPARK

aims to provide a light-weight, convenient, extensible and computationally efficient platform for ABM modellers.

## **4.2 Factor selection**

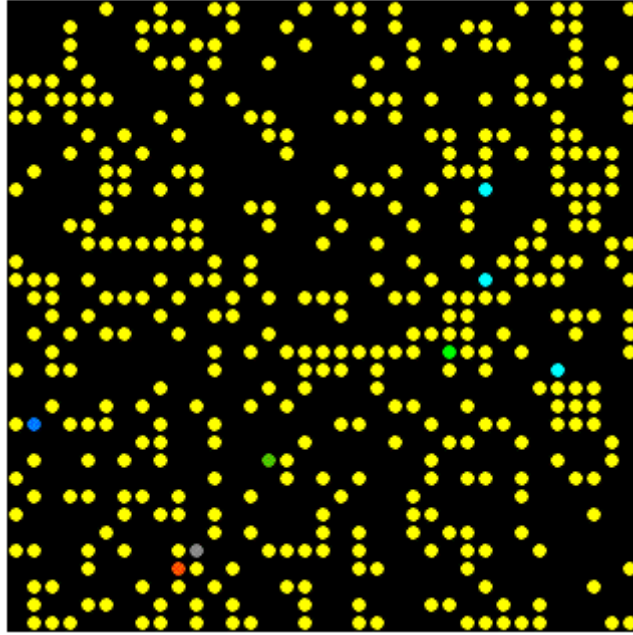
According to the background material introduced in chapter 2 regarding the COVID-19 outbreak, it is imperative to abstract the simulation model from all the complicated situations. The question of which factors should be considered into the scene then becomes an essential concern to be figured out in the first place, for it could influence the final results of the simulation extensively.

## **4.3 Simulation customisation**

Since it would not be enough to run the model for only once, the way to provide the user with an easy-configured simulation needs to be settled as well. Should the user configure each run by executing some code? Or should all the parameters be stored in a file separated from the code? This issue, in some sense, determines the target users of this project.

## **4.4 Visualisation design**

Another problem worth thinking is how the simulation should be presented to the users to help them directly get to the conclusion. Proceeding technical data analysis afterwards is one thing, however, making the simulation accessible to regular users when running it on site could be a different story. The fact is that some current ABM environments do generate the visualisation like the one shown in **Figure 1**, which is quite hard to comprehend.



**Figure 1: Visualisation that is hard to comprehend**

## **Chapter 5 – Approach**

### **5.1 Framework selection - Mesa**

After thoroughly investigating the most frequently used ABM frameworks, Mesa is selected to establish the simulation for this project. Mesa (Project Mesa Team 2016) is a modular framework written in Python to build, analyse and visualise agent-based models. It consists of three primary modules which are the modelling module, the analysis module and the visualisation module. The modelling and analysis modules are written in pure Python. The visualisation module is established based on the Python web framework named Tornado, resulting in the feasibility to upgrade the visualisation by adding JavaScript features to the front-end code. The reason to choose Mesa is that Python and Java are the two programming languages taught during the MSc Computing programme, between which Python has more flexibility and is easier to get the environment started. On the contrary, frameworks like NetLogo which are written in other languages like Lisp would be difficult for the coder who has the coding experience in Java/Python to understand since their syntax would look quite peculiar. Furthermore, the framework is still at the embryonic stage and has plenty of room for improvement.

### **5.2 Factor selection**

#### **5.2.1 Fatality rate by age group**

Dividing the infected agent by the age group is the first factor taken into consideration. As stated in chapter 2, the statistics show that the fatality rate of the COVID-19 varies in age. It would make the model more emulational by adding the age from 0 to 89 as an attribute of the agent.

#### **5.2.2 Face mask**

We can learn from the shortage of face mask supply at the beginning of the COVID-19

outbreak that whether people wear face masks is another significant measure to cease the passing probability of the virus. To test if this factor does have a macroscopical impact for preventing the virus from spreading, the agents are also offered the options of wearing face masks (or not).

### **5.2.3 Incubation and symptomatic stage**

It is not enough to only distinguish the agent by the healthy/infected state. One of the features of the COVID-19 that horrifies people is that if a person got infected, the virus has an incubation period that the carrier would show no symptoms. So both incubation and symptomatic stage have been taken into account to extend more possibility for the agent's behaviour.

### **5.2.4 Hospital**

The medical level cannot be neglected when concerning a virus outbreak like COVID-19. For this ABM model, a simplified hospital feature is created to simulate real-world medical treatment. It also has a maximum capacity and could overload when there are too many agents infected.

### **5.2.5 Social distancing policy**

According to the real-world situation, nations like the United Kingdom have conducted a social-distancing policy to wind down the spread of the virus. This simulation can test whether a self-isolation measure is useful and to what extent it could achieve the best results. A toggle to switch the automatic mode is also coded for this feature. When the toggle is off, the simulation will run under one solitary social distancing level from the beginning to the very end. In contrast, when the auto-mode is on, the simulation becomes more pliable. Stricter social distancing pattern will be adopted when the infected rate reaches a certain level.

### 5.2.6 Immunity loss

Several cases have been reported recently of people who recovered from the COVID-19 and got a second infection. The model has been simplified as the agent will gain immunity after recovery, and the probability of losing immunity again becomes an intriguing area to explore, which may reveal the potential risks after the outbreak is controlled in the future.

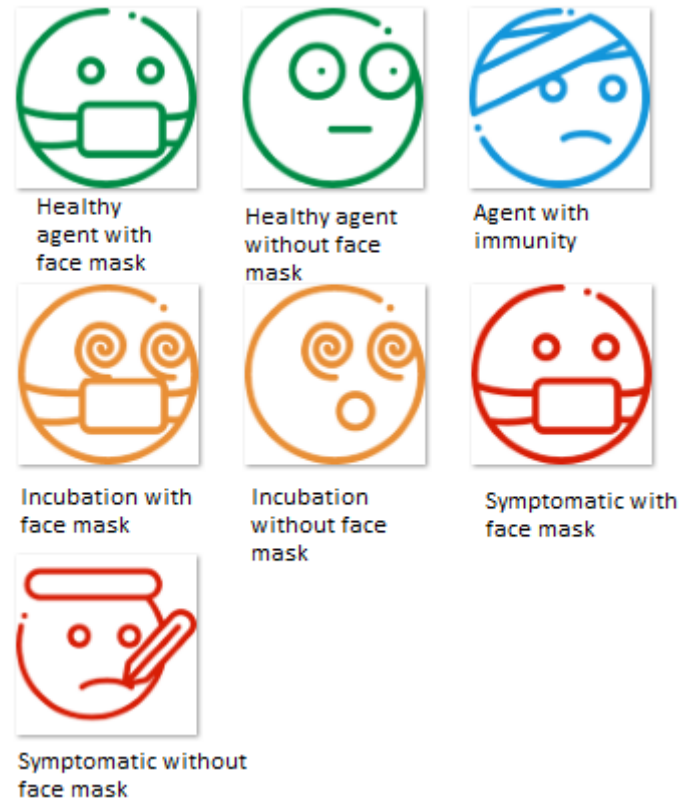
## 5.3 Simulation customisation - GUI configuration

There are several attempts proposed to improve user experience. At first, an individual Python module named `const.py` was created to store all the parameters. However, it was not clear enough for the user without Python experience to read and change the elements within the code. Then there goes the idea of collecting all the configuration data in a `.ini` file (Initialisation File), which is a default format for the Windows system to configure settings. To manage this feature, the project adopted a built-in Python module called `ConfigParser`. With decent annotations, the users can change the configuration of the simulation by editing `config.ini` with the default editor Notepad. Still, asking users to edit the `.ini` file manually leaves some uncertainty. Unexpected input without validation could cause unknown errors. Hence the last step to perfect this approach is by designing a graphical user interface using the built-in Python module `Tkinter`. With the GUI shown in **Figure 2**, the user can configure the simulation by filling a few blanks, and all the data entered will be validated by the code first.





the default circles with stick icons(Xu 2019) to improve the accessibility of the simulation. **Figure 3** displays the icon group applied to distinguish the agents in different states. It gives the user a more straightforward view to note the overview developments of the simulation. In the meantime, there are some critical figures calculated and presented in the visualisation module as line charts to make sure that the user can quickly check the changing trends of the statistics.



**Figure 3: Stick icons representing different states of the agent**

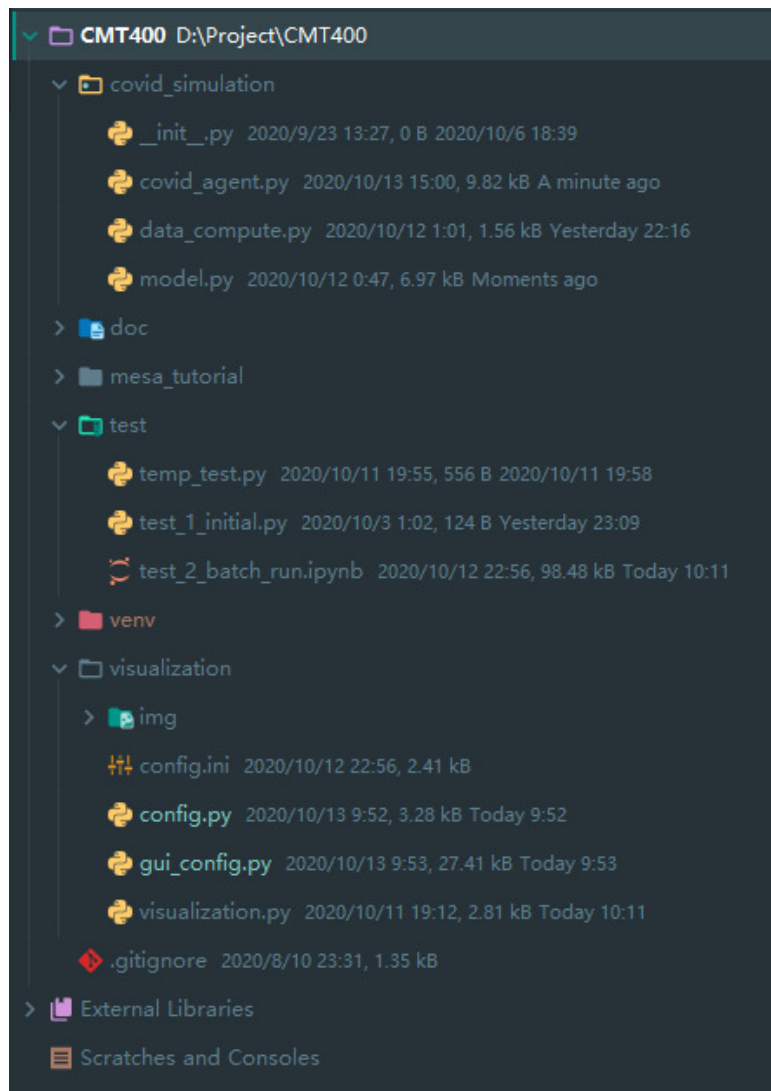
#### 5.4.2 Multiple simulation and data analysis

The project offers a grid visualisation module for the user to run a single simulation and observe the process thoroughly. Besides, it also supports the function of running multiple simulations in batches, using the module BatchRunner and the library named matplotlib. With this module, the user can run tests of a large number of simulations and get the key results displayed in scatter plots.

## Chapter 6 – Implementation

### 6.1 Code structure

The code structure of the project is displayed in **Figure 4**. The covid\_simulation package contains the kernel of the model: covid\_agent.py for the agent construction; model.py for the model construction and data\_compute.py for storing the essential figures calculations. The test\_2\_batch\_run.ipynb starts a Jupyter server to run simulation tests in batches using the BatchRunner module of Mesa. Lastly, the visualisation directory has all the components for visualisation of single simulations. The sub-folder /img stores all the icons for agents in different conditions. The file config.py is the initial version for generating configurations to config.ini while gui\_config.py is the upgraded version with a GUI which is more user-friendly. Visualisation.py holds all the elements for initiating a Tornado server to run the web simulations. All the major modules will be introduced below.



**Figure 4: Code structure of the project**

## 6.2 Covid agent module

Class CovidAgent is a subclass of the Agent class of the Mesa framework. It configures the behaviours of the agent.

### 6.2.1 `__init__()`

The constructor of the class defines all the attributes of an agent, as shown in **Figure 5**. It takes four parameters which are the ID of the agent, the model the agent belongs to, whether the agent is infected and whether it wears a face mask. The age attribute calls `self.set_age()` method to assign a random age to each agent. The other four attributes

state that whether the agent is symptomatic/in hospital/immune/dead, respectively. The `self.immunity_loss_toggle` and the `self.social_distancing_toggle`, on the other hand, are the trigger for starting a countdown for the agent's immunity loss and switching the agent to the self-isolated mode.

```
class CovidAgent(Agent):
    # each agent has an unique ID, and a status of whether it is infected
    def __init__(self, unique_id, model, is_infected: bool, wear_mask: bool):
        super().__init__(unique_id, model)
        # agents age from 0 to 89
        self.age = self.set_age()
        self.is_infected = is_infected
        self.wear_mask = wear_mask
        self.has_symptom = False
        self.in_hospital = False
        self.infection_toggle = False
        self.incubation_count = 0
        self.symptomatic_count = 0
        self.infection_countdown = -1
        self.has_immunity = False
        self.is_dead = False
        self.fatality_rate = 0
        self.immunity_loss_rate = 0
        self.immunity_countdown = -1
        self.immunity_loss_toggle = False
        self.social_distancing_toggle = False
```

**Figure 5: Constructor of the CovidAgent class**

### 6.2.2 set\_age()

This method assigns an age attribute to the agent, as displayed in **Figure 6**. In the beginning, it calls the `random.randint()` method to generate random ages. However, the statistics for age distribution in the United Kingdom (Clark 2020) and other countries show that the age distribution turns out to be a bell-shaped curve in **Figure 7**. Hence, the model considers using an edge-constrained normal distribution to imitate an approximate age distribution. Since the number of new-borns does not reach the bottom of the bell-shaped curve, the user could move the curve to the left by reducing the Mu

of the normal distribution to get the right shape. The method will then check if the age generated is between the section of 0 to 89, if it does, the method will assign it to the agent.

```
def set_age(self):
    while True:
        temp_age = math.floor(self.random.normalvariate(int(covid_model['MU']),
                                                         int(covid_model['SIGMA'])))
        if 0 <= temp_age <= 89:
            return temp_age
```

Figure 6: Code of set\_age()

### Population of the United Kingdom in 2019, by age group (in million people)

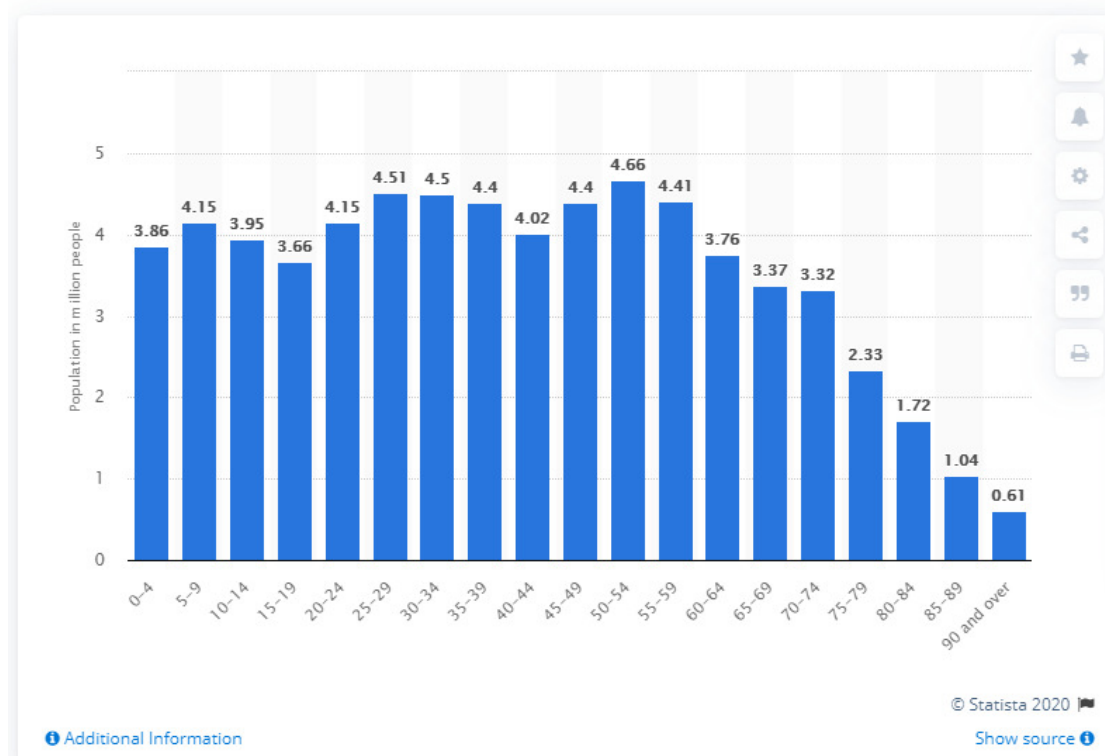


Figure 7: Population of the UK in 2019 by age group

#### 6.2.3 check\_quarantine\_status()

The method check\_quarantine\_status() displayed in **Figure 8** is for checking and changing the current status of the automatic social distancing mode for the agent. It verifies the auto\_social\_distancing attribute of the model module to confirm whether

the model is running in the automatic mode or the manual one. If the model is running in automatic mode, for each step, the agent would call the two methods of the model module to update the list to see whether itself should be self-isolated for the current step.

```
# check if auto quarantine change mode is on, if not switch to manual mode
def check_quarantine_status(self):
    if self.model.auto_social_distancing:
        self.model.auto_quarantine_get_symptomatic_rate()
        self.model.auto_quarantine_update_quarantine_list()
    else:
        self.model.manual_quarantine()
```

**Figure 8: Code of check\_quarantine\_status()**

#### 6.2.4 hospital\_treatment()

This method shown in **Figure 9** builds the core structure of the medical treatment feature. It first checks whether the model is running with a hospital. If the hospital is activated, it takes the agents who show symptoms out of the grid if the hospital still has room for more patients. Since there is still no vaccine or specific medicine to treat COVID-19, the hospital does not increase the chance of recovery; it only provides a quarantine environment for those who are removed from the grid.

```
# check if hospital mode is on
def hospital_treatment(self):
    if self.model.hospital_activated:
        # if the agent is symptomatic and there's still room in the hospital and it's not in it
        if self.has_symptom and self.model.hospital_occupation < \
            self.model.hospital_capacity and not self.in_hospital:
            # remove the agent from the grid
            self.model.grid.remove_agent(self)
            self.model.hospital_occupation += 1
            self.in_hospital = True
```

**Figure 9: Code of hospital\_treatment()**

#### 6.2.5 move()

The move() method shown in **Figure 10** decides where the agent moves in the grid. It

checks all the eight cells nearby and appends all the empty cells to a list. Then the agent randomly chooses one cell as its new position for that particular step.

```
def move(self):
    if not self.in_hospital:
        # find all the possible steps and if it's empty then append it into a list
        all_steps = self.model.grid.get_neighborhood(self.pos, moore=True,
                                                    include_center=False)

        possible_steps = []
        for cell in all_steps:
            if Grid.is_cell_empty(self.model.grid, cell):
                possible_steps.append(cell)

        # if the list is not empty then find a random cell for the agent to move in
        if possible_steps:
            new_position = self.random.choice(possible_steps)
            self.model.grid.move_agent(self, new_position)
```

**Figure 10: Code of move()**

#### 6.2.6 pass\_covid()

If the agent is infected with COVID-19, the method shown in **Figure 11** determines whether it could pass the virus to its neighbour agents. It has a for-loop to loop through all the nearby agents and a few if-else to check whether either agent is in self-isolation or wear a face mask. Then it compares the relevant passing probability with a randomly generated integer to get the passing result.

```

def pass_covid(self):
    if not self.in_hospital:
        # find all the cellmates near the agent
        cellmates = self.model.grid.get_neighbors(self.pos, True)
        for cellmate in cellmates:
            if not cellmate.has_immunity:
                # the agent itself or the target agent is being self-isolated
                if self.is_infected and (self.social_distancing_toggle or cellmate.social_distancing_toggle):
                    if self.random.randint(0, 1000) <= \
                        (float(pass_probability['PASS_PR_QUARANTINE']) * 1000):
                        cellmate.is_infected = True
                # both agents are not self-isolated
            else:
                if self.is_infected and not self.wear_mask:
                    if not cellmate.wear_mask:
                        if self.random.randint(0, 1000) <= \
                            (float(pass_probability['PASS_PR_BOTH_OFF']) * 1000):
                                cellmate.is_infected = True
                    else:
                        if self.random.randint(0, 1000) <= \
                            (float(pass_probability['PASS_PR_CONTACT_ON']) * 1000):
                                cellmate.is_infected = True
                elif self.is_infected and self.wear_mask:
                    if not cellmate.wear_mask:
                        if self.random.randint(0, 1000) <= \
                            (float(pass_probability['PASS_PR_CARRIER_ON']) * 1000):
                                cellmate.is_infected = True
                    else:
                        if self.random.randint(0, 1000) <= \
                            (float(pass_probability['PASS_PR_BOTH_ON']) * 1000):
                                cellmate.is_infected = True

```

Figure 11: Code of pass\_covid()

### 6.2.7 get\_infected() and infection\_count()

These two methods displayed in **Figure 12** generate the incubation and symptomatic period after an agent is confirmed infected, and start a countdown for both periods. When the incubation ends, the self.has\_symptom toggle changes to True to state that the agent starts to show symptoms. Besides, there is a self.infection\_toggle set to make sure that the pass\_covid() method does not reset the countdown.



```

def get_infected(self):
    # infection toggle is to prevent from multiple info update
    if self.is_infected and not self.infection_toggle:
        self.incubation_count = self.random.randint(int(incubation['INCUBATION_MIN']),
                                                    int(incubation['INCUBATION_MAX']))
        self.symptomatic_count = self.random.randint(int(symptomatic['SYMPTOMATIC_MIN']),
                                                    int(symptomatic['SYMPTOMATIC_MAX']))
        self.infection_countdown = self.incubation_count + self.symptomatic_count
        self.infection_toggle = True

def infection_count(self):
    # every step the agent moves the countdown goes down by 1
    if self.is_infected:
        self.infection_countdown -= 1
    if self.infection_countdown == self.symptomatic_count:
        self.has_symptom = True

```

**Figure 12: Code of get\_infected() and infection\_count()**

### 6.2.9 infection\_end()

This method shown in **Figure 13** determines the agent's fate after the self.infection\_countdown attribute reaches 0. It compares a randomly generated integer with the fatality rate configured by the agent's age to decide whether the agent dies from the COVID-19 or survives and gains immunity. If the agent survives and is currently in the hospital, the model will place the agent back into the grid. Otherwise, the agent will be removed from both the grid and the schedule if it is dead.

```

def infection_end(self):
    # when the countdown reaches 0
    if self.infection_countdown == 0:
        self.fatality_rate = float(fatality_rate
                                   [str(math.floor(self.age / 10))]) * 1000
        if self.fatality_rate >= self.random.randint(0, 1000):
            # the agent dies
            self.is_dead = True
            self.is_infected = False
            self.has_symptom = False
            if not self.in_hospital:
                # if it's not in the hospital then remove it from the grid
                self.model.grid.remove_agent(self)
            else:
                # if in the hospital then remove it from the hospital
                self.in_hospital = False
                self.model.hospital_occupation -= 1
            # remove the agent from the schedule
            self.model.schedule.remove(self)
            # FOR DEBUG USAGE
            # print("***\nAgent " + str(self.unique_id) + " is dead and rem
            self.infection_countdown = -1
        else:
            # the agent got immunity
            self.has_immunity = True
            # if it recovers in the hospital, place it back to the grid
            if self.in_hospital:
                self.model.grid.position_agent(self)
                self.in_hospital = False
                self.model.hospital_occupation -= 1
            # reset the status
            self.is_infected = False
            self.has_symptom = False
            self.infection_countdown = -1

```

Figure 13: Code of infection\_end()

#### 6.2.10 immunity\_loss\_check() and immunity\_loss()

These two methods which are shown in **Figure 14** complete the immunity loss feature of the agent. They first make a comparison to decide whether the agent will lose its

immunity. If the answer is yes, the method assigns the agent with another countdown to calculate the time left. When the countdown reaches 0, the immune agent will be reset to a normal healthy one. Like the `get_infected()` method, this one also switches the `self.immunity_loss_toggle` to prevent repetitious countdown.

```
def immunity_loss_check(self):
    # if the agent has immunity and not been checked yet
    if self.has_immunity and not self.immunity_loss_toggle:
        # turn on the toggle to prevent multiple check
        self.immunity_loss_toggle = True
        # get the probability of losing immunity
        self.immunity_loss_rate = float(immunity_loss['IMMUNITY_LOSS_PR']) * 1000
        if self.random.randint(0, 1000) <= self.immunity_loss_rate:
            self.immunity_countdown = self.random.randint(int(immunity_loss['IMMUNITY_LOSS_MIN']),
                                                            int(immunity_loss['IMMUNITY_LOSS_MAX']))

def immunity_loss(self):
    if self.immunity_loss_toggle and self.immunity_countdown > 0:
        self.immunity_countdown -= 1
    # when the countdown reaches 0, reset the agent to a healthy one without immunity
    elif self.immunity_loss_toggle and self.immunity_countdown == 0:
        self.immunity_countdown = -1
        self.has_immunity = False
        self.infection_toggle = False
        self.immunity_loss_toggle = False
```

**Figure 14: Code of `immunity_loss_check()` and `immunity_loss()`**

## 6.3 Covid model module

Class `CovidModel` is a subclass of the `Model` class of Mesa framework. It assigns all the agent to a grid and controls their behaviour. All the critical components are introduced below.

### 6.3.1 `__init__()`

The constructor of `CovidModel` is displayed in **Figure 15**. It requires a few key parameters such as the total number of the agents  $N$ ; the number of initially infected agents  $M$ ; the number of healthy/infected agents who wear face masks  $J$  and  $K$  and the capacity of the hospital  $L$ . It also necessitates the size of the grid and the flag for activating the hospital and social distancing policy. The user can set all the figures

mentioned above in the configuration module. For the COVID-19 simulation, each agent is regarded as one person, so multiple agents can not exist in a single cell. Thus the `self.grid` attribute of the model is set to `SingleGrid`. As for the scheduler, it needs to activate each agent once per step in random order, resulting in the `RandomActivation` class adopted for the `schedule` attribute. The last attribute the constructor has is `self.data_collector`, which is an instance of the `DataCollector` class for collecting all the real-time simulation figures calculated in `data_compute.py`.

```
def __init__(self, N: int, M: int, J: int, K: int, L: int, width, height,
             hospital_activated: bool, auto_social_distancing: bool,
             manual_quarantine_lvl: int):
    super().__init__()
    self.agent_number = N
    self.initial_infected = M
    self.healthy_with_mask = J
    self.carrier_with_mask = K
    self.hospital_capacity = L
    self.hospital_occupation = 0
    self.hospital_activated = hospital_activated
    self.auto_social_distancing = auto_social_distancing
    self.manual_quarantine_lvl = manual_quarantine_lvl
    self.grid = SingleGrid(width, height, True)
    self.schedule = RandomActivation(self)
    self.schedule_end_steps = 0
    self.schedule_end_flag = False
    self.highest_morbidity_rate = 0
    self.running = True
    self.agent_list = []
    self.symptomatic_list = []
    self.symptomatic_rate = 0
    self.quarantine_toggle = 0
    self.quarantine_list = []
    self.quarantined_agent_length_1 = 0
    self.quarantined_agent_length_2 = 0
```

**Figure 15: Constructor of the CovidModel class**

### 6.3.2 `check_all_agents()` and `get_end_time()`

These two methods which are shown in **Figure 16** check when there is no agent infected any more and then assign the time stamp to the `self.schedule_end_steps` attribute to

record the number of steps that the simulation takes to control the epidemic situation.

```
# A function to check if all agents are not infected
def check_all_agents(self):
    for agent in self.agent_list:
        if agent.is_infected:
            return False
    return True

def get_end_time(self):
    # if all the agents get rid of infection and the model has
    if self.check_all_agents() and not self.schedule_end_flag:
        self.schedule_end_steps = self.schedule.steps
        # set the flag to true to only record the steps once
        self.schedule_end_flag = True
```

Figure 16: Code of check\_all\_agents() and get\_end\_time()

### 6.3.3 auto\_quarantine\_get/update/reset\_quarantine\_list()

These three methods shown in **Figure 17** are the core method for the automatic social-distancing mode. The auto\_quarantine\_get\_quarantine\_list() method calculate the rate of symptomatic agents for each step. Next, the auto\_quarantine\_update\_quarantine\_list() method checks whether the rate reaches either threshold to trigger a different level of social-distancing policy. If it does, the method randomly picks a set of agents using random.sample() and set them to the self-isolated status.

```

def auto_quarantine_get_symptomatic_rate(self):
    # if the agent has symptom, append it to the list
    self.symptomatic_list = []
    for agent in self.agent_list:
        if agent.has_symptom:
            self.symptomatic_list.append(agent)
    # then calculate the symptomatic_rate
    self.symptomatic_rate = len(self.symptomatic_list) / len(self.agent_list)

def auto_quarantine_update_quarantine_list(self):
    # get number of agents who need to be self-isolated for both quarantine level
    self.quarantined_agent_length_1 = round(len(self.agent_list) * float(quarantine_rate["1"]))
    self.quarantined_agent_length_2 = round(len(self.agent_list) * float(quarantine_rate["2"]))
    # level 1
    if float(quarantine_threshold['level_1_threshold']) <= self.symptomatic_rate < float(
        quarantine_threshold['level_2_threshold']) and self.quarantine_toggle != 1:
        self.auto_quarantine_reset_quarantine_list()
        # pick random agents (number set by lvl 1) to stay still
        self.quarantine_list = self.random.sample(self.agent_list, self.quarantined_agent_length_1)
        # set the toggle to 1
        self.quarantine_toggle = 1
    # level 2
    elif float(quarantine_threshold['level_2_threshold']) <= self.symptomatic_rate and self.quarantine_toggle != 2:
        self.auto_quarantine_reset_quarantine_list()
        # pick random agents (number set by lvl 2) to stay still
        self.quarantine_list = self.random.sample(self.agent_list, self.quarantined_agent_length_2)
        # set the toggle to 2
        self.quarantine_toggle = 2
    elif float(quarantine_threshold['level_1_threshold']) > self.symptomatic_rate and self.quarantine_toggle != 0:
        self.auto_quarantine_reset_quarantine_list()
        # at lvl 0 no one needs to self-isolate
        self.quarantine_list = []
        self.quarantine_toggle = 0
    # all the agents in the list to stay still
    for agent in self.quarantine_list:
        agent.social_distancing_toggle = True

def auto_quarantine_reset_quarantine_list(self):
    for agent in self.agent_list:
        agent.social_distancing_toggle = False

```

**Figure 17: Code of quarantine methods**

### 6.3.4 manual\_quarantine()

If the auto-mode is switched off, this method presented in **Figure 18** will be called to pick a specific amount of agents according to the social-distancing level entered as a parameter to proceed self-isolation.

```

def manual_quarantine(self):
    if not self.quarantine_toggle:
        self.quarantine_toggle = 1
        quarantined_agent_length = round(len(self.agent_list)
                                         * float(quarantine_rate[str(self.manual_quarantine_lvl)]))
        self.quarantine_list = self.random.sample(self.agent_list, quarantined_agent_length)
        for agent in self.quarantine_list:
            agent.social_distancing_toggle = True

```

**Figure 18: Code of manual\_quarantine()**

## 6.4 Configuration module

### 6.4.1 config.py

The model selects the built-in module of Python named ConfigParser to set up all the configurations. At first, all the settings are coded in a Python file called config.py. According to the official documentation, an instance of the ConfigParser class is announced as config in **Figure 19**. The syntax for creating a standard .ini file is similar to the way of creating a dictionary using Python, which is like "config[section] = {key: value}". Furthermore, the .ini file takes a semicolon as a symbol to start a line of annotation, for which can be hard-coded as a line of string in Python. So the final code for this module is presented as in **Figure 20**.

```

config = configparser.ConfigParser()

```

**Figure 19: Instantiate ConfigParser as config**

```

config['covid_model'] = {
    "; Number of agents\n"
    "N": "200",
    "; Number of initially infected agents\n"
    "M": "30",
    "; Number of healthy agents who wear face masks\n"
    "J": "0",
    "; Number of infected agents who wear face masks\n"
    "K": "0",
    "; Mu of age distribution\n"
    "MU": "30",
    "; Sigma of age distribution\n"
    "SIGMA": "15",
    "; Width of the grid\n"
    "width": "50",
    "; Height of the grid\n"
    "height": "50"
}

```

**Figure 20: Syntax of ConfigParser in Python**

#### 6.4.2 gui\_config.py

A Graphical User Interface presented in **Figure 21** is added to this feature using the built-in module of Python called Tkinter to make this module more user-friendly. The GUI uses radio buttons and Comboboxes to prevent the user from entering the wrong input for the primary settings. As for the rest of the entries, they all have a default value for the users' reference. Other than that, the submission button is linked to the method `update_btn_pressed()`, and it calls the `validate_data()` method to make sure that all the data submitted is in the correct format and range. If all the data passes the validation, they will be written into the `config.ini` file, and there will be a message box popping out as a success hint (**Figure 22**). Otherwise, if any of the data goes wrong, the GUI will



raise an error message box (Figure 23).

**COVID-19 Agent-based Simulation Configuration**

**Simulation Settings**

Activate Hospital: ☐ Yes ☒ No

Activate Automatic Social Distancing Policy: ☐ Yes ☒ No

Manually Set Social Distancing Level:  (This setting is only activated when auto social-distancing policy is switched off)

**Model Settings**

Number of Agents:

Number of Initially Infected Agents:

Number of Healthy Agents Wear Face Masks:

Number of Carriers Wear Face Masks:

Mu of Age Distribution:

Sigma of Age Distribution:

Width of The Grid:

Height of The Grid:

Hospital Capacity:

**Social Distancing Policy**

Level 0 Rate of Social-distancing Agents:

Level 1 Rate of Social-distancing Agents:

Level 2 Rate of Social-distancing Agents:

Threshold For Agents To Start Level 1 Social-distancing Policy:

Threshold For Agents To Start Level 2 Social-distancing Policy:

**Fatality Rate**

0~9 Years Old:

10~19 Years Old:

20~29 Years Old:

30~39 Years Old:

40~49 Years Old:

50~59 Years Old:

60~69 Years Old:

70~79 Years Old:

80~89 Years Old:

**Passing Probability**

Both Agents Wear Masks:

Only The Carriers Wear Masks:

Only The Healthy Contacts Wear Masks:

Both Agents Don't Wear Masks:

One Of The Agents Is Self-isolated:

**Incubation & Symptomatic**

Minimum Incubation Period:

Maximum Incubation Period:

Minimum Symptomatic Period:

Maximum Symptomatic Period:

**Immunity Settings**

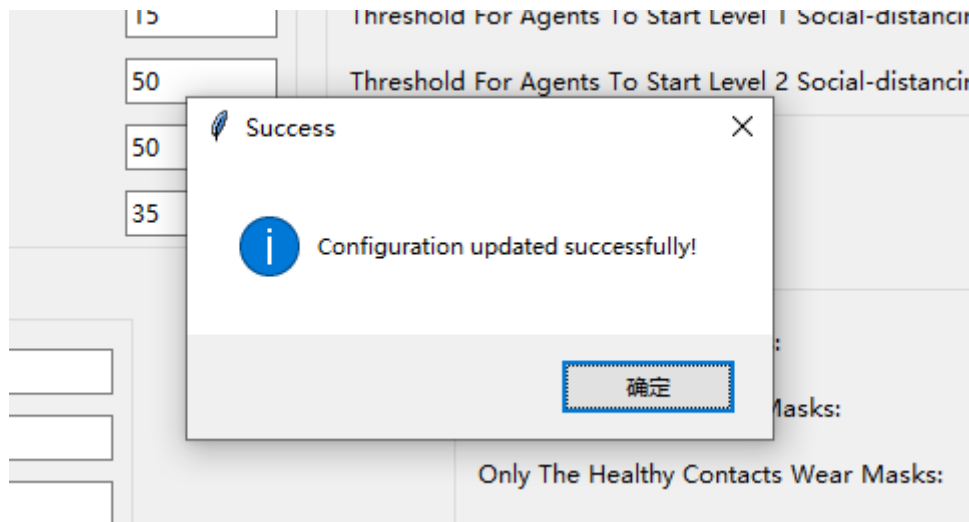
Probability of Agents Losing Immunity After Recovery:

Minimum Time Agents Lose Immunity After Recovery:

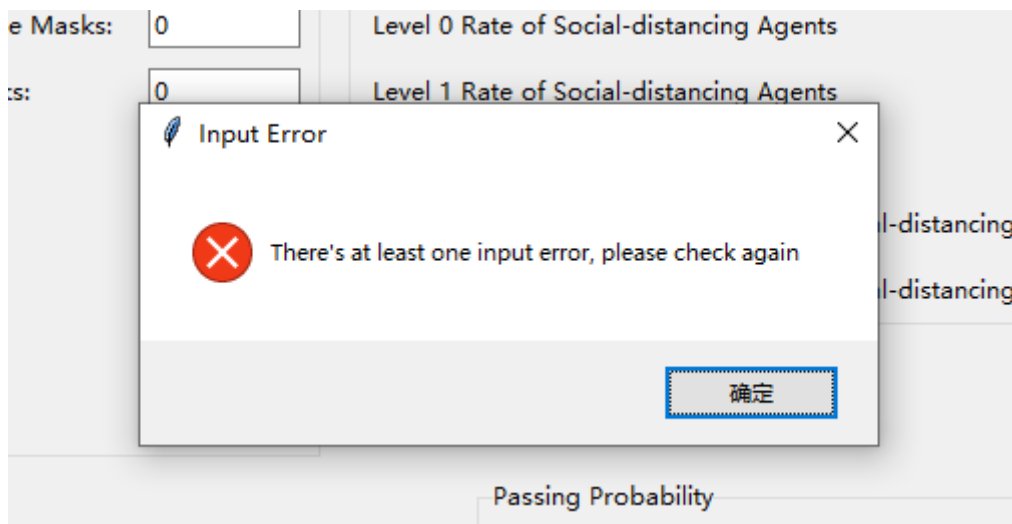
Maximum Time Agents Lose Immunity After Recovery:

**Update Configuration**

**Figure 21: GUI of the configuration module**



**Figure 22: Success message box**



**Figure 23: Error message box as exception handling**

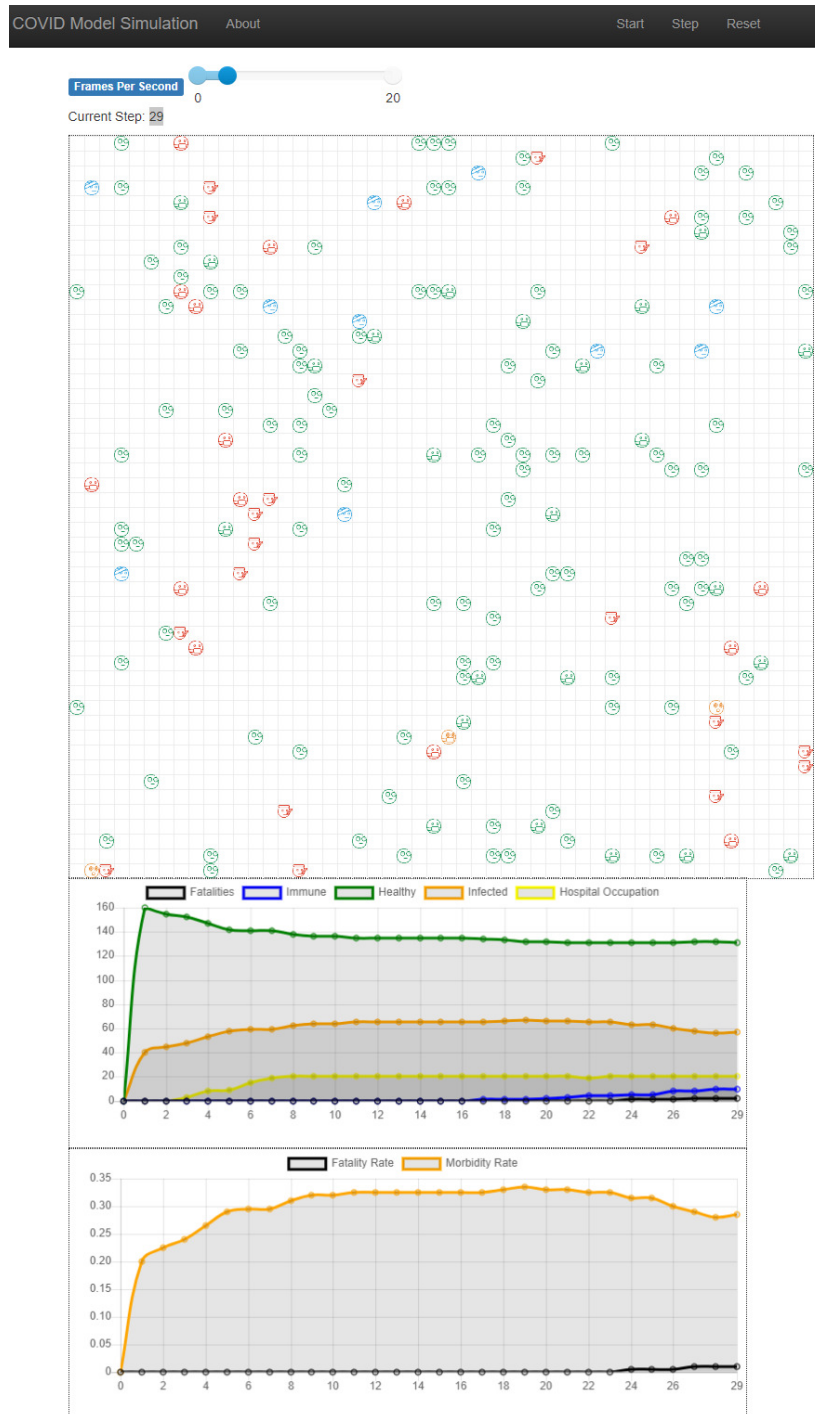
## 6.5 Visualisation module

In this module, there is a method named `agent_portrayal()` (**Figure 24**) coded to select the relevant icon for each agent according to its status. The visualisation consists of three components, a `CanvasGrid` instance for drawing the grid with agents acting inside and two `ChartModule` instances for displaying two line charts with key data calculated in `data_compute.py`. The `ChartModule` collects the data from the data collector created in the model module. When the user runs this module, it establishes a web server using its `ModularServer` feature, which is an adaption of the Tornado web framework. The `ModularServer` picks the configurations from the `config.ini` file. The final visualisation

of the simulation is shown in **Figure 25**. The user can watch the whole simulation, check the data for each step and control the speed of its process.

```
def agent_portrayal(agent):
    portrayal = {
        "Filled": "true",
        "Layer": 0,
    }
    if agent.has_immunity:
        portrayal["Shape"] = "img/immune.png"
    elif agent.is_infected and not agent.has_symptom and not agent.wear_mask:
        portrayal["Shape"] = "img/incubation_without_mask.png"
    elif agent.is_infected and not agent.has_symptom and agent.wear_mask:
        portrayal["Shape"] = "img/incubation_mask.png"
    elif agent.is_infected and agent.has_symptom and not agent.wear_mask:
        portrayal["Shape"] = "img/symptomatic_without_mask.png"
    elif agent.is_infected and agent.has_symptom and agent.wear_mask:
        portrayal["Shape"] = "img/symptomatic_mask.png"
    elif not agent.is_infected and agent.wear_mask:
        portrayal["Shape"] = "img/healthy_mask.png"
    else:
        portrayal["Shape"] = "img/healthy_without_mask.png"
    return portrayal
```

**Figure 24: Method agent\_portratal()**



**Figure 25: Visualisation of the simulation**

## 6.6 Batch run module

This module is coded in a Jupyter Notebook file. It is constructed using the BatchRunner module of Mesa, which requires a few parameters to proceed with a simulation. The module provides the user with a few pre-configured tests. The syntax

of the code is shown in **Figure 26**. For each test, BatchRunner takes a Mesa model class as well as two dictionaries with both constant and variable parameters stored in them. It also needs to take the number of iterations and maximum steps per run as input. As for output, it expects a model\_reporter parameter which takes methods from the data\_compute.py to calculate the results wanted, and then it can present the results in scatter plots with the matplotlib.pyplot module, as displayed in **Figure 27**.

```
# Batch run No.1, no agents wear masks, no hospital, no social distancing
fixed_params_0 = {
    "width": 50,
    "height": 50,
    "N": 200,
    "J": 0,
    "K": 0,
    "L": 0,
    "hospital_activated": 0,
    "auto_social_distancing": 0,
    "manual_quarantine_lvl": 0
}

variable_params_0 = {
    "M": range(1, 32, 5)
}

batch_run_0 = BatchRunner(CovidModel,
                          variable_params_0, variable_params_0: {'M': range(1, 32, 5)}
                          fixed_params_0, fixed_params_0: {'width': 50, 'height': 50, 'N': 200, 'J': 0, 'K': 0, 'L': 0, 'hospital_activated': 0, 'auto_social_distancing': 0, 'manual_quarantine_lvl': 0}
                          iterations=20,
                          max_steps=300,
                          model_reporters={"Fatality_Rate": compute_fatality_rate,
                                           "Step_Number": get_step_number,
                                           "Highest_Morbidity_Rate": get_highest_morbidity_rate})

batch_run_0.run_all() batch_run_0: <mesa.batchrunner.BatchRunner object at 0x000001C24D31F288>

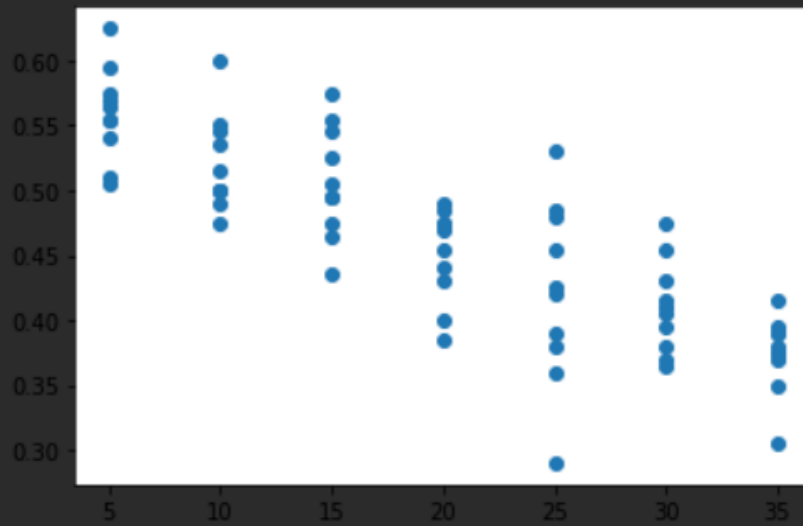
###

run_data_0 = batch_run_0.get_model_vars_dataframe() batch_run_0: <mesa.batchrunner.BatchRunner object at 0x000001C24D31F288>
run_data_0.head() run_data_0: {DataFrame: (140, 14)}
plt.scatter(run_data_0.M, run_data_0.Fatality_Rate) run_data_0: {DataFrame: (140, 14)} run_data_0: {DataFrame: (140, 14)}
```

**Figure 26: Basic syntax for BatchRunner**

```
9 run_data_2 = batch_run_2.get_model_vars_dataframe()
run_data_2.head()
plt.scatter(run_data_2.L, run_data_2.Fatality_Rate)

9 <matplotlib.collections.PathCollection at 0x16081453488>
```



**Figure 27: Scatter plot generated by matplotlib**

## Chapter 7 - Results and analysis

### 7.1 Overview

The batch run module simulates a testing environment. In short, the environment creates a 50\*50 grid and initially places 200 agents in it. There are six tests in total and 20 iterations for each test. The fatality rate, the number of steps taken to end the epidemic situation and the highest morbidity rate the model reaches are displayed as the data on the Y-axis of the scatter plots, respectively. The maximum step of each simulation is set to 300, which means if there is still agent infected by the point the step reaches 300, the simulation will terminate, and the number of steps taken will be set to 300. Other configurations are set, as shown in **Figure 28**.

The screenshot displays a configuration interface for a simulation, organized into several sections with input fields and sliders.

- Model Settings:**
  - Number of Agents: 200
  - Number of Initially Infected Agents: TBD
  - Number of Healthy Agents Wear Face Masks: TBD
  - Number of Carriers Wear Face Masks: TBD
  - Mu of Age Distribution: 30
  - Sigma of Age Distribution: 15
  - Width of The Grid: 50
  - Height of The Grid: 50
  - Hospital Capacity: TBD
- Social Distancing Policy:**
  - Level 0 Rate of Social-distancing Agents: 0
  - Level 1 Rate of Social-distancing Agents: 0.6
  - Level 2 Rate of Social-distancing Agents: 0.9
  - Threshold For Agents To Start Level 1 Social-distancing Policy: 0.1
  - Threshold For Agents To Start Level 2 Social-distancing Policy: 0.2
- Fatality Rate:**
  - 0~9 Years Old: 0.5
  - 10~19 Years Old: 0.4
  - 20~29 Years Old: 0.2
  - 30~39 Years Old: 0.3
  - 40~49 Years Old: 0.3
  - 50~59 Years Old: 0.4
  - 60~69 Years Old: 0.4
  - 70~79 Years Old: 0.4
  - 80~89 Years Old: 0.5
- Passing Probability:**
  - Both Agents Wear Masks: 0.015
  - Only The Carriers Wear Masks: 0.05
  - Only The Healthy Contacts Wear Masks: 0.7
  - Both Agents Don't Wear Masks: 0.95
  - One Of The Agents Is Self-isolated: 0.02
- Incubation & Symptomatic:**
  - Minimum Incubation Period: 1
  - Maximum Incubation Period: 14
  - Minimum Symptomatic Period: 14
  - Maximum Symptomatic Period: 35
- Immunity Settings:**
  - Probability of Agents Losing Immunity After Recovery: 0.83
  - Minimum Time Agents Lose Immunity After Recovery: 7
  - Maximum Time Agents Lose Immunity After Recovery: 14

**Figure 28: Other configurations for the tests**

### 7.2 Batch run 1 - Without additional conditions

The first test is to simulate a basic environment in which no agents wear face masks,

and every agent moves around continuously no matter what status it is in (manual social distancing level set to 0). There is also no hospital established, and the government does not implement any social-distancing policy. The parameters are listed in **Figure 29**, for which the number of the initially infected agent is set as the variable on the X-axis. The fatality rate, number of steps taken and the highest morbidity rate are presented in the scatter plot in **Figure 30/31/32**.

```
# Batch run No.1, no agents wear masks, no hospital, no social distancing
fixed_params_0 = {
    "width": 50,
    "height": 50,
    "N": 200,
    "J": 0,
    "K": 0,
    "L": 0,
    "hospital_activated": 0,
    "auto_social_distancing": 0,
    "manual_quarantine_lvl": 0
}

variable_params_0 = {
    "M": range(1, 32, 5)
}

batch_run_0 = BatchRunner(CovidModel,
                           variable_params_0, variable_params_0: {'M': range(1, 32, 5)}
                           fixed_params_0, fixed_params_0: {'width': 50, 'height': 50, 'N': 200, '
                           iterations=20,
                           max_steps=300,
                           model_reporters={"Fatality_Rate": compute_fatality_rate,
                                             "Step_Number": get_step_number,
                                             "Highest_Morbidity_Rate": get_highest_morbidity_rate})
```

**Figure 29: Parameters for test 1**



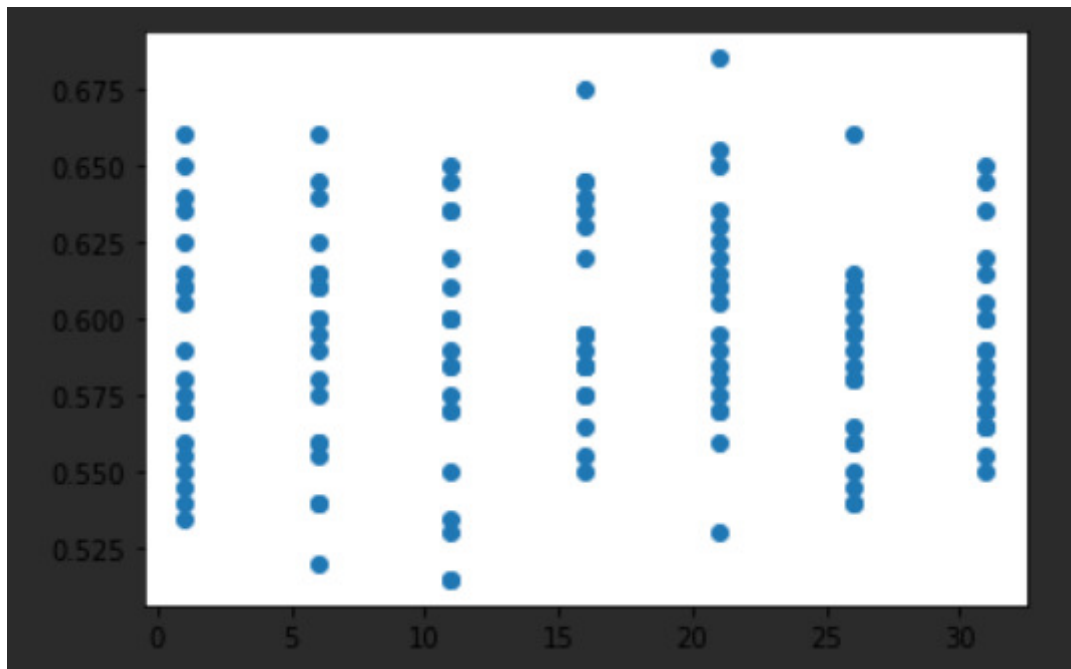


Figure 30: Scatter plot of fatality rate of test 1

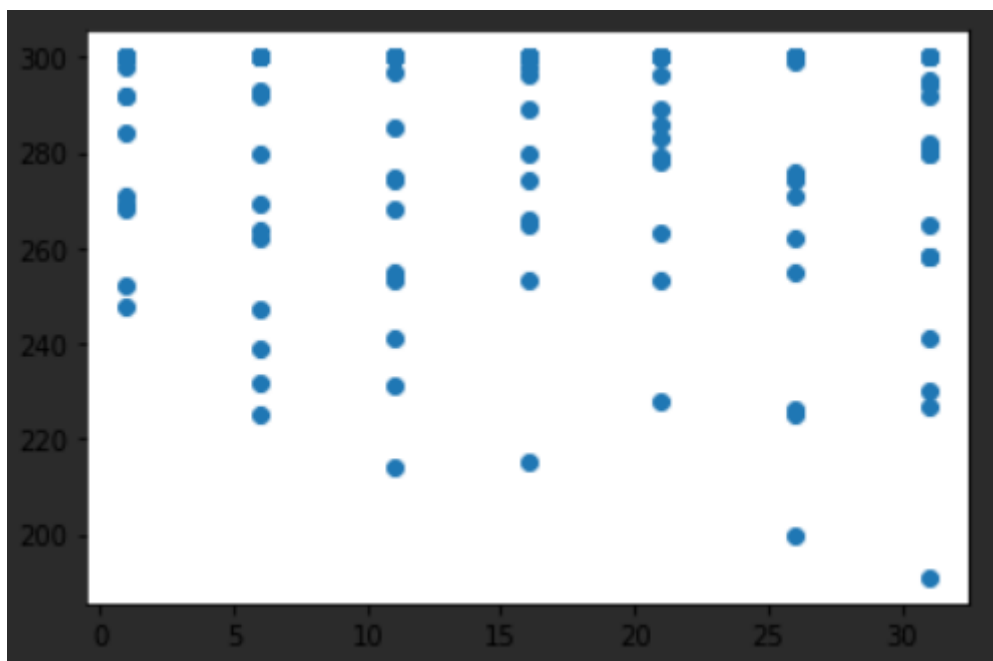
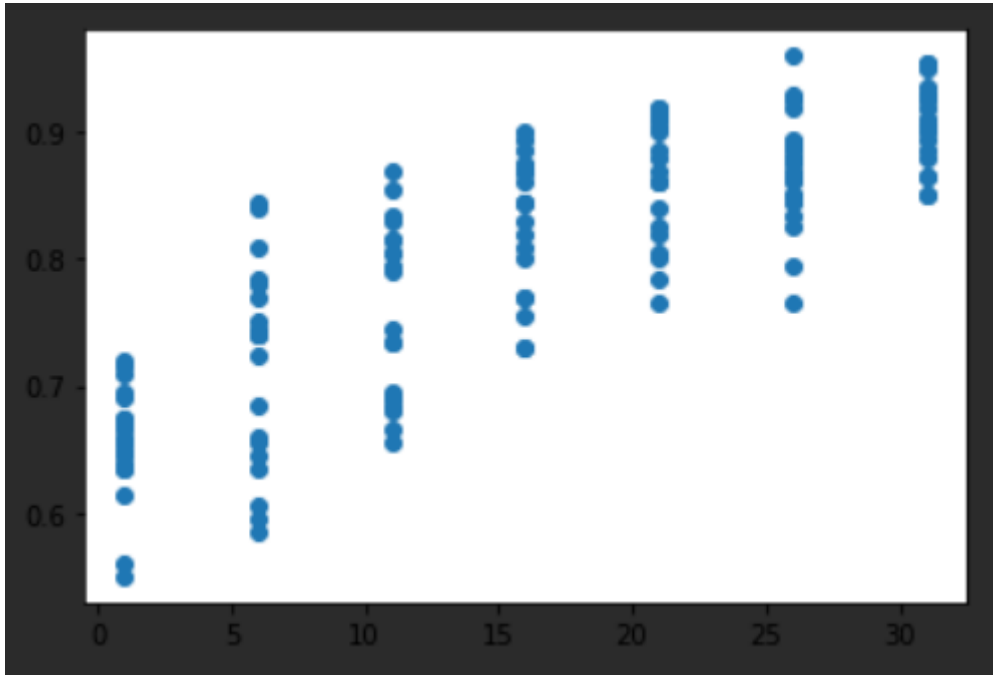


Figure 31: Scatter plot of steps taken for test 1



**Figure 32: Scatter plot of the highest morbidity rate of test 1**

According to the plot above, if there is no additional condition added, the fatality rate of agents floats between 50.0% to 68.0%, and it usually takes more than 220 steps for the COVID outbreak to subside. Besides, as the number of initially infected agents grows, the highest morbidity rate increases from 65% to over 90%, which means almost all the agents have been infected during the epidemic.

### 7.3 Batch run 2 - Wearing face masks

The second test reveals the effect of wearing face masks. The parameters of this test are listed in **Figure 33**. It assumes all the 30 initially infected agents wear face masks and get the changing trend of the fatality rate by increasing the number of the rest of agents who wear face masks. The scatter plots of the results are shown in **Figure 34/35/36**.

```
# Batch run No.2, agents wear masks, no hospital, no social distancing
fixed_params_1 = {
    "width": 50,
    "height": 50,
    "N": 200,
    "M": 30,
    "K": 30,
    "L": 0,
    "hospital_activated": 0,
    "auto_social_distancing": 0,
    "manual_quarantine_lvl": 0
}

variable_params_1 = {
    "J": range(0,171,10)
}

batch_run_1 = BatchRunner(CovidModel,
                           variable_params_1, variable_params_1: {'J': range(0, 171, 10)}
                           fixed_params_1, fixed_params_1: {'width': 50, 'height': 50, 'N': 200, 'M': 30, 'K': 30, 'L': 0, 'hospital_activated': 0, 'auto_social_distancing': 0, 'manual_quarantine_lvl': 0}
                           iterations=20,
                           max_steps=300,
                           model_reporters={"Fatality_Rate": compute_fatality_rate,
                                              "Step_Number": get_step_number,
                                              "Highest_Morbidity_Rate": get_highest_morbidity_rate})
```

Figure 33: Parameter for test 2

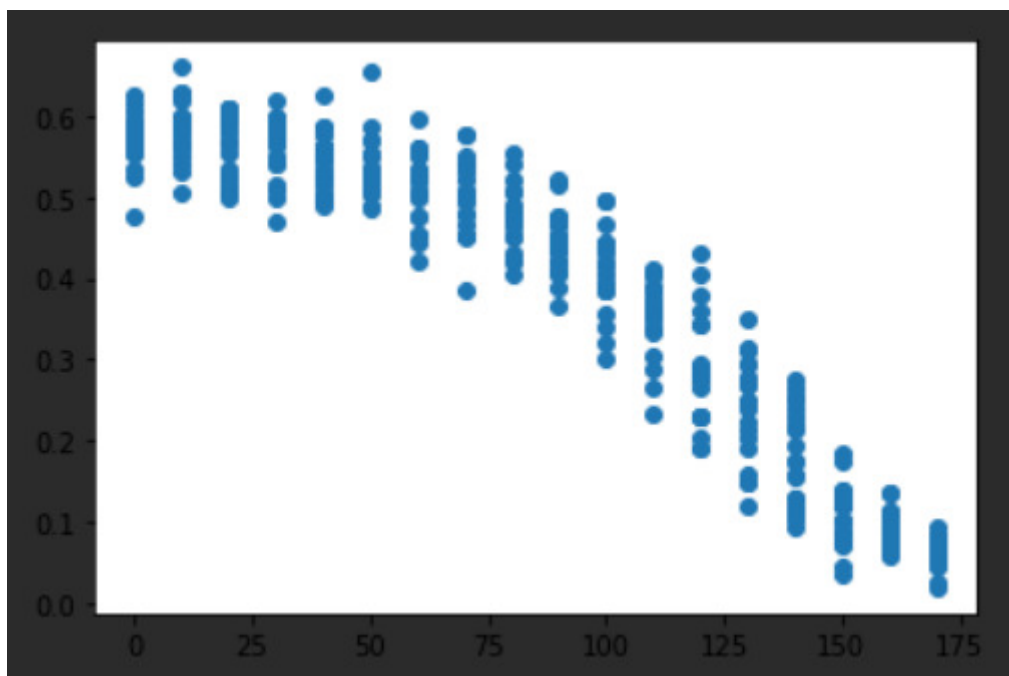
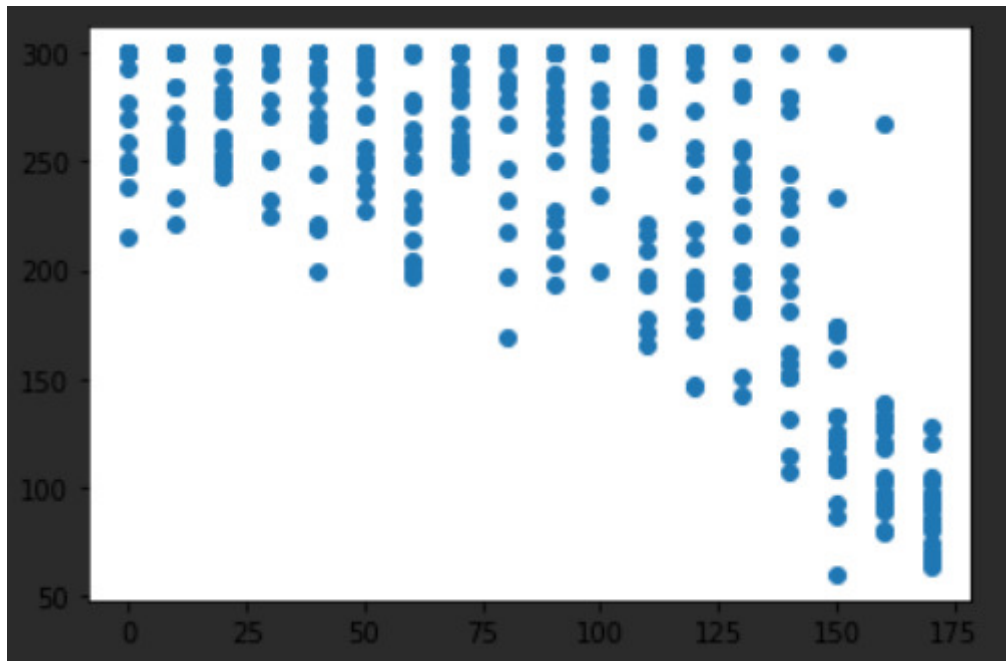
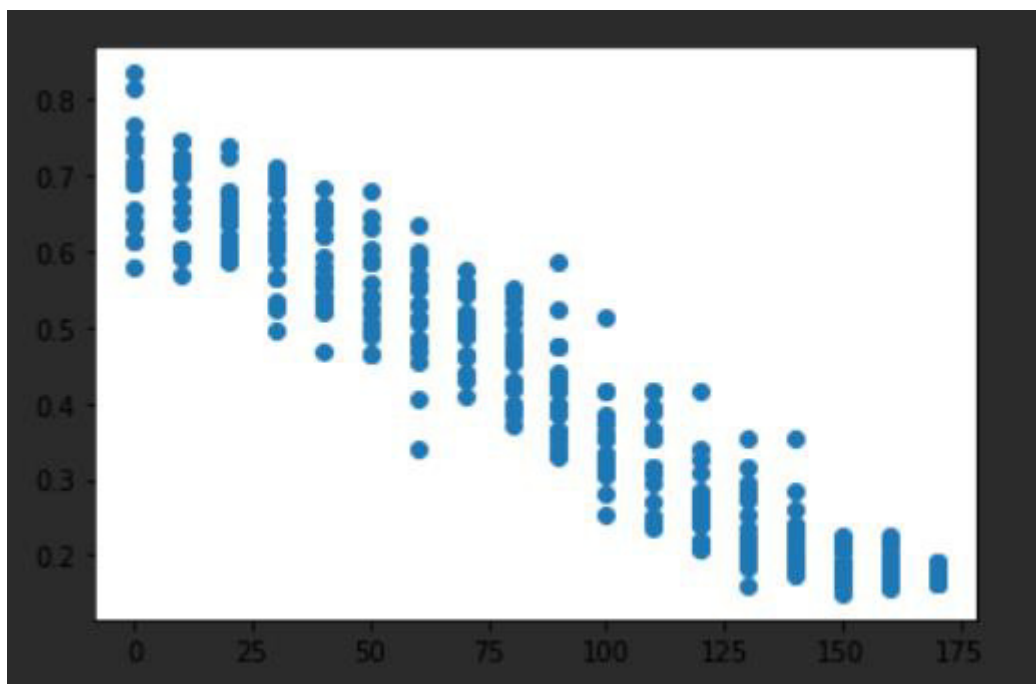


Figure 34: Scatter plot of fatality rate of test 2



**Figure 35: Scatter plot of steps taken for test 2**



**Figure 36: Scatter plot of the highest morbidity rate of test 2**

According to the test results, wearing face masks has a positive impact on controlling the outbreak. When over half of the population wear masks, the fatality rate starts to drop and finally reaches around 7%. Moreover, when the total number of agents who wear masks reaches 190, in other words, when over 95% of agents wear face masks, the number of steps to eliminate the epidemic suddenly drops below 150 steps.

Moreover, the advantage of face masks also shows in the morbidity rate, as the number of agents who wear masks grows, the highest morbidity rate drops linearly from 75% to only 18%. Most of the agents have not been infected before the situation is under control.

## 7.4 Batch run 3 - Hospital activated

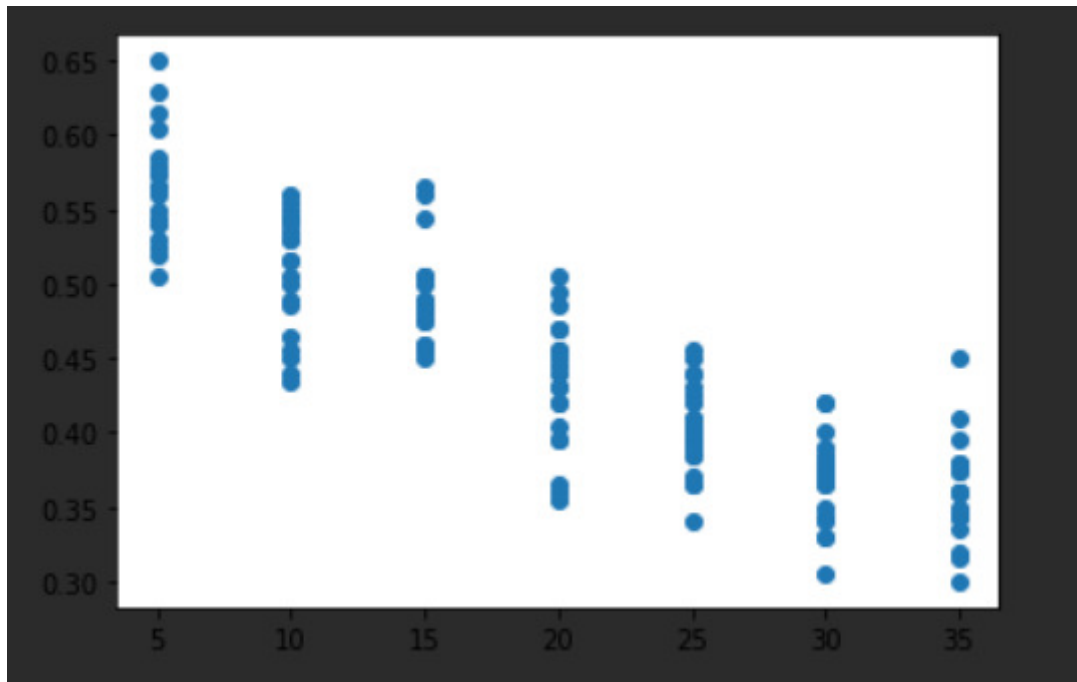
This test reveals the effect of establishing a hospital. As declared in section 6.2.4, it is assumed that the hospital does not accelerate the recovery in this model, it is more like setting up a quarantine zone. The code for this test is displayed in **Figure 37**, in which the capacity of the hospital is assigned as the only variable. The scatter plots of the results are shown in **Figure 38/39/40**.

```
# Batch run No.3, no agents wear masks, hospital activated, no social distancing
fixed_params_2 = {
    "width": 50,
    "height": 50,
    "N": 200,
    "M": 30,
    "J": 0,
    "K": 0,
    "hospital_activated": 1,
    "auto_social_distancing": 0,
    "manual_quarantine_lvl": 0
}

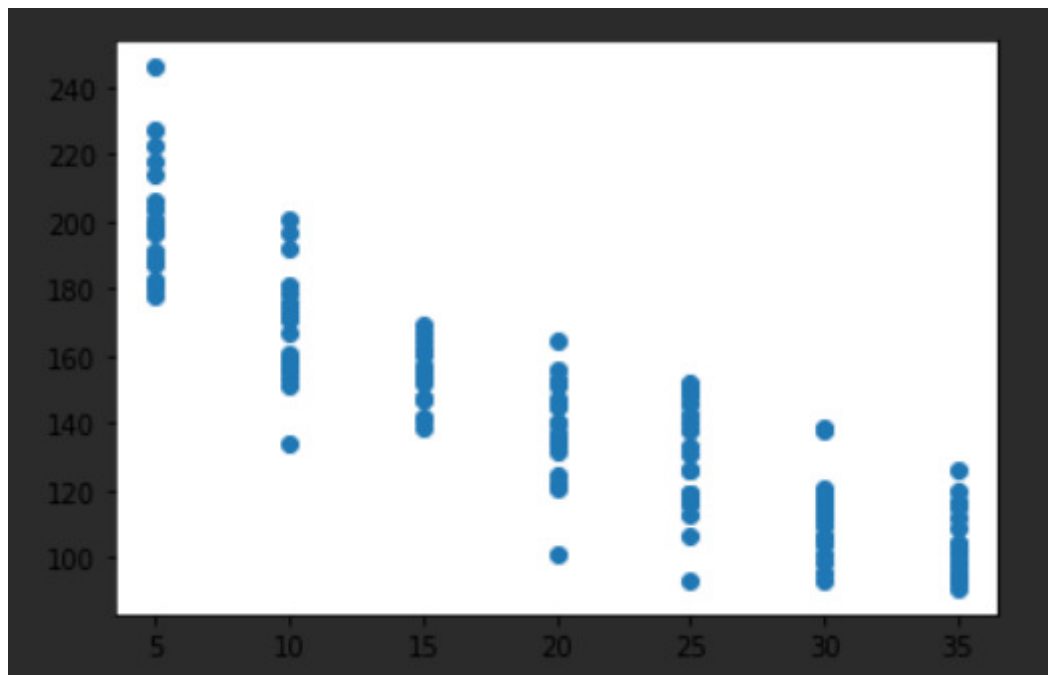
variable_params_2 = {
    "L": range(5, 36, 5)
}

batch_run_2 = BatchRunner(CovidModel,
                           variable_params_2, variable_params_2: {'L': range(5, 36, 5)}
                           fixed_params_2, fixed_params_2: {'width': 50, 'height': 50, 'N': 200, 'M': 30, 'J': 0, 'K': 0, 'hospital_activated': 1, 'auto_social_distancing': 0, 'manual_quarantine_lvl': 0}
                           iterations=20,
                           max_steps=300,
                           model_reporters={"Fatality_Rate": compute_fatality_rate,
                                              "Step_Number": get_step_number,
                                              "Highest_Morbidity_Rate": get_highest_morbidity_rate})
```

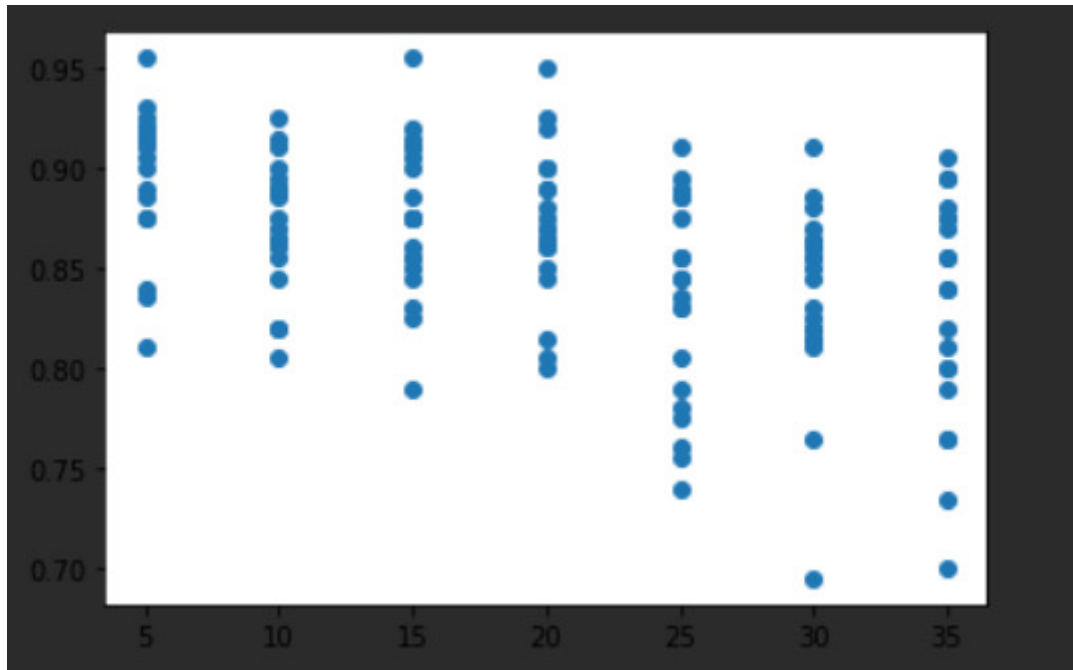
**Figure 37: Parameters for test 3**



**Figure 38: Scatter plot of fatality rate of test 3**



**Figure 39: Scatter plot of steps taken for test 3**



**Figure 40: Scatter plot of the highest morbidity rate of test 3**

We can learn from the plots that both the fatality rate and the steps show a dropping trend as the capacity of the hospital rises. When the capacity of the hospital get to 35 (17.5% of the population), the fatality rate is lowered by 20%, and it only takes about 110 steps for the model to end the outbreak. However, setting up a quarantine zone(the hospital) does not reduce the highest morbidity rate which is still ranging from 75% to 92.5%. Even though it does help to control the epidemic from spreading, most of the agents have been infected during the process.

## 7.5 Batch run 4 - Flexible social distancing policy

This test has the same parameters as Batch Run 1, except for the automatic social distancing feature, as displayed in **Figure 41**. The auto-mode is on for the test, which means when more than 10% of the agents show symptoms (to simulate the condition that the government detects the outbreak), 60% of the agents will begin self-isolation; when the symptomatic rate reaches 20%, 90% of the agents will be self-isolated. The scatter plots of the results are shown in **Figure 42/43/44**.

```

# Batch run No.4, no agents wear masks, no hospital, auto social distancing activated
fixed_params_3 = {
    "width": 50,
    "height": 50,
    "N": 200,
    "J": 0,
    "K": 0,
    "L": 0,
    "hospital_activated": 0,
    "auto_social_distancing": 1,
    "manual_quarantine_lvl": 0
}

variable_params_3 = {
    "M": range(1, 32, 5)
}

batch_run_3 = BatchRunner(CovidModel,
                           variable_params_3, variable_params_3: {'M': range(1, 32, 5)}
                           fixed_params_3, fixed_params_3: {'width': 50, 'height': 50, 'N': 200, 'J': 0, 'K': 0, 'L': 0, 'hospital_activated': 0, 'auto_social_distancing': 1, 'manual_quarantine_lvl': 0}
                           iterations=20,
                           max_steps=300,
                           model_reporters={"Fatality_Rate": compute_fatality_rate,
                                              "Step_Number": get_step_number,
                                              "Highest_Morbidity_Rate": get_highest_morbidity_rate})

```

Figure 41: Parameters for test 4

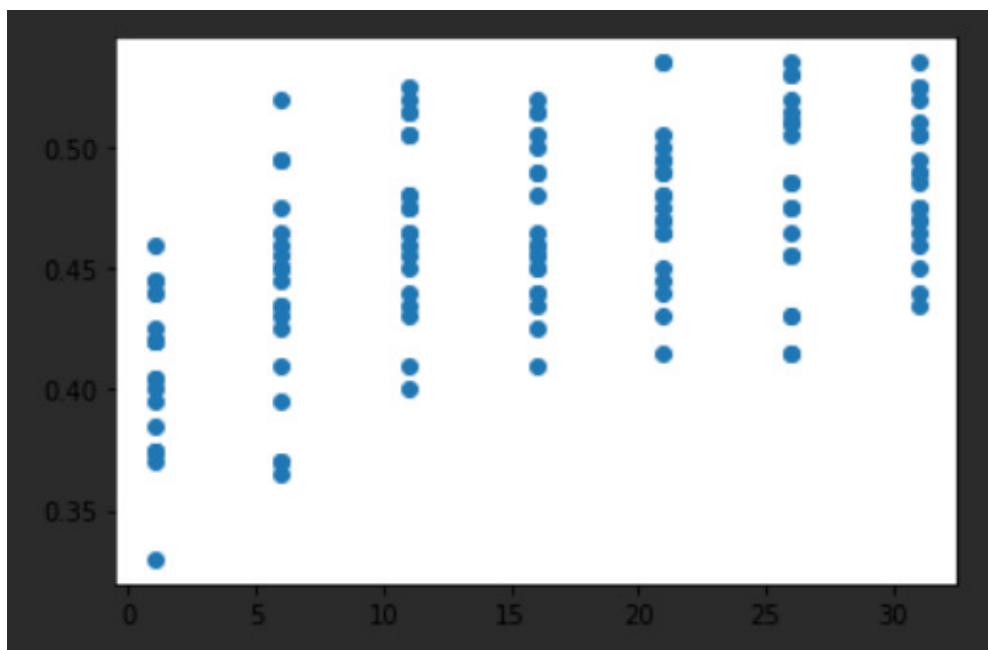
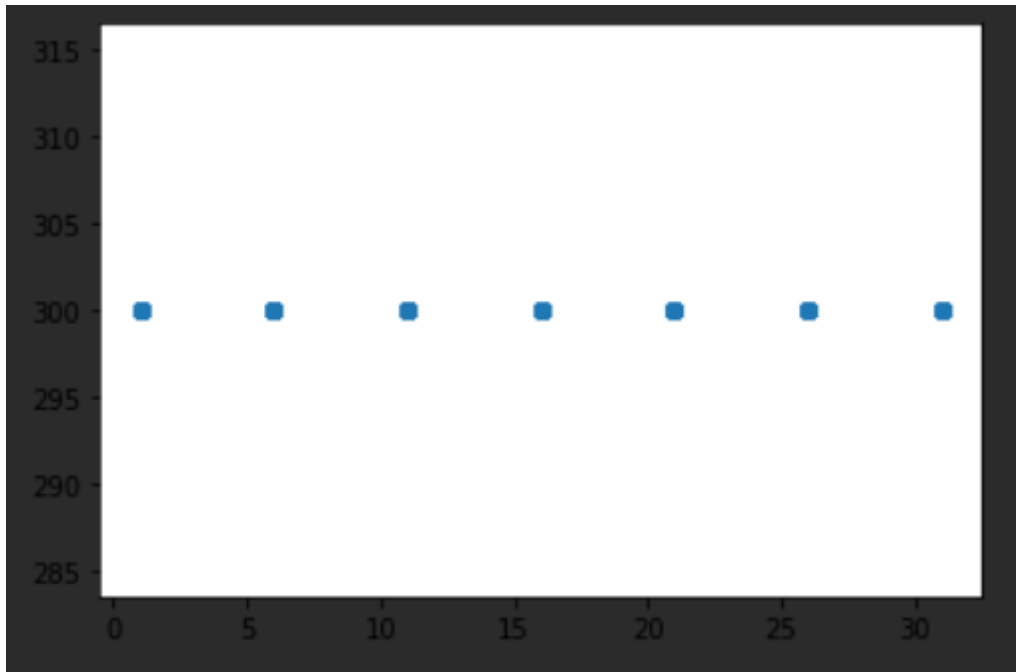
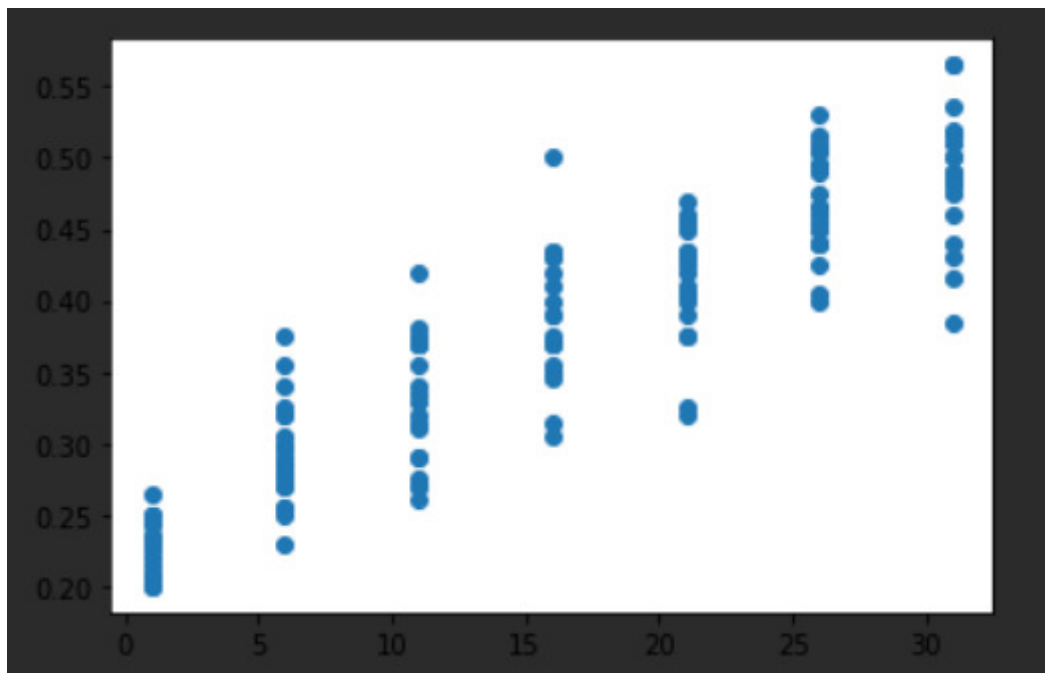


Figure 42: Scatter plot of fatality rate of test 4





**Figure 43: Scatter plot of steps taken for test 4**



**Figure 44: Scatter plot of the highest morbidity rate of test 4**

The results do not look bright. Even though the fatality rate seems to decrease by 10% than the first test, it should be noticed that all the 140 tests have reached the maximum number of steps, which means during all these tests, the epidemic situation has not ended within 300 steps. The flexible policy only temporarily postpones the spreading of COVID-19; there are still agents infected in the model. The linearly growing

morbidity rate is also telling the same conclusion: as time goes by, the fatality rate will rise as more agents will die from the virus.

## 7.6 Batch run 5 - strict social distancing policy

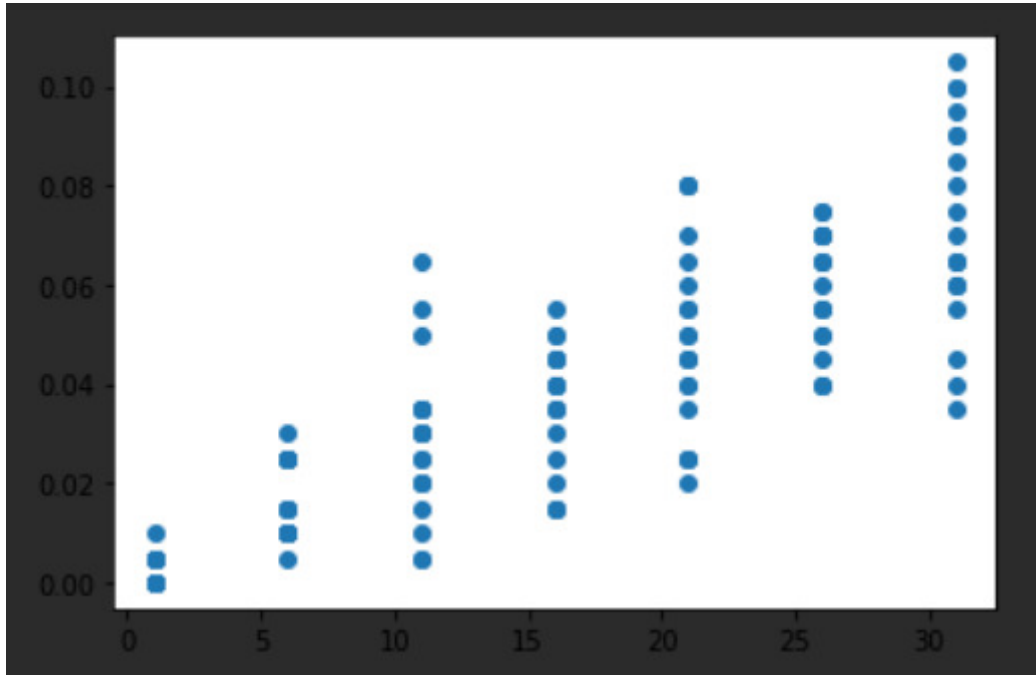
This test is similar to the last one, but with a stringent social-distancing level that 90% of the agents are self-isolated from the very beginning of the simulation to the end. It is actually an ideal assumption to examine the effect of a strict social-distancing policy. The parameters set are shown in **Figure 45**, and the results are displayed in **Figure 46/47/48**.

```
# Batch run No.5, no agents wear masks, no hospital, no social distancing activated
# Manual social distancing is switched to level 2
fixed_params_4 = {
    "width": 50,
    "height": 50,
    "N": 200,
    "J": 0,
    "K": 0,
    "L": 0,
    "hospital_activated": 0,
    "auto_social_distancing": 0,
    "manual_quarantine_lvl": 2
}

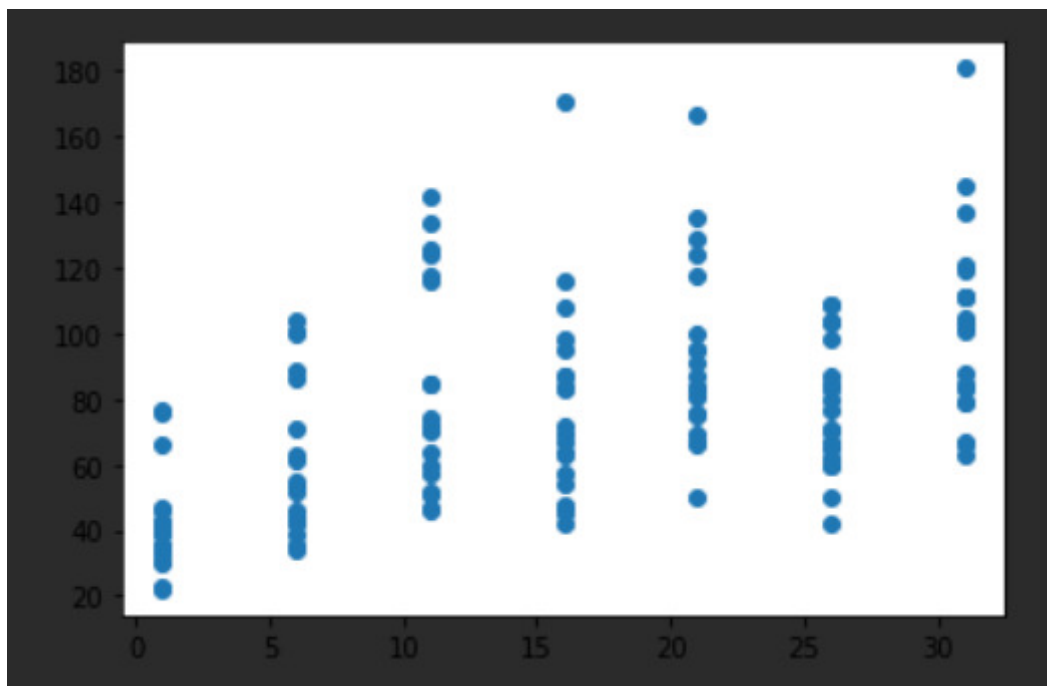
variable_params_4 = {
    "M": range(1, 32, 5)
}

batch_run_4 = BatchRunner(CovidModel,
                           variable_params_4, variable_params_4: {'M': range(1, 32, 5)}
                           fixed_params_4, fixed_params_4: {'width': 50, 'height': 50, 'N': 200, 'J':
                           iterations=20,
                           max_steps=300,
                           model_reporters={"Fatality_Rate": compute_fatality_rate,
                                             "Step_Number": get_step_number,
                                             "Highest_Morbidity_Rate": get_highest_morbidity_rate})
```

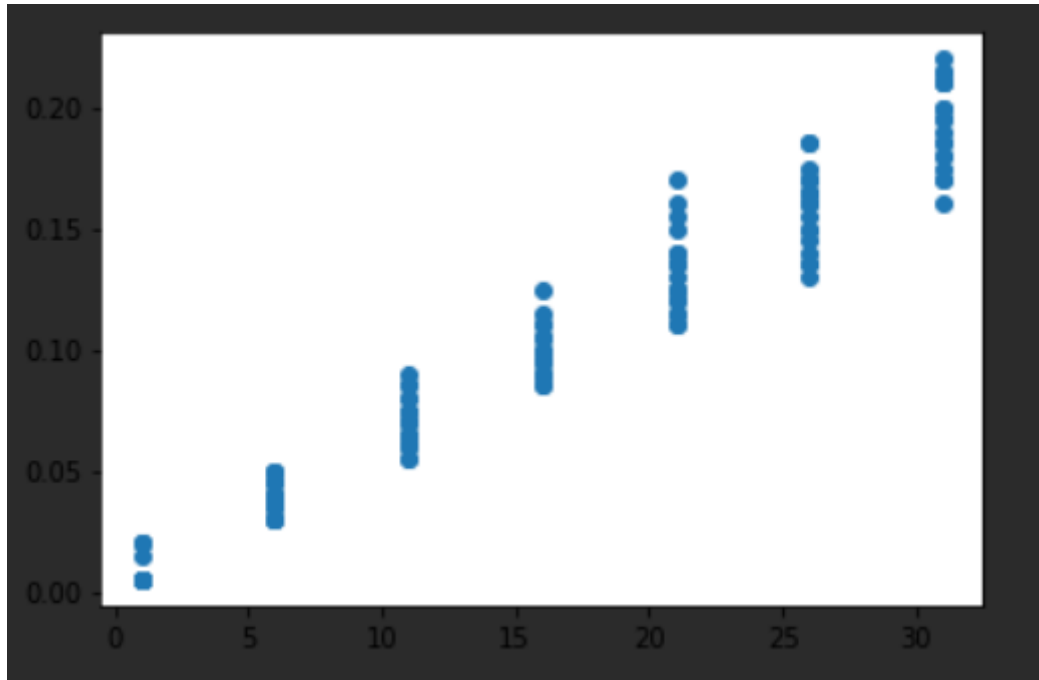
Figure 45: Parameters for test 5



**Figure 46: Scatter plot of fatality rate of test 5**



**Figure 47: Scatter plot of steps taken for test 5**



**Figure 48: Scatter plot of the highest morbidity rate of test 5**

The results explain that this policy does have an influential impact on controlling the epidemic. The fatality rate plummets to the level below 10%, and for most of the tests, it only takes about 50 to 130 steps for the outbreak to be eliminated. Furthermore, even with 30 infected agents in the grid, the highest morbidity rate stops at about 22%, which indicates that the strict policy does prevent the virus from spreading.

## 7.7 Batch run 6 - All measures adopted

The last test simulates an optimistic situation in which every agent wears a face mask, and there is a hospital with increasing capacity set as the variable for the X-axis. Moreover, to simulate the real-world situation, the automatic social distancing policy is switched on as well. All the parameters are coded in **Figure 49**. The scatter plots of the results are shown in **Figure 50/51/52**.

```

# Batch run No.6, agents wear masks, hospital activated, auto social distancing activated
fixed_params_5 = {
    "width": 50,
    "height": 50,
    "N": 200,
    "M": 30,
    "J": 170,
    "K": 30,
    "hospital_activated": 1,
    "auto_social_distancing": 1,
    "manual_quarantine_lvl": 0
}

variable_params_5 = {
    "L": range(5, 36, 5),
}

batch_run_5 = BatchRunner(CovidModel,
                           variable_params_5, variable_params_5: {'L': range(5, 36, 5)}
                           fixed_params_5, fixed_params_5: {'width': 50, 'height': 50, 'N': 200, 'M': 30, 'J': 170, 'K': 30, 'hospital_activated': 1, 'auto_social_distancing': 1, 'manual_quarantine_lvl': 0}
                           iterations=20,
                           max_steps=300,
                           model_reporters={"Fatality_Rate": compute_fatality_rate,
                                              "Step_Number": get_step_number,
                                              "Highest_Morbidity_Rate": get_highest_morbidity_rate})

```

Figure 49: Parameters for test 6

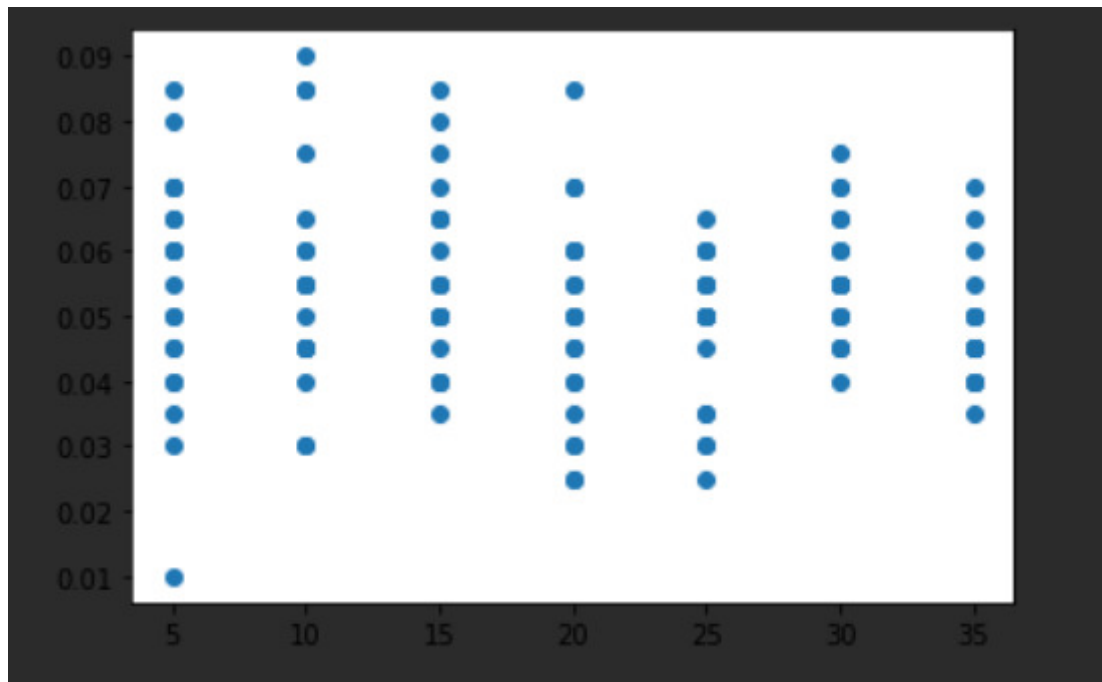
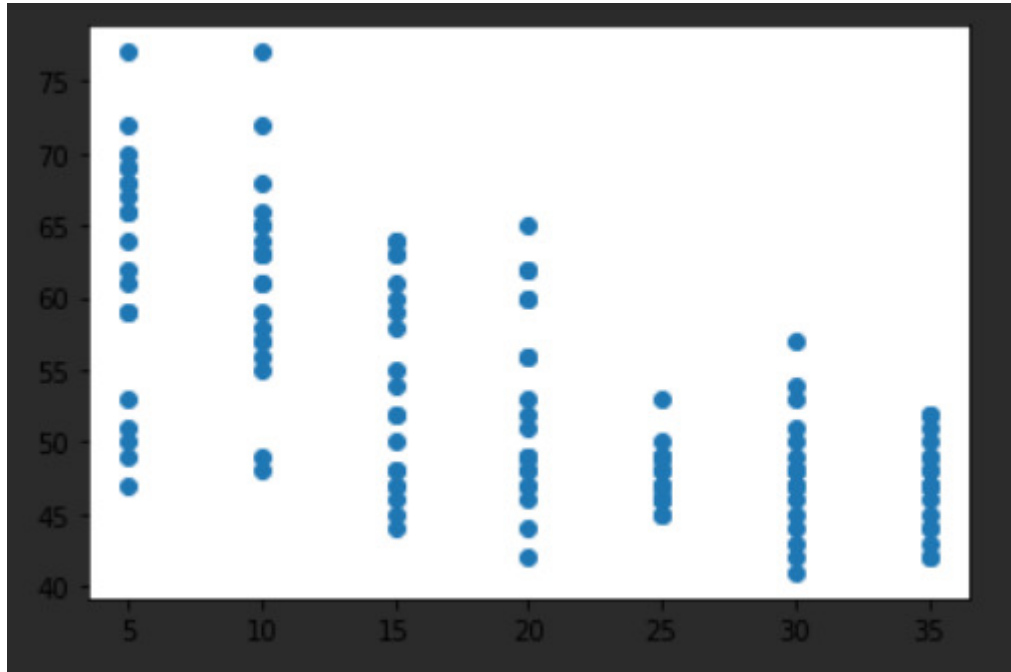
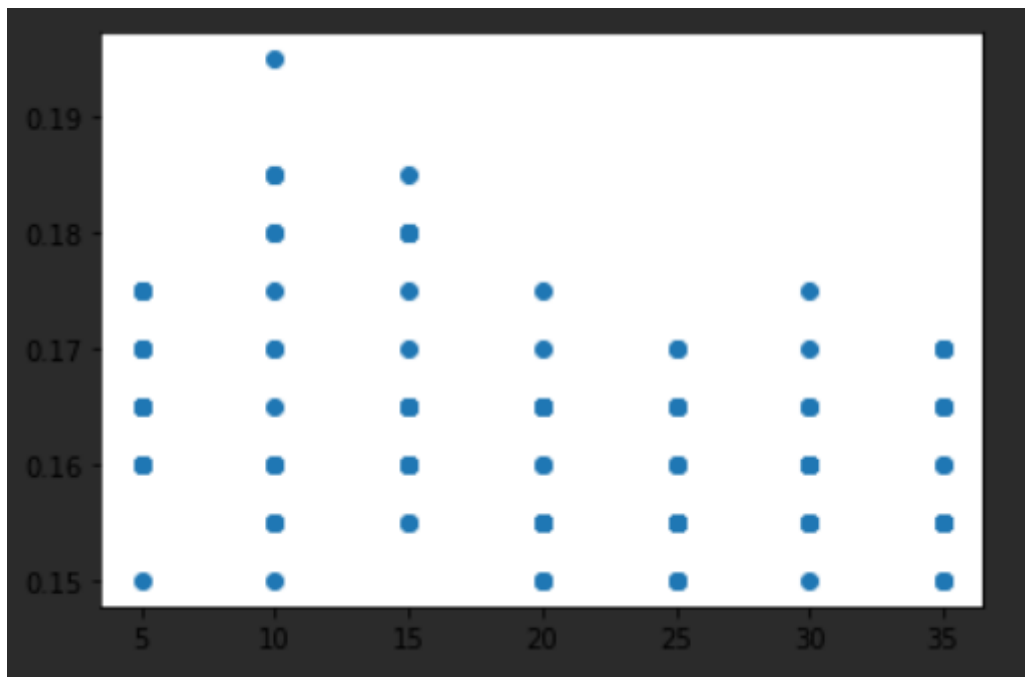


Figure 50: Scatter plot of fatality rate of test 6



**Figure 51: Scatter plot of steps taken for test 6**



**Figure 52: Scatter plot of the highest morbidity rate of test 6**

With all the measures taken, this test presents the best result so far. The fatality rate has dropped below 9%, and the number of steps to end the COVID-19 outbreak is down to about 75 steps and even less. When the hospital capacity reaches 35 (17.5%), the model eliminates the epidemic situation in less than 55 steps. The highest morbidity rate is also steadily below 20%, which looks quite promising.

## Chapter 8 – Conclusion

The COVID-19 epidemic has been the most important event happened in 2020. It has caused millions of death, and exceedingly impacted the global economy and people's life. This project has built a simplified ABM model and formulate a few controlled experiments to preliminarily validate the influence of the measures adopted for controlling the outbreak. According to the results, wearing face masks, establishing quarantine zones and proceeding proper social-distancing policy have all shown an advantageous effect regarding the issue. Furthermore, the simulation shows better results when these methods stack with each other. These tests have shown their significance for reference. As a typical example, the result of the fourth test is quite similar to the second wave coming to some of the European countries in September 2020. Nations like the United Kingdom have adopted a social-distancing policy from May to the end of August, and the situation did grow better for a short period. Nevertheless, once the government loosened the lockdown level, the number of confirmed cases suddenly rose again in September 2020(Triggle 2020). Accordingly, researchers could take this project as a reference for further lucubration. The readers, on the other hand, could learn from the simulation and raise the awareness of prevention. In addition, there is still plenty of room for improvement of this model; factors like personal health condition, history of respiratory illness and different daily routine have not been taken into consideration. Agents could be categorised into groups to simulate people with different identities. Geographical modification regarding the grid is also feasible as there could be individual sectors set as school, hospital, cinema and restaurant. The framework Mesa could also use some upgrade on its source code to improve the visualization module and add more features.

## Chapter 9 – Reflection

"Making study serves the practical purpose." is a Chinese idiom which I took as my life motto. For this dissertation project, I have used every single thing I learnt from the lessons in the MSc Computing programme.

Merely understanding the basic syntax of the programming languages is just a beginning. The first essential ability is to read and learn from other people's project. In this case, of course, it refers to the Mesa framework. Although Mesa has provided the user with a tutorial on its official website, there are a lot of brilliant features hidden in the source code. I have to read through all the code behind the scene to figure out which I should use to construct the ABM of my own, and there are, indeed, some tricky details which require extra attention. For example, for the "space.py" of the Mesa source code, there are two methods named `get_neighborhood()` and `get_neighbors()` which look similar but return different values; `get_neighborhood()` returns a list of cells of a certain point, while `get_neighbors()` returns a list of non-None objects of a given coordinate. Distinctions like this helped me to gain a better understanding of the concepts defined in the framework such as cell, grid and schedule, and of course, led me to the correct coding.

The other ability sharpened during the project is always to keep thinking and upgrade the current work. I will take the configuration module as an example. At first, I hard-coded all the parameters into a file named "const.py". Considering the possibility of releasing the project for other users to share in the future, it was not enough using pure Python to achieve the configuring feature. Therefore I started to go through the official documentation of Python and then found one module which was precisely what I needed: `ConfigParser`. I not only learnt the instruction from the documentation but also studied through the material of how ".ini" file functions in the system. After I finished `config.py`, the idea of "why not make a GUI like what you did in the CMT303 Software Engineering class" came into my mind. This time I decided to use the tool I was familiar with, Tkinter, to perfect this feature. All the work mentioned above formed the final



version of `gui_config.py`, which is a lot more user-friendly.

The last thing I would like to mention is to be creative and have fun. I have watched a few ABM projects on GitHub made by other developers after I started coding, and they all designed their simulations with simple geometric shapes like circle, rectangle or dot, which looks quite dull to me. So, I decided to add some fun elements to draw my users' attention and improve their user experience. Emoji was the first thing I considered, but it was still too colourful and intricate which could cause visual ambiguity; I eventually picked the stick-drawing icons with a single colour as my way to distinguish the agents in different states. This move did support me with fun every time I run the simulation. I could now proudly say that I completed the project with pleasure.

However, due to the limited time given, and in some sense, my laziness, I still left some work undone. The Mesa module is still under development, and its visualisation component is built based on a template of the Tornado framework, which makes it a bit disfigured without further modification on the layout and interaction. It was one of the things I should have fixed. Furthermore, I should have added geographical boundary for the feature to construct certain areas in the grid. The lack of awareness in time management and self-discipline leaves the work unfinished as a pity. But it has become a valuable lesson for my future career: time slips through our fingers like sand, and we must make the best of it.

## References

- Andersen, K.G. et al. 2020. The proximal origin of SARS-CoV-2. *Nature Medicine* 26(4). doi: 10.1038/s41591-020-0820-9.
- Cai, J. et al. 2020. Early Release - Indirect Virus Transmission in Cluster of COVID-19 Cases, Wenzhou, China, 2020 - Volume 26, Number 6—June 2020 - *Emerging Infectious Diseases journal - CDC*. [wwwnc.cdc.gov](http://wwwnc.cdc.gov/eid/article/26/6/20-0412_article) 26(6). Available at: [https://wwwnc.cdc.gov/eid/article/26/6/20-0412\\_article](https://wwwnc.cdc.gov/eid/article/26/6/20-0412_article).
- CDC 2020. Coronavirus Disease 2019 (COVID-19) – Symptoms. Available at: <https://www.cdc.gov/coronavirus/2019-ncov/symptoms-testing/symptoms.html>.
- Chin, S. 2013. Overview. Available at: <http://flame.ac.uk/docs/overview.html> [Accessed: 9 October 2020].
- Clark, D. 2020. UK: population, by age 2018 | Statista. Available at: <https://www.statista.com/statistics/281174/uk-population-by-age/> [Accessed: 13 October 2020].
- Correction to *Lancet Infect Dis* 2020; published online March 27. [https://doi.org/10.1016/S1473-3099\(20\)30200-0](https://doi.org/10.1016/S1473-3099(20)30200-0). 2020. *The Lancet Infectious Diseases* 20(6), p. e116. doi: 10.1016/s1473-3099(20)30394-7.
- Criteria for releasing COVID-19 patients from isolation. 2020. Available at: <https://www.who.int/news-room/commentaries/detail/criteria-for-releasing-covid-19-patients-from-isolation>.
- Ganyani, T. et al. 2020. Estimating the generation interval for coronavirus disease (COVID-19) based on symptom onset data, March 2020. *Eurosurveillance* 25(17). doi: 10.2807/1560-7917.es.2020.25.17.2000257.
- Goldstein, J.R. and Lee, R.D. 2020. Demographic perspectives on the mortality of COVID-19 and other epidemics. *Proceedings of the National Academy of Sciences* 117(36), p. 202006392. Available at: <https://www.pnas.org/content/pnas/early/2020/08/19/2006392117.full.pdf> [Accessed: 29 August 2020].
- Grant, M.C. et al. 2020. The prevalence of symptoms in 24,410 adults infected by the novel coronavirus (SARS-CoV-2; COVID-19): A systematic review and meta-

- analysis of 148 studies from 9 countries. Hirst, J. A. ed. PLOS ONE 15(6), p. e0234765. doi: 10.1371/journal.pone.0234765.
- Hamner, L. et al. 2020. High SARS-CoV-2 Attack Rate Following Exposure at a Choir Practice — Skagit County, Washington, March 2020. MMWR. Morbidity and Mortality Weekly Report 69(19), pp. 606–610. doi: 10.15585/mmwr.mm6919e6.
- Hoertel, N. et al. 2020. A stochastic agent-based model of the SARS-CoV-2 epidemic in France. Nature Medicine 26(9), pp. 1417–1421. doi: 10.1038/s41591-020-1001-6.
- Huang, C. et al. 2020. Clinical features of patients infected with 2019 novel coronavirus in Wuhan, China. The Lancet 395(10223). Available at: [https://www.thelancet.com/journals/lancet/article/PIIS0140-6736\(20\)30183-5/fulltext](https://www.thelancet.com/journals/lancet/article/PIIS0140-6736(20)30183-5/fulltext).
- Jackson, J.C. et al. 2017. Agent-Based Modeling. Social Psychological and Personality Science 8(4), pp. 387–395. doi: 10.1177/1948550617691100.
- La Rosa, G. et al. 2020. Coronavirus in water environments: Occurrence, persistence and concentration methods - A scoping review. Water Research 179, p. 115899. doi: 10.1016/j.watres.2020.115899.
- La Scola, B. et al. 2020. Viral RNA load as determined by cell culture as a management tool for discharge of SARS-CoV-2 patients from infectious disease wards. European Journal of Clinical Microbiology & Infectious Diseases 39(6), pp. 1059–1061. doi: 10.1007/s10096-020-03913-9.
- Leclerc, Q.J. et al. 2020. What settings have been linked to SARS-CoV-2 transmission clusters? Wellcome Open Research 5(83), p. 83. Available at: <https://wellcomeopenresearch.org/articles/5-83>.
- Liu, T. et al. 2020. Prevalence of IgG antibodies to SARS-CoV-2 in Wuhan - implications for the ability to produce long-lasting protective antibodies against SARS-CoV-2. doi: 10.1101/2020.06.13.20130252.
- Luke, S. et al. 2003. MASON Multiagent Simulation Toolkit. Available at: <https://cs.gmu.edu/~eclab/projects/mason/> [Accessed: 9 October 2020].
- Nguyen, L.H. et al. 2020. Risk of COVID-19 among front-line health-care workers and the general community: a prospective cohort study. The Lancet Public Health 5(9). Available at: [https://www.thelancet.com/journals/lanpub/article/PIIS2468-2667\(20\)30164-X/fulltext](https://www.thelancet.com/journals/lanpub/article/PIIS2468-2667(20)30164-X/fulltext).

Parri, N. et al. 2020. Children with Covid-19 in Pediatric Emergency Departments in Italy. *New England Journal of Medicine* . doi: 10.1056/nejmc2007617.

Project Mesa Team 2016. Mesa Overview — Mesa .1 documentation. Available at: <https://mesa.readthedocs.io/en/master/overview.html> [Accessed: 10 October 2020].

Solovyev, A. 2010. SPARK - Simple Platform for Agent-based Representation of Knowledge. Available at: <http://www.pitt.edu/~cirm/spark/> [Accessed: 9 October 2020].

Triggle, N. 2020. Covid: The second wave is here - but how bad will it be? BBC News 1 October. Available at: <https://www.bbc.com/news/health-54362994> [Accessed: 15 October 2020].

Wajnberg, A. et al. 2020. Humoral immune response and prolonged PCR positivity in a cohort of 1343 SARS-CoV 2 patients in the New York City region. doi: 10.1101/2020.04.30.20085613.

Wilensky, U. 1999. NetLogo Home Page. Available at: <https://ccl.northwestern.edu/netlogo/> [Accessed: 16 October 2020].

Xu, C. 2019. Icon Packs Designed by ColinXu. Available at: <https://www.iconfont.cn/collections/detail?spm=a313x.7781069.1998910419.dc64b3430&cid=4116> [Accessed: 11 October 2020].