BIP Kreativitätsgymnasium Leipzig

BESONDERE LERNLEISTUNG (BeLL)
Abiturjahrgang 2021/22

Simulation of Local Epidemics Using Graphs

Janis Lennart Kupfer

Innenbetreuer:	Matthias Richter
Außenbetreuer:	Dr. rer. nat. Gunnar Wichmann Laborleiter HNO-Forschungslabor Klinik für Hals-, Nasen-, Ohrenheilkunde/Plastische Chirurgie Medizinische Fakultät, Universität Leipzig
Abgabedatum:	22.12.2021

Eidesstattliche Erklärung

Ich erkläre hiermit eidesstattlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Die aus fremden Quellen übernommenen Gedanken sind als solche kenntlich gemacht. Ich bin mir bewusst, dass eine unwahre Erklärung rechtliche Folgen haben kann.

Ort, Datum Unterschrift

Contents

1 Introduction

Epidemics and the spreading of diseases has historically been a very important issue. As diseases continue to cause damage to the population affected. Therefore, being able to both model and predict the course of an epidemic is vital. Through the science of epidemiology, it is already possible to mathematically model such. As a result, mathematical models can be used to predict the possible outcome of an epidemic. As diseases may differ in certain attributes, such as the incubation period or acquisition of immunity, different epidemiological models are used.

The application developed in this work, in turn, uses a different method of predicting the outcome. Specifically, it aims to do so by simulating both populations and spreading of the disease. The population in a simulation is generated randomly. People are seen as vertices in a graph, while the connections between them are edges. A population in the application is therefore seen as a graph. Using multiple random simulations, such a program should return similar results to mathematically proven models. In order to be used generally, the parameters used in the application are rather vague, so that all most commonly used epidemiological models and even diseases and occurrences not accounted for in such models may be simulated.

In this work, the general theoretical baselines will be discussed first. This includes epidemiological and information technology baselines. Then, the application and its structure will be explained, starting with the general structure, and then following with the model, view model and view layers. Finally, this work will discuss the results gained through simulating various different models with varying inputs, ending with an evaluation and possible further development.

2 Theoretical Baselines

2.1 Epidemiological Baselines

“Epidemiology is the study of how disease is distributed in populations and of the factors that influence or determine this distribution.” (Celentano, Szklo 2019, 2) Staffan E. Norell states in his book “A Short Course in Epidemiology” that “Epidemiology is the study of disease occurrence and includes the study of associations between disease and characteristics of individuals and their environments.” (Norell 1992, 1)

Whereas the first definition describes epidemiology as the distribution among a general population, the second definition also entails the characteristics of individuals. Another source states that the most important difference between epidemiology and other sciences centered around diseases is the focus on groups of patients instead of single individuals (see Weiß 2019, 202). The aims of epidemiology are studying how diseases spread, including patterns of spreading and factors leading to the spreading of diseases, the study of the natural course of a disease, their prognosis (including changing characteristics over time) as well as the evaluation of and developing measurements and models to allow for a reliable assessment of actions undertaken against the disease. Therefore, epidemiology is closely related to other sciences to properly interpret these results, and information technology has now become an important part of epidemiology, since the great amount of data for multiple factors influencing the outcome via various routes must be interpreted, and the tools this science provides are often vital for interpretation of complex interactions in different scenarios.

There are three different terms to describe the course of a disease and how it spreads. An endemic is a disease that has a habitual presence in a given geographical area or usually high-frequent occurrence in that area. An epidemic describes the occurrence of a disease in a specific geographical area or community that exceeds the normal expectancy of other such diseases. It is derived from a common or propagated source. A pandemic lastly is a term referencing a worldwide occurrence. (see Celentano, Szklo 2019, 16-17)

2.1.1 Description of the General Course of an Epidemic

Epidemiological models are used most to describe the course of epidemics in a population. There are several compartments (see Brauer, Castillo-Chavez and Feng 2019, 23), which are groups of people in different stages of diseases. Each model consists of different groups and describes ways people move between the groups. There are four distinct compartments or states to which a given individual may belong. These are susceptible, infected, removed and exposed (see Brauer, Castillo-Chavez and Feng 2019, 23).

The susceptible compartment, abbreviated with *S*, describes the number of people that are not yet infected (susceptible, at risk). The infected compartment *I* denotes the number of people who have been infected and can spread the disease. The removed compartment *R* describes the people who have been infected and who cannot contract the disease. This includes people that either have died or people that have gained immunity. Lastly, the exposed compartment *E* denotes people who have been exposed to the disease but are not infectious yet. (see Brauer, Castillo-Chavez and Feng 2019, 23)

Each compartment is modeled as a function, with the number of people in that compartment depending on the time. As people move from one compartment to another, the functions are affected, thereby mathematically modeling the development of the population. This is done with differential equations. “A differential equation contains one or more terms involving derivatives of one variable (the dependent variable, y) with respect to another variable (the in-

dependent variable, x)."[1] The solutions to these are the functions s(t), e(t), i(t) and r(t). Through this, the development of diseases can be mathematically modeled.

2.1.2 The SI-Model

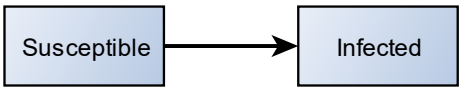

Figure 1: Visualization of the SI model

The SI-model is made up of two main groups. Those are the susceptible *S* and the infectious *I*. The model assumes that there is a certain number of susceptible S_0 people at the start, and a certain number of infectious I_0 people. If someone gets infected, they stay so for the rest of their life. Those then spread the disease among the susceptible.[2]

The mathematical model without vital dynamics is based on the following assumptions.

1. Individuals only transfer from the susceptible into the infectious.
2. In the total population (*N*) there are no deaths nor births.

The basic assumption is that

$$S(t) + I(t) = N$$

The increase of infected $\frac{dI(t)}{dt}$ can be modelled by the differential equation

$$\frac{dI(t)}{dt} = \beta \cdot \frac{S(t)}{N} \cdot I(t)$$

with β: transmission rate. The solution of this equation is the function

$$i(t) = \frac{i_0}{i_0 + (1 - i_0) \cdot e^{-\beta \cdot t}}$$

with $i(t)$: the fraction of infected dependent on time and $i_0 = i(t = 0)$. The fraction $s(t)$ is therefore

$$s(t) = 1 - i(t)$$

[1] Differential Equations. In http://www.maths.surrey.ac.uk/explore/vithyaspages/differential.html [last seen 09/05/2021]
[2] see SI and SIS models. In https://docs.idmod.org/projects/emod-generic/en/latest/model-si.html [last seen 09/05/2021]

2.1.3 The SIS-Model

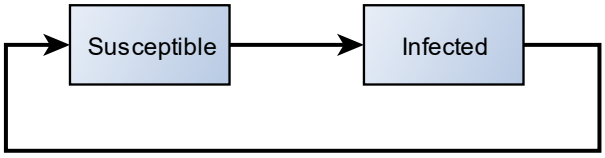

Figure 2: Visualization of the SIS model

In the SIS-model, after being infected, people can again become susceptible and then infected again. Such diseases are often endemic, which creates two different possible models. One includes birth- and death rates in the model, the other does not (see Brauer, Castillo-Chavez and Feng 2019, 23). Because of the recovery and the probability of it, there are two different outcomes to the model. One is a disease-free state. If the recovery is faster than the new infections, the population will eventually become disease-free. The other outcome is the steady state, where the number of infected people is constantly high, and the number of susceptible people is constantly low.[3]

2.1.4 The SIR-Model

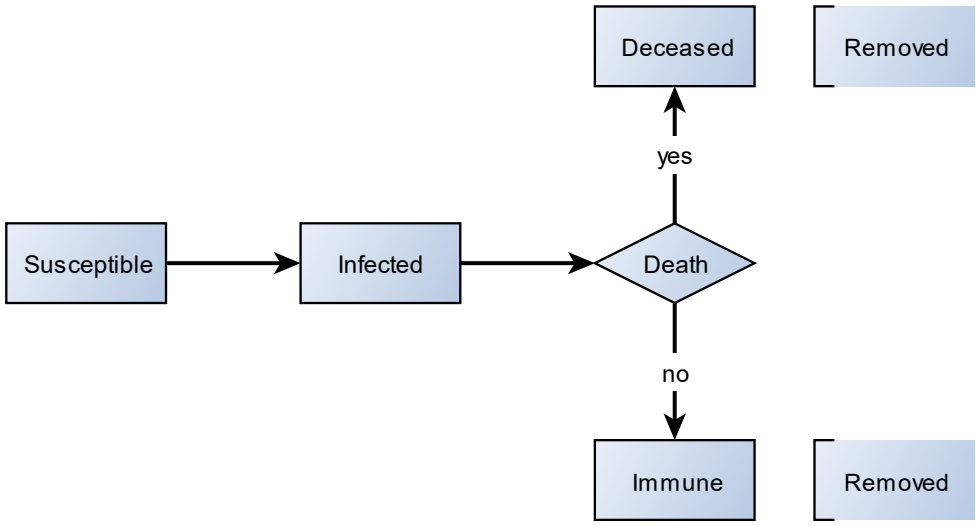

Figure 3:Visualization of the SIR model

The SIR-model makes use of three compartments. The susceptible, the infected and the removed. Subjects in this model move from the susceptible to the infected, ultimately being removed. This removal can be caused either through isolation, immunization, or through death. Because of this, an SIR-model eventually always reaches a disease-free equilibrium, as all people end up being removed, therefore not being able to be infected again. (see Brauer, Castillo-Chavez and Feng 2019, 23)

[3] see SI and SIS models. In https://docs.idmod.org/projects/emod-generic/en/latest/model-si.html [last seen 09/05/2021]

2.1.5 The SIRS-Model

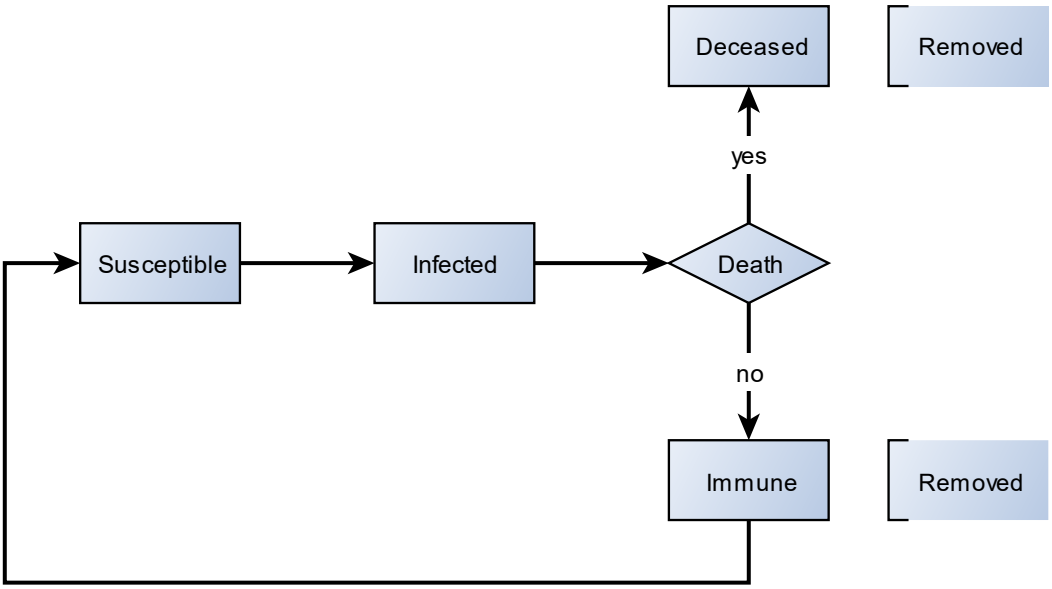

Figure 4:Visualization of the SIRS model

The SIRS-model expands the SIR-model, in that people, after being removed, can become susceptible again. Therefore, the removed compartment represents a temporary immunity. Depending on the rate of infection and recovery, the SIRS-model may have different outcomes. One assumes that more people recover than get infected, so the number of infected people stagnates or even reaches zero. The other, in turn, assumes that the rate of infection is greater than the recovery, so the cases end up switching, with the removed people stagnating or even reaching zero. (see Brauer, Castillo-Chavez and Feng 2019, 23)

2.1.6 The SEIR- and SEIRS-Model

The major difference between these models and the SIR-/SIRS-models is that the group exposed is added as well (see Brauer, Castillo-Chavez and Feng 2019, 23). This however does not change the overall solutions to these models, as it simply adds a phase between infection and the point subjects become infectious.

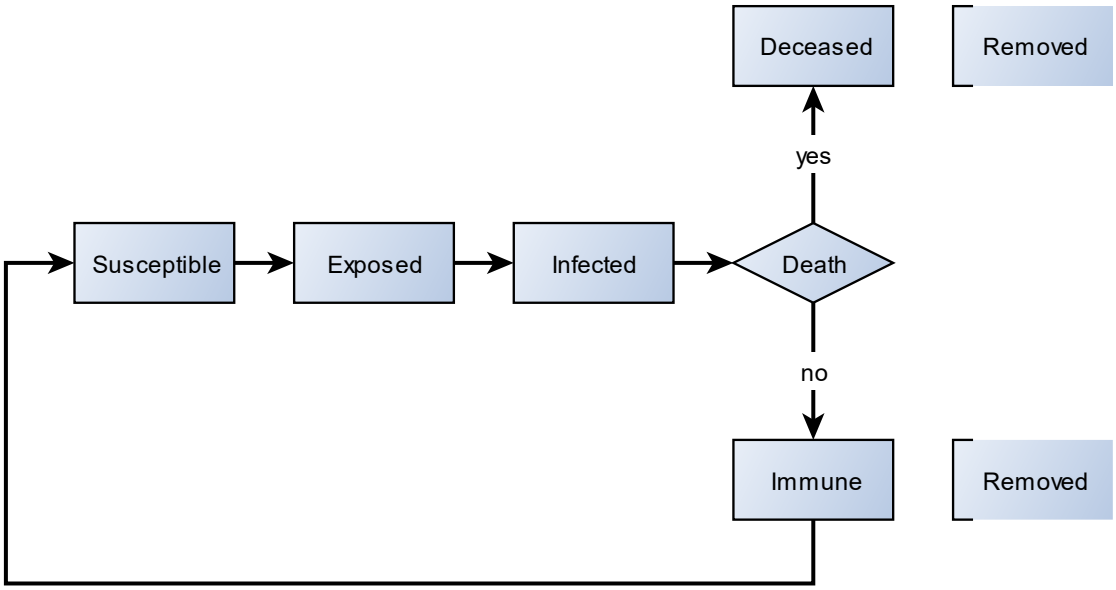

Figure 5:Visualization of the SEIRS model

2.2 Mathematical Baselines

2.2.1 Probability Measures

When looking at a random experiment, the value of the result is called the random variable *X*. The results of the experiment can be modeled using a function, which is then called the probability distribution *P(X)*. The different results for *X* will often revolve around one specific value. This value is then called the mean value. The standard deviation is used to express how much the other values differ from the mean value on average. (see Papula 2001, 316)

2.2.2 The Binomial Distribution

A Bernoulli experiment is a random experiment that has only two outcomes. Therefore, if the outcome *A* has a probability of p, then the counter outcome $\bar{A}$ has a probability of $q = 1 - p$. A Bernoulli process assumes that this experiment is done multiple times independently of each other. The number of times a certain outcome has occurred then becomes the random variable when executing the experiment *n* times. The resulting probability distribution is called a binomial distribution. In this case, X is a natural number. (see Papula 2001, 346)

This function is expressed as follows:

$$F(x) = P(X \leq x) = \sum_{k \leq x} \binom{n}{k} \; p^k \cdot q^{n-k}$$

P(X ≤ x) describes the probability of the event *A* occurring at most *x* times. *X* describes how often the event *A* occurs in total. The value *p* represents the probability of event *A* occurring in a single experiment. *q*, in turn, represents the probability of the counter outcome $\bar{A}$ occurring. Lastly, *n* is the number of times the experiment is repeated. One example for a binomial distribution used in the program is the number of contacts a person has. The number of contacts varies between people in the population, but it is always a natural number. (see Papula 2001, 352)

2.2.3 The Normal Distribution

Unlike the binomial distribution, the normal distribution has a rational number as its random variable. The probability of one specific outcome is always zero. Instead, the normal distribution contains information about the probability of the outcome existing between two values. (see Papula 2001, 367)

The function is modeled as follows:

$$F(x) = P(-\infty < X \leq x) = \frac{1}{\sqrt{2\pi} \cdot \sigma} \cdot \int_{-\infty}^{x} e^{-\frac{1}{2}\left(\frac{t-\mu}{\sigma}\right)^2} dt$$

The parameter μ represents the mean value, the parameter $\sigma > 0$ is the standard deviation. *P* denotes the probability that *X* is less than or equals *x*. (see Papula 2001, 368)

2.2.4 The De Moivre–Laplace Theorem

As it is difficult to calculate probabilities in a binomial distribution containing greater values, the De Moivre-Laplace Theorem proposes that a normal distribution can be used to approximate such a binomial distribution, should the number of outcomes be too great to be reasonably simulated using a binomial distribution. (see Papula 2001, 386)

The parameters μ and σ are replaced as follows:

$$\mu = n \cdot p \text{ and } \sigma = \sqrt{n \cdot p \cdot q} = \sqrt{n \cdot p\,(1-p)}.$$

This is usually used, when the inequation $n \cdot p\,(1-p) > 9$ applies. (see Papula 2001, 386)

2.3 Information Technology Baselines

2.3.1 Graph Theory

In graph theory, graphs are made up of vertices and edges. Vertices represent objects in a system, edges are connections between these vertices, representing the relations between the objects. Generally, graphs can be separated into two types. There are directed graphs and undirected graphs (see Saake and Sattler 2002, 401). Since the application only uses undirected graphs, directed graphs will not be explained further.

In undirected graphs, an edge connects two vertices in both directions. For example, if an edge connects two vertices called *v1* and *v2*, and the edge exists in an undirected graph, it automatically means that *v1* is related to *v2*, and that *v2* is related to *v1*. Since edges are defined as connections between two vertices, there are no edges from a vertex to itself, meaning there are no loops. (see Saake and Sattler 2002, 401ff)

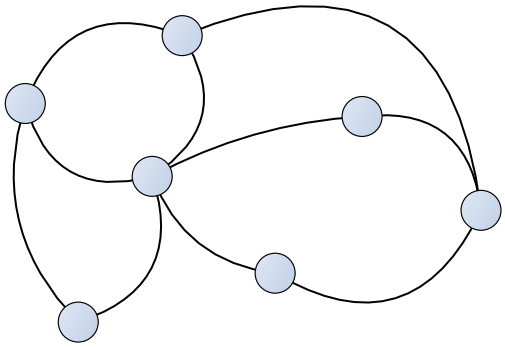

Figure 6: Visualization of a graph.

A graph is defined through an ordered pair, consisting of the set of of vertices and the set of of edges (see Saake and Sattler 2002, 403).

2.3.2 Relational Database Management Systems

A relational database was chosen since the program creates a lot of data in the simulation. In order to have an acceptable performance, a relational database is required. Another reason why is because, should multiple instances of the application work on the same project, they can access data simultaneously.

The program can be run with multiple different types of databases. Those chosen types were MySQL, MariaDB and SQLite. MySQL was chosen because it is one of the most commonly used types. MariaDB is a fork often found in Linux computers. Lastly SQLite, which does not need an explicit database server, was chosen since it can be loaded with simply sample data and can therefore be used as a quick demonstration.

2.3.3 Description of Database Structures using E/R-Models

An entity-relationship-model, or E/R-model, is a data model based on entities and relationships between them. Data models represent a simplified version of reality. They are used to organize data and to depict relationships between elements in the data. The before mentioned entities are an abstraction from an object in the real world. Entities possess attributes. Relationships are the connections between the entities, possibly also having attributes themselves. (Herold, Lurz and Wohlraab 2007, 441)

2.3.4 The Relational Database Model

A relational data model is a method of implementing an E/R-model. It categorizes entities within tables, each table having attributes. The entities in these tables are then assigned specific values for the attributes. This is how the data base is realized in this application. (Herold, Lurz and Wohlraab 2007, 442)

2.3.5 The Model-View-View Model Pattern

While the E/R-model is used to organize only the database, and therefore the storing of data itself, MVVM is a design pattern that is used in developing applications as a whole. It divides the program into three different parts, the view, the view model and the model. (see Britch, Schonning, Dunn and Craig 2017)

Figure 7: The MVVM Pattern

The view represents the user interface. It is the part the user interacts with directly. The code of the view should be kept minimal, as the actual calculations happen in other parts of the program. (see Britch, Schonning, Dunn and Craig 2017)

The view model connects the view and the model. It is the part that interacts with user input, while the view only shows the results visually. It also uses data from the model to better convey information to the user via the view. (see Britch, Schonning, Dunn and Craig 2017)

Lastly, the model is the strictly non-visual part of the program. It is used to store necessary data. (see Britch, Schonning, Dunn and Craig 2017)

The way these parts interact is linearly. The view is aware of the view model's activities, the view model is aware of the model's activities. (see Kouraklis 2016, 8)

In general, the MVVM pattern is used to simplify the process of developing an application. Splitting the program into multiple parts makes it easier to notice flaws within one specific part and rectifying them. (see Kouraklis 2016, 1)

3 Creating the Simulation Software

3.1 Prerequisites and Licensing

3.1.1 Used Development Software

Embarcadero Delphi Community Edition

Embarcadero Delphi[4] is an Integrated Development Environment (IDE) version 10.4.2 to design software applications. It includes the Delphi language[5] and the Visual Component Library[6] utilized to develop the application.

Embarcadero Delphi was used under the Embarcadero Community Edition License[7].

MySQL Workbench Community Edition

The MySQL Workbench is part of the MySQL Community Edition[8]. The MySQL Workbench was used to create, design, and manage the MySQL databases, as well as the according documentation. The version last used was 8.0.27.

MySQL was running under the GNU General Public License (GPLv2)[9].

HeidiSQL

Similar to the MySQL Workbench, HeidiSQL[10] version 11.3 helped create, design, and manage the SQLite and MariaDB databases.

HeidiSQL is OpenSource and was released under GPL[11].

yEd Graph Editor

yEd is a graph editor[12]. With it, the graphics in this work were created.

The editor runs under the yEd Software License Agreement Version 1.2, which is Freeware. The license text is included in the distributed software.

---

[4] See Delphi Community Edition. In https://www.embarcadero.com/de/products/delphi/starter [last seen 30 October 2021]

[5] Delphi Language Reference. In https://docwiki.embarcadero.com/RADStudio/Sydney/en/Delphi_Language_Reference [last seen 30 October 2021]

[6] VCL. In https://docwiki.embarcadero.com/RADStudio/Sydney/en/VCL [last seen 30 October 2021]

[7] Embarcadero Softwarelizenz- und Supportvertrag. In https://www.embarcadero.com/de/products/rad-studio/rad-studio-eula [last seen 30 October 2021]

[8] MySQL Community Edition. In https://www.mysql.com/de/products/community/ [last seen 30 October 2021]

[9] GNU General Public License, Version 2. In https://www.gnu.org/licenses/old-licenses/gpl-2.0.html [last seen 30 October 2021]

[10] HeidiSQL. In https://www.heidisql.com [last seen 30 October 2021]

[11] GNU General Public License. In https://raw.githubusercontent.com/HeidiSQL/HeidiSQL/master/out/gpl.txt [last seen October 2021]

[12] yEd graph editor. In https://www.yworks.com/products/yed [last seen 30 October 2021]

3.1.2 Used Third Party Libraries

Icons8

Icons8, version 1.0[13], is a Freeware[14] library containing images and icons used throughout the program.

ZeosLib 7.2.12

ZeosLib[15] is an Open Source software[16] library containing the database components used to access the supported Database Management Systems. In the application version 7.2.12 of this library utilized.

TeeChart Std. VCL / FMX Components

TeeChart[17] is a software library containing charting components. It was used to create the graphical display of the simulation's results. Steema grants a license for royalty free use in compiled applications[18].

AMRandom

AMRandom[19] is a Freeware translation of Alan Miller's Random Module for FORTRAN-90 into Delphi language. It contains several functions to calculate values of probability distributions.

3.1.3 Database Connection Libraries

MySQL

The library libmysql.ddl[8] connects the application to the MySQL Community Edition Database Server. It is deployed with the MySQL Server, the currently used version being 8.0.23. It has a GNU General Public License (GPLv2)[9].

MariaDB

The library libmariadb.dll[20] connects the application to the MariaDB Database Server. This is running under version 3.1.12. It has a Creative Commons Attribution/ShareAlike 3.0 Unported license[21].

---

[13] Icons8. In https://icons8.com [last seen 30 October 2021]

[14] Icons8 license. In https://icons8.com/license [last seen 30 October 2021]

[15] ZeosLib. In https://sourceforge.net/projects/zeoslib/ [last seen 30 October 2021]

[16] GNU Lesser General Public License, Version 2.1. In https://www.gnu.org/licenses/old-licenses/lgpl-2.1.de.html [last seen 30 October 2021]

[17] Steema. In https://www.steema.com/product/vcl [last seen 30 October 2021]

[18] TeeChart Pro VCL / FMX Components. In https://www.steema.com/?/linkIn/VCL_license [last seen 30 October 2021]

[19] AMRandom. In https://www.esbconsult.com/download.htm [last seen 30 October 2021]

[20] MariaDB. In https://mariadb.org [last seen 30 October 2021]

[21] GNU General Public License, Version 3. In https://www.gnu.org/licenses/gpl-3.0.de.html [last seen 30 October 2021]

SQLite

Lastly, sqlite3.dll[22] is a library implementing the functionality of SQLite, running under version 3.34.1. It is public Domain[23].

3.1.4 Licensing of the Developed Software

The source code created in this project and the created application are provided under the MIT license (see The MIT license). The licensee is granted the use, modification and distribution of the software without restriction. Other licenses apply to specific parts of the software, including TeeChart, ZeosLib, Icons8 and AMRandom.

3.2 General Structure of the Application

In the application being programed in this work, there is no singular view, but instead multiple views interacting not only with the view model, but also among themselves.

The *MainForm* is used as the general outline of the visual part, loading in different views based on the buttons above.

There are login-related views, those being the *LoginView* and the *ConnectionAssistantView*. These are the parts shown to the user either when logging in, or when using the assistant to create a new database.

Then there are the views of the project itself, being the *ProjectsView*, the *TasksView* and the *ResultsView*. The *ProjectsView* appears when clicking the Projects button, serving as the setup for a project, simulation setups and simulations. In this view the user enters all information necessary to plan the entire simulation. During the planning of the simulation, the application creates the tasks necessary to simulate. These tasks can then be seen in the *TasksView* and started individually. Lastly, the *ResultsView* is used to shows the results of the simulation.

Like with the views, there is not one view model, but many instead, being the *Simulation*, the *Simulation.PersonState*, the *Simulation.RunDay* and the *SimulationSetup*. The Simulation view model is the central part of the view models, simulating the course of the epidemic while using the other view models to do so. The other view models are, in turn, concerned with a specific part of the simulation. The *Simulation.PersonState* view model contains information and necessary functions about the status of people. It is used by the Simulation when deciding about changes in the state of a person, such as them getting infected. The *Simulation.RunDay* view model is used when calculating properties of a RunDay. Another task of both previously mentioned view models is connecting and saving data to the database, each view model representing a different part of it.

Lastly, the *SimulationSetup* view model is used to initialize the Simulation itself.

[22] SQLite. In https://www.sqlite.org/index.html [last seen 30 October 2021]

[23] SQLite is Public Domain. In https://www.sqlite.org/copyright.html [last seen 30 October 2021]

3.3 General Course of a Project

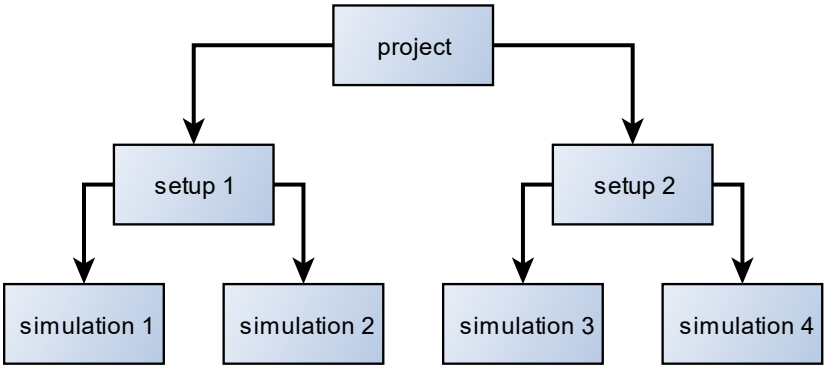

Figure 8: General course of a project

When creating a project, the user can choose to implement not only one, but multiple setups and simulations. A setup represents a population, randomly generated based on the values entered by the user. A simulation is the actual execution of the epidemic in that population. It can prove effective to use multiple setups and multiple simulations. During a simulation, there may be some unlikely, however still possible, events that could lead to a false result. Using multiple simulations makes it possible to use averages, negating the effect of such unfortunate results. Multiple population can be used for the same reason, so that it becomes less easy to get a false result.

3.4 The Model Layer

3.4.1 A Graph Model Describing Epidemics

This work uses a graph model to describe the impact of a pandemic on the general population. People in a population are modeled as vertices, the contacts between those people as edges. At the start of the simulation, a population is created. The contacts between the people are then randomly generated. Through this, a graph is constructed. Through inserted attributes, such as susceptibility or contact probability, the edges are weighted accordingly. Using these weights, it is determined based on probability thresholds which people exceeding the probability threshold have been infected. Because of that, the graph changes, updating the population and course of epidemic. This is repeated for each day in the simulation. Finally, it is determined how the different compartments changed over the course of the simulation.

The attributes used in the simulation can be divided into three classes.

Attributes of the Graph

These attributes describe the graph itself and are consistent between every population. This allows that the different outcomes can be reasonably compared. This class of attributes consists of *Population Size*, *Infectious on Start* and *Simulation Days*. *Population Size* determines the exact amount of people in the population. *Infectious on Start* determines the number of people already infected on day 0. Lastly, *Simulation Days* describes the number of days and therefore defines numbers in repeating the steps in the simulation.

Attributes of Contacts

The attributes of contacts determine the weight of the edges, including the probability of meeting susceptible individuals and frequency to conduct the infection (virulence). To better simulate differ-

ent factors that influence these two attributes, there are multiple attributes. Those include *Contacts per Person*, *Contact Probability*, *Susceptibility*, and *Infectiousness* (virulence). Through *Susceptibility* and *Infectiousness*, the probability of an infection is determined, while *Contacts per Person* and *Contact Probability* influence the meetings. All the attributes described in this paragraph are randomly determined and vary between different populations.

Attributes of a Person

Finally, these attributes describe the course of the disease for one person, as the person moves through the different compartments. These consist of *Incubation Period*, *Infectious Delay* (lag time between infection and ability to infect others), *Infectious Period*, *Disease Period*, and *Mortality*.

There are two different processes described with the attributes. One is about the way the person moves through the compartments, the other is about the infectiousness of the person. While *Incubation Period* and *Disease Period* describe the former, *Infectious Delay* and *Infectious Period* make up the latter. Lastly, there is *Mortality*. This attribute randomly determines whether a person dies or survives.

All of these attributes are randomly determined and therefore vary between different populations.

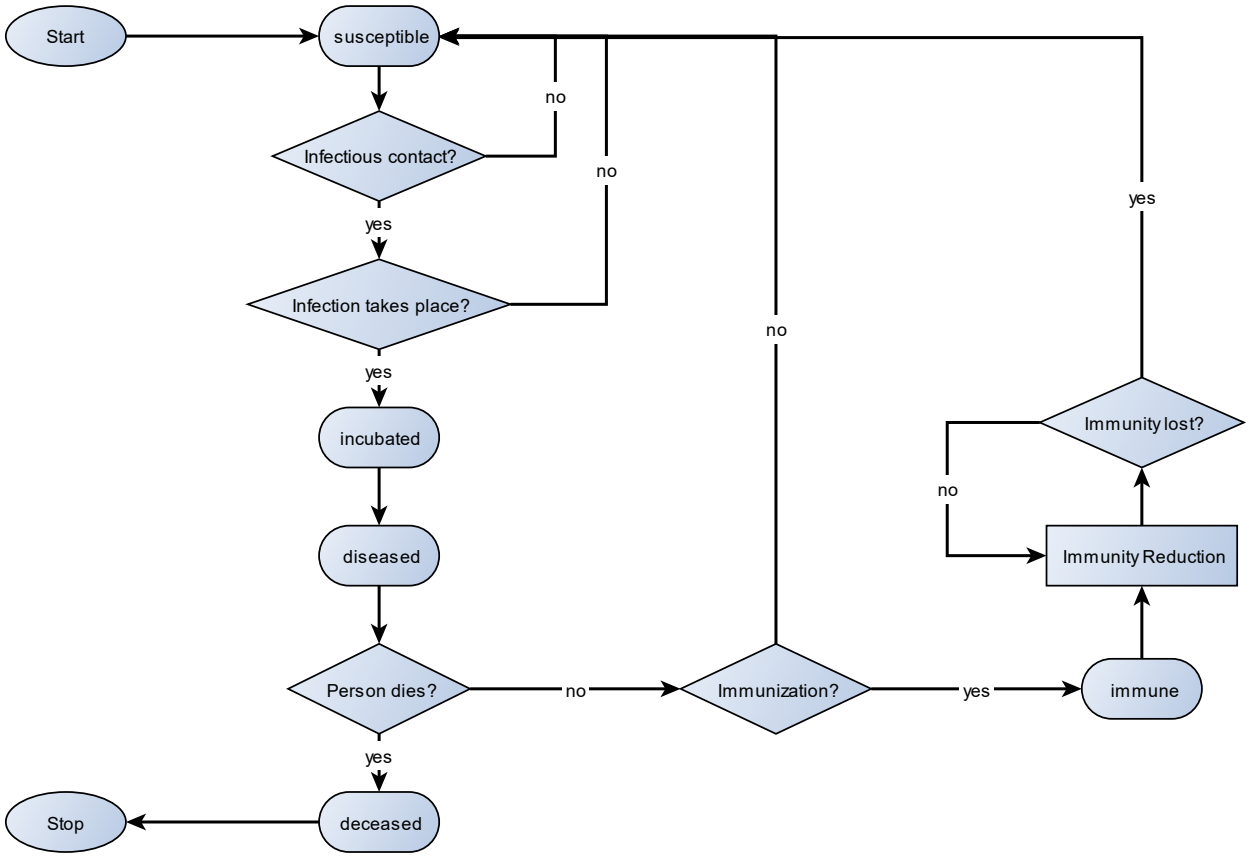

Figure 9: Course of the disease within one person

3.4.2 The Entity-Relationship-Model

The following figure shows the ER-Model of data structure used in the application.

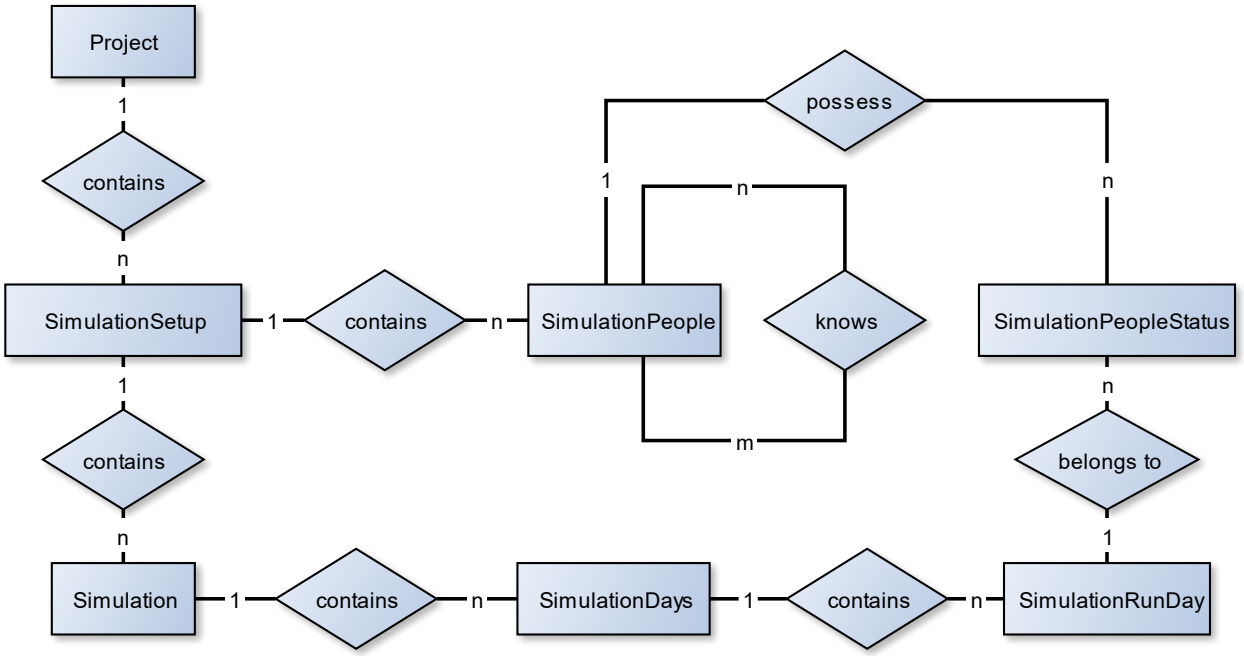

Figure 10: The Entity-Relationship-Model

The entities in the model are connected via relationships. The first is the *Project* entity. It contains the *SimulationSetups*. These, in turn, contain the simulations and the *SimulationPeople*. The *SimulationPeople* are related to themselves by knowing each other. This represents the contacts between them. They also possess a status, represented by the entity *SimulationPeopleStatus*. A Simulation contains *SImulationDays*, which then contain *SimulationRunDays*. Finally, multiple statuses belong to one *SimulationRunDay*.

3.4.3 The Relational Database Model

The following graph visualizes the setup of the relational database.

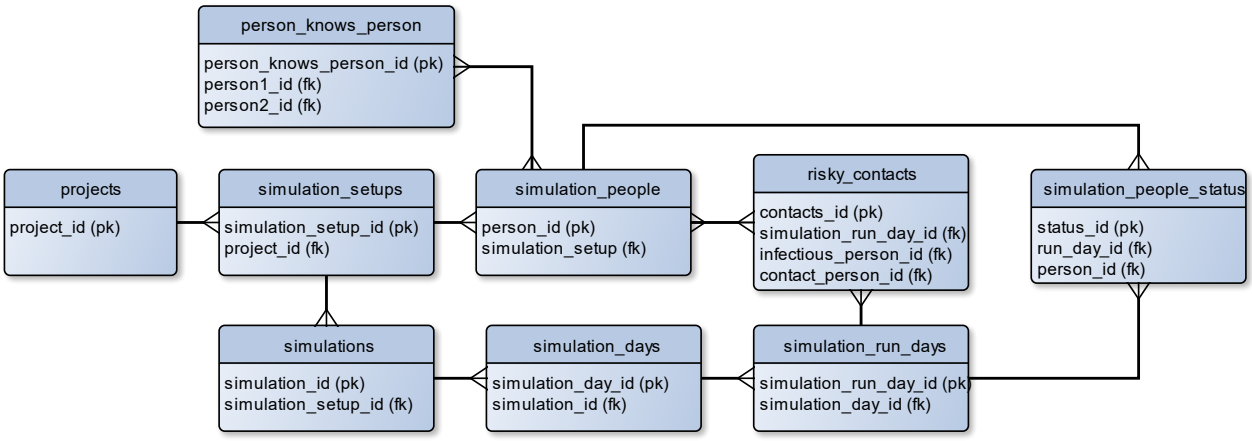

Figure 11: The relational database model

The database model consists of multiple tables, each representing multiple entities in the database.

First there is the projects table. It contains the information the user puts in. It does not have specific values for each attribute, such as Infectiousness, but it contains the mean and standard deviation values necessary to calculate them.

Table 1: The relation projects

Attribute name	Data type	Description
project_id	char(36)	The ID of the project. It is randomly generated and not put in by the user. It is strictly generated by the program itself and can therefore not be entered by the user.
name	varchar(100)	The name of the project. It cannot be longer than 100 characters.
description	text	A description of the project.
population_number	int	The number of people in the population (N).
default_infectious_on_start	int	The number of people (n_0) infected on day 0.
default_simulation_duration	int	The number of days the simulation is to run.
mean_contacts	double	The mean value of contacts per person in the population per day.
deviation_contacts	double	The standard deviation for number of contacts.
mean_contact_probability	double	The mean value for the probability of a meeting between two people with an established contact.
deviation_contact_probability	double	The standard deviation for the probability of a meeting.
mean_susceptibility	double	The mean value for how susceptible people in the simulation are to the disease.
deviation_susceptibility	double	The standard deviation for how susceptible people are.
mean_infectiousness	double	The mean value of infectiousness of a disease.
deviation_infectiousness	double	The standard deviation of infectiousness.
mean_initial_immunity	double	The mean value for initial immunity in a person. As the immunity decreases over time, this value is used to calculate the initial immunity.
deviation_initial_immunity	double	The standard deviation for the initial immunity.

mean_mortality	double	The mean value for mortality in infected people.
deviation_mortality	double	The standard deviation for mortality in infected people.
mean_infectious_delay_period	double	The mean value for the delay before a person becomes infectious.
deviation_infectious_delay_period	double	The standard deviation for the infectious delay period.
mean_infectious_period	double	The mean value for the time a person is infectious.
deviation_infectious_period	double	The standard deviation for the time a person is infectious.
mean_incubation_period	double	The mean value for the time a person is incubated but does not show symptoms yet.
deviation_incubation_period	double	The standard deviation for the time a person is incubated.
mean_disease_period	double	The mean value for the time a person is sick and shows symptoms.
deviation_disease_period	double	The standard deviation for the time a person is diseased.
mean_halving_immunity_period	double	The mean value for the time until a person's immunity has halved.
deviation_halving_immunity_period	double	The standard deviation for the time until a person's immunity has halved.
mean_immunity_period	double	The mean value for the time until a person has lost immunity completely.
deviation_immunity_period	double	The standard deviation for the time until a person has lost immunity completely.

The second table is the *simulation_setups* table. It contains information necessary for the *Simulation Setups* page in the program. A simulation setup is essentially a specific population. As the course of the epidemic is randomly generated, it might be useful to use multiple different populations instead of just one. The *simulation_setups* table is connected to the projects table via a foreign key, that being the *project_id*.

Table 2: The table simulation_setups

Field name	**Data type**	**Description**
simulation_setup_id	char(36)	The ID for the simulation setup.
project_id	char(36)	The foreign key that connects the projects table to the simulation_setups table.
name	varchar(100)	The name of the simulation setup.
default_simulation_duration	int	The number of days the simulation

		lasts.
infectious_onn_start	int	The number of people already infected on day 0.
setup_state	varchar(100)	The current state of the setup.

The next table is the simulations table. This table contains the important information for the *Simulation* page in the program. The simulation is the course of the epidemic. It is important if the user wants to simulate an epidemic within the same population multiple times. It is connected to the *simulation_setups* table with the foreign key *simulation_setup_id*.

Table 3: The table simulations

Field name	**Data type**	**Description**
simulation_id	char(36)	The ID of the specific simulation.
simulation_setup_id	char(36)	The foreign key connecting the simulations table to the simulation_setups table.
name	varchar(100)	The name of the simulation.
description	text	A description of the simulation.
simulation_duration	int	The number of days the simulation lasts.
number_of_runs	int	Describes how often the simulation is to be repeated.

The next table, *simulation_days*, contains important information about each specific day, keeping track of how the disease spreads and comparing it to other runs. This becomes important when evaluating aggregated date for multiple runs and creating a general course of the epidemic based on summarizing all the previous runs. The foreign key *simulations_id* connects the *simulation_days* table to the simulations table.

Table 4: The table simulation_days

Field Name	**Data type**	**Description**
simulation_day_id	char(36)	The ID for the simulation_day.
simulation_id	char(36)	The foreign key connecting the simulation_days table to the simulation table.
day_number	int	The number of the current day.
mean_susceptible	double	The mean value of susceptible people on that day.
deviation_susceptible	double	The standard deviation for how many people were susceptible on that specific day.
delta_susceptible	double	The difference of people susceptible compared to the previous day.
mean_incubated	double	The average of people incubated on that day.

deviation_incubated	double	The standard deviation of people incubated on that day.
delta_incubated	double	The difference of people incubated.
mean_diseased	double	The mean value for how many people were diseased on that day.
deviation_diseased	double	The standard deviation for how many people were diseased.
delta_diseased	double	The difference of people diseased.
mean_infected	double	The mean value for people infected.
deviation_infected	double	The standard deviation for people infected.
delta_infected	double	The difference of people infected.
mean_immune	double	The mean value for people who have gained immunity.
deviation_immune	double	The standard deviation for people who have become immune.
delta_immune	double	The difference of people who have become immune
mean_deceased	double	The mean value of people who have died because of the disease.
deviation_deceased	double	The standard deviation of people who have died.
delta_deceased	double	The difference of people who have died.
mean_removed	double	The mean value for people who are in the removed compartment. This contains people who have gained immunity and people who have died.
deviation_removed	double	The standard deviation for people who are removed.
delta_removed	double	The difference of people who have been removed.
mean_infectious	double	The mean value for people who have been infected and can infect other people.
deviation_infectious	double	The standard deviation for infectious people.
delta_infectious	double	The difference of people who are infectious.
mean_reproduction	double	The mean value for the reproduction count.
deviation_reproduction	double	The standard deviation for the reproduction count.

delta_reproduction	double	The difference of the reproduction count.
mean_infection_risk	double	The mean possibility of someone getting infected.
deviation_infection_risk	double	The standard deviation for someone getting infected.
delta_infection_risk	double	The difference in the possibility of someone getting infected.
mean_infected_last_7_days	double	The mean value of people who got infected during the last seven days.
deviation_infected_last_7_days	double	The standard of people who got infected during the last seven days.
delta_infected_last_7_days	double	The difference of people who got infected during the previous seven days.
mean_new_infected	double	The mean value of people who got infected during the day.
deviation_new_infected	double	The standard deviation of people who got infected during the day.
delta_new_infected	double	The difference of people who got infected during the day compared to the previous day.

The next table is the *simulation_run_days* table. It keeps track of the number of people in the different categories. It is therefore connected to the *simulation_days* table via the foreign key *simulation_day_id.*

Table 5: The table simulation_run_days

Field Name	**Data type**	**Description**
simulation_run_day_id	char(36)	The ID for the simulation_run_day.
simulation_day_id	char(36)	The foreign key connecting the simulation_run_days table to the simulation_days table.
day_number	int	The number of the current day.
run_number	int	The number of the current run. Each run represents a specific course in a specific population.
count_susceptible	int	The number of people ($n_{susceptible}$) susceptible.
count_incubated	int	The number of people (n_{inf_lag}) currently incubated.
count_diseased	int	The number of people (n_{ill}) currently diseased.
count_infected	int	The number of people ($n_{infected}$) who have been infected with the disease.

count_immune	int	The number of people (n_{immune}) who have gained immunity.
count_deceased	int	The number of people (n_{deceased}) who have died because of the disease.
count_removed	int	The number of people who have been removed, including people who have died and those who have become immune ($n_{\text{immune}} + n_{\text{deceased}}$).
count_infectious	int	The number of people ($n_{\text{infectious}}$) who can infect others.
delta_susceptible	int	The difference of people who are susceptible to disease.
delta_incubated	int	The difference of people who are incubated.
delta_diseased	int	The difference of people who are diseased.
delta_infected	int	The difference of people who have been infected.
delta_immune	int	The difference of people who have gained immunity.
delta_deceased	int	The difference of people who have died because of the disease.
delta_removed	int	The difference of people who have been removed.
delta_infectious	int	The difference of people who are infectious.
reproduction_count_infected	int	The reproduction count of people who are infected.
reproduction_count_infectious	int	The reproduction count of the infectious people
count_new_infected	int	The number of people that are newly infected.
infected_history_1	int	Describes how many people were infected one day before.
infected_history_2	int	Describes how many people were infected two days before.
infected_history_3	int	Describes how many people were infected three days before.
infected_history_4	int	Describes how many people were infected four days before.
infected_history_5	int	Describes how many people were infected five days before.
infected_history_6	int	Describes how many people were infected six days before.

infected_history_7	int	Describes how many people were infected seven days before.
sum_infected_7_days	int	The total number of people who were infected in the seven days before current day.
count_risky_contacts	int	The number of contacts between an infected and a susceptible/immune party.

The next table is the *simulation_people* table. It contains information about the general course of the disease within one person. It does, however, not include information about the current status of the person, as this is the task of the *simulation_people_status* table. The table *simulation_people* is connected to the *simulation_setups* table via the foreign key *simulation_setup_id*.

Table 6: The table simulation_people

Field Name	Data type	Description
person_id	char(36)	The ID of the person.
simulation_setup_id	char(36)	The foreign key connecting the simulation_people table and the simulation_setups table
person_number	int	In order to represent visually represent a person for the user, an easily understandable number is assigned to each person instead of the complicated ID.
contacts	int	The contacts a person has.
initial_day_of_infectious	int	The day a person initially becomes infectious.
susceptibility	double	How susceptible a person is to the disease.
infectiousness	double	How infectious the disease is for the specific person
infectious_delay_period	int	The time a person is infected, but not yet infectious.
infectious_period	int	The time a person is infectious.
incubation_period	int	The time a person is infected but does not show symptoms yet.
disease_period	int	The number of days a person shows symptoms.
halving_immunity_period	int	The time until the person's immunity has halved.
immunity_period	int	The total time until a person loses immunity.
initial_immunity	double	The initial immunity.

mortality	double	The probability that the person dies of the disease.

The next table is *person_knows_person*. It is used to represent the contact between two people. It is connected to the *simulation_people* table via two foreign keys, *person1_id* and *person2_id*. Each foreign key represents one person in this contact.

Table 7: The table person_knows_person

Field Name	Data type	Description
person_knows_person_id	char(36)	The ID of the contact.
person1_id	char(36)	The foreign key representing one of the two people connected via the contact.
person2_id	char(36)	The foreign key representing the second person.
contact_probability	double	The probability of the two people meeting and possibly infecting another.

Next, there is the table *risky_contacts*. It becomes important should the user choose to keep contact information. It is connected to the *simulation_people* table with the foreign keys *infectious_person_id* and *contact_person_id*. It is also connected to the *simulation_run_days* table via the foreign key *simulation_run_day_id*.

Table 8: The table risky_contatcs

Field Name	Data type	Description
contacts_id	char(36)	The ID of the contacts.
simulation_run_day_id	char(36)	The foreign key connecting this table to the *simulation_run_day* table.
infectious_person_id	char(36)	The foreign key representing the infectious person of the two people who are connected through a contact.
contact_person_id	char(36)	They foreign key representing the other person.
contact_took_place	boolean	Describes whether a contact has taken place or not.
immunity_works	boolean	If the person is immune, it is randomly determined if the immunity actually functions. This field then contains the result of that check.
infection_took_place	boolean	This field contains information about if the person, after getting into contact with an infected person, was infected themselves.

The next table is the *simulation_people_status* table. It contains information about the current status of a person. It is connected to both the *simulation_run_days* table and the *simulation_people* table via the foreign keys *run_day_id* and person_id. Usually the values are only saved for the previously simulated day.

Table 9: The table simulation_people_status

Field Name	Data type	Description
status_id	char(36)	The ID of the current status.
run_day_id	char(36)	The foreign key connecting the table to the *simulation_run_days* table.
person_id	char(36)	The foreign key connecting this table to the *simulation_people* table.
current_immunity	double	The current immunity.
days_left_incubated	int	The number of days a person is still incubated.
days_left_diseased	int	The number of days a person still has left diseased. Should the person not be diseased at all, the value is automatically set to zero.
days_left_infectious_delay	int	The number of days left until a person becomes infectious.
days_left_infectious	int	The number of days a person can infect others still left.
days_left_immune	Int	The number of days still left until a person completely loses immunity.
count_infected	Int	The number of people this person has infected on this specific day.
sum_infected	Int	The number of people this person has infected in total.
incubated	boolean	Describes whether a person is incubated or not.
diseased	boolean	Describes whether a person is diseased or not.
infected	boolean	Describes whether a person is infected or not.
immune	boolean	Describes whether a person is immune or not.
deceased	boolean	Describes whether a person has died or not.
removed	boolean	Describes whether a person is removed or not.
susceptible	boolean	Describes whether a person is susceptible or not.
infectious	boolean	Describes whether a person is infec-

		tious or not.

Ultimately, the tasks table is responsible for the tasks section in the program. A task can for example be the initialization of a run or the simulation of a specific day in a specific run. Runs become especially important when multiple computers are working with the same project. It is not connected to any other table.

Table 10: The table tasks

Field Name	Data type	Description
task_id	char(36)	The ID of the task.
record_id	char(36)	A field that contains either the setup- or simulation-ID.
task_type	varchar(100)	The type of task, such as setup initialization.
task_priority	Int	A specific value given to the task, describing its priority compared to other tasks.
task_status	varchar(100)	The status of the task.
task_owner	varchar(100)	This value describes which computer is currently working on the task.
day_number	Int	The number of the day, should the computer be working on simulating a day.
run_number	Int	As each day belongs to a certain run, this value describes the number of the run the day is contained in.
created_on	date	Describes when the task was created.
first_start_on	datetime	Describes when the task was first started.
last_start_on	datetime	Describes the last time the task was started.
completed_on	datetime	Describes when the task was completed.
calculation_time	time	Describes how long in total it took to calculate the task. If the task was started, then stopped, and later started again and completed, the time between the two starts is not counted.

3.4.4 Classes of the Model

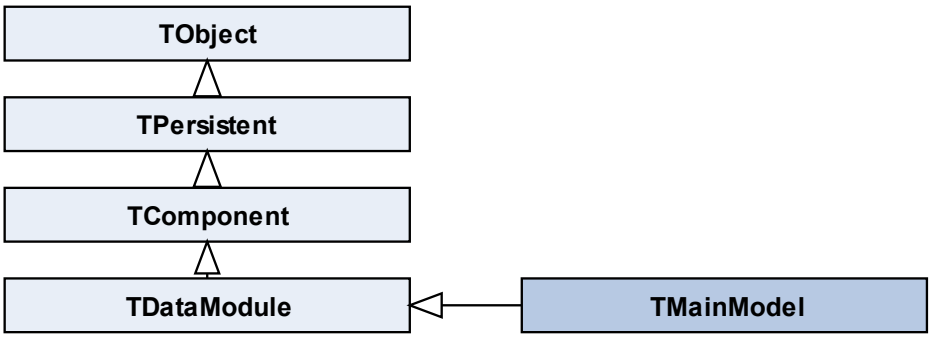

Figure 12: Inheritance overview of the MainModel

Table 11: Classes of the Model

Unit/Class	**Function**
Models.MainModel	The Main Model that saves information into the database.
Models.Nullable	A model that makes it possible to set values as null, meaning there is no assigned value.
Models.SettingsManager	*Models.SettingsManager* handles the settings and informs other parts of the program.

3.5 The View Model Layer

3.5.1 Classes of the View Model

The view model is comprised of multiple units, all inheriting from *TObject*.

Table 12: Classes of the view model

Unit	**function**
ViewModels.Simulation	The main view model. This part is where the simulation itself is executed. It contains functions to initialize the simulation, day one and run day one, as well as to execute a day and a run day. Finally, it also evaluates days. Parts of these functions are included in other units of the view model.
ViewModels.Simulation.Contacts	Inserts contact information into the *risky_contacts* table.
ViewModels.Simulation.Day	Loads information from the *simulation_days* table, while also writing newly simulated data into the table.
ViewModels.Simulation.PersonState	Inserts into and loads from the *simulation_people_status* table. It also contains the function to initialize day zero, which is then used in the main simulation.
ViewModels.Simulation.PersonStatesList	Initializes the states of the people, while also inserting

	them into the database.
ViewModels.Simulation.RunDay	Calculates values specific to the *simulation_run_days* table, such as the delta values.
ViewModels.Simulation.Task	This unit mostly handles the grabbing and releasing of tasks.
ViewModels.SimulationSetup	This unit is not part of the simulation. Instead, it creates the population the disease is spread in.

3.5.2 Task Managing

Tasks are necessary for several reasons. First, since different parts of the simulations must be done in a certain order, they are split into multiple ordered tasks. Another reason is that, as a result of using a relational database, multiple computers can work on the same simulation at the same time. A task contains several steps of the algorithm, which cannot be executed separately. Therefore, this specific part of the simulation is separated as a task from the rest of the complete algorithm. This functionality is made possible by the class *TSimulationTask*. The following figure shows the most relevant attributes and methods of this class.

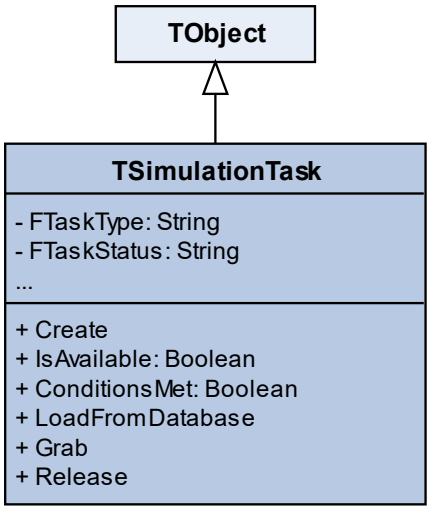

Figure 13: The class TSimulationTask

The private attribute *FTaskType* describes purpose of the task. Possible values of this attribute are further explained in the following table.

Table 13: The different task types

Value	**Description**
'Setup Initialization'	The Setup Initialization creates the population and the contacts between them. It also applies the specific attributes of each person.
'Simulation Initialization'	The simulation initialization creates the necessary records in the database.
'Run Initialization'	The Run Initialization creates day zero and initializes other values, such as the infected histo-

	ry.
'Run Execution'	The simulation of one specific day in one specific run. It decides whether people meet and progresses the course of the disease for every person.
'Day Evaluation'	It compares the results of all days with the same number across the runs to finally create an average course.

The other private attribute shown in the graph is the *FTaskStatus*. The possible values are again explained in the following table.

Table 14: The different task types

Value	**Description**
‘not started’	Tasks that are created as not started are created, but not yet completable.
‘queued’	An available task that is not currently worked on or completed, therefore being able to be executed.
‘running’	A task that is currently being worked on, but not yet completed.
‘finished’	A task that has been completed.

The first method contained in the graph is Create. It is an inherited method used to create instances of the task class.

IsAvailable checks whether a task can be performed. It therefore checks the *FTaskStatus*. Only queued tasks are available. Should the task be queued, this method will return the value ‘true’, if the task is however finished or running, this method will return ‘false’.

As certain tasks depend on other tasks having already been finished, *ConditionsMet* checks this and return a *boolean* value based on whether all the necessary tasks have been finished. The conditions are as follows.

Table 15: Completion requirements

FTaskType	**Conditions**
Setup Initialization	The Setup Initialization does not require any other tasks to be finished.
Simulation Initialization	In order to execute a Simulation Initialization, the Setup Initialization must already be finished.
Run Initialization	To execute the Run Initialization, the Simulation Initialization must have been finished first.
Run Execution	A Run Execution requires the previous day of the same run to be executed first. If the current day number is one, then the Run Initialization must be finished.
Day Evaluation	The Day Evaluation needs a corresponding day

	across all runs and the previous day evaluation to be finished.

LoadFromDatabase loads the values of the task from the database.

Grab marks the task as running, and therefore not available anymore. This becomes useful when working on the same project with multiple machines.

Finally, Release is used upon finishing a task, marking it as finished.

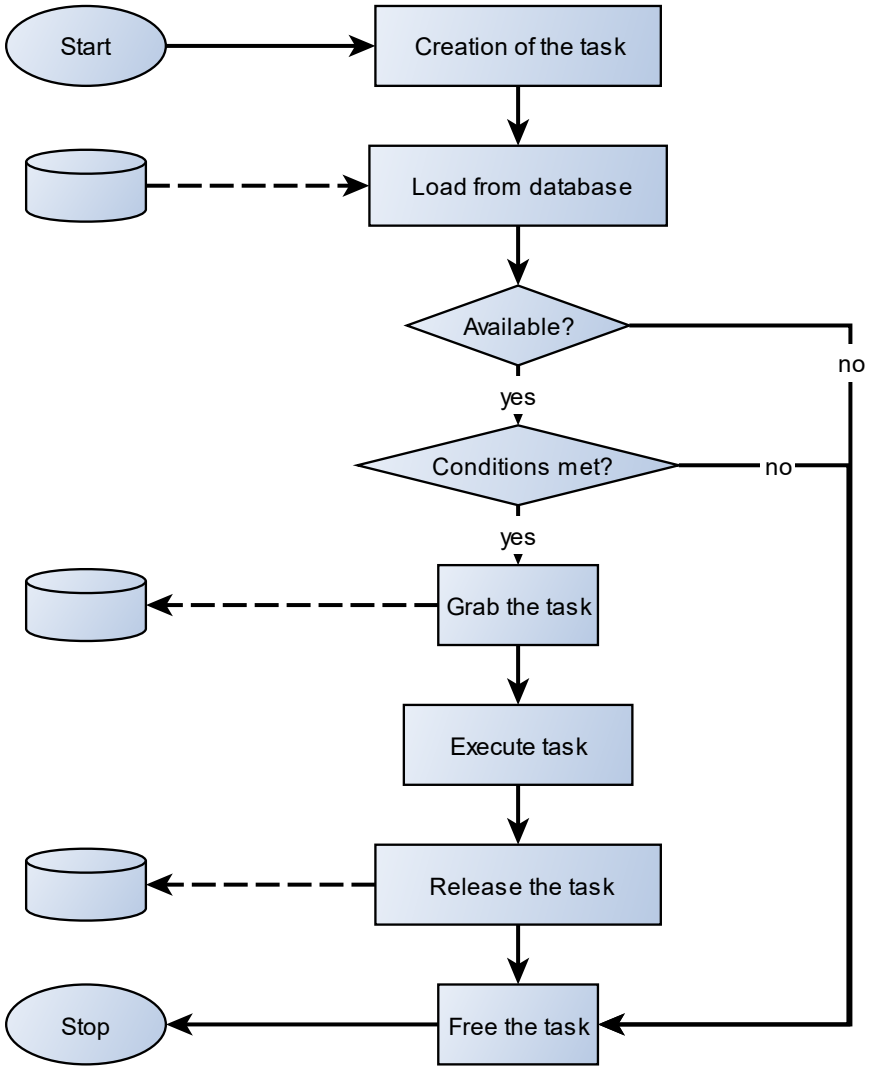

Figure 14: Executing a task

3.5.3 Creating a Population

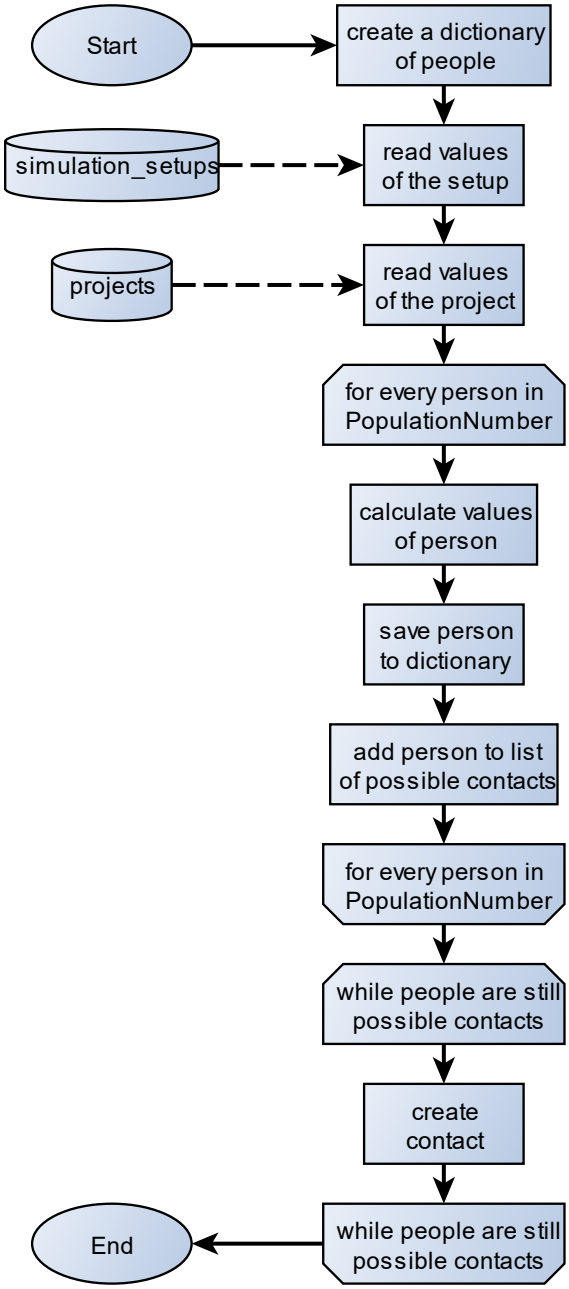

Figure 15: Creating a population

When creating a population, the most important part are the contacts and how they are created. Every person is put into a dictionary, as well as a list of possible contacts. As people are assigned a number of contacts, they are saved just as often into the list of possible contacts. Each person then is assigned contacts randomly from the list of possible contacts. If there are still contacts left after going through every person, new ones are randomly added to other people in the population.

3.5.4 Initializing the Simulation

When initializing the simulation, an entry for each day in the simulation is created within the *simulation_days* table.

3.5.5 Executing the Simulation

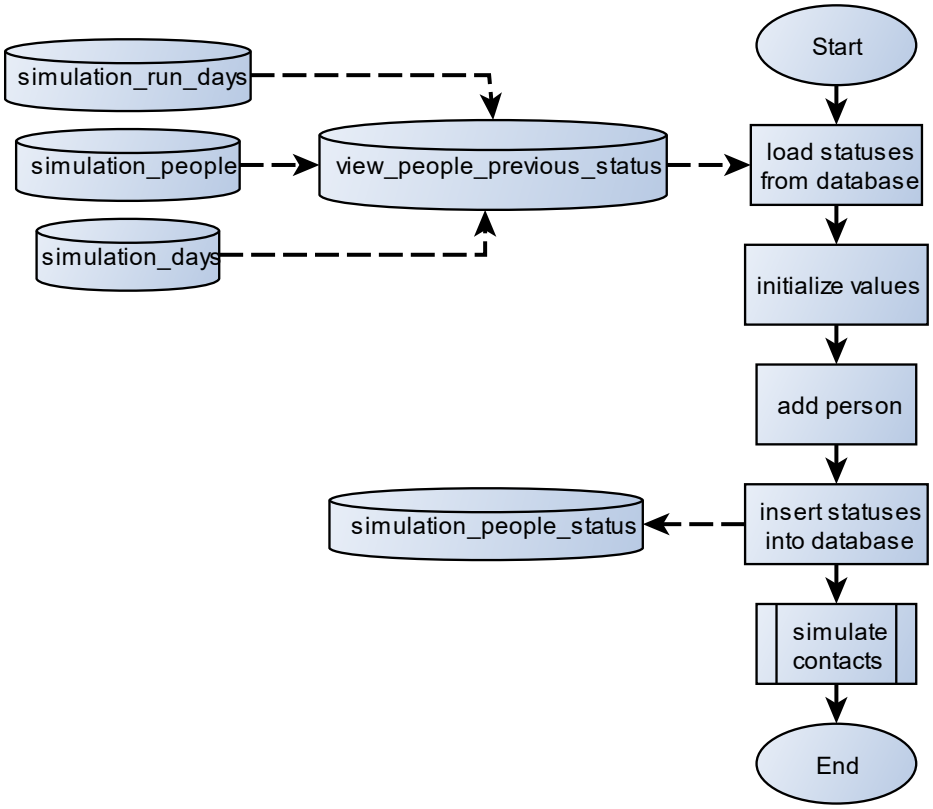

Figure 16: Executing the simulation

During the execution of the simulation, the result of the function is first initialized as false. Should any problems arise during the simulation, this assures that it is stopped at the end. After that, the program establishes a connection to the database. After this connection is established, the program checks if the task selected is available to be simulated. A task would not be available if it is currently worked on. If the task is available, the program checks whether the preconditions are met or not. The preconditions would not be met, for example, if a necessary task, such as the previous day, is not finished. If the conditions are met, the application then initializes the states of the people, inserts them into the corresponding database, simulates contacts and then disconnects from the database.

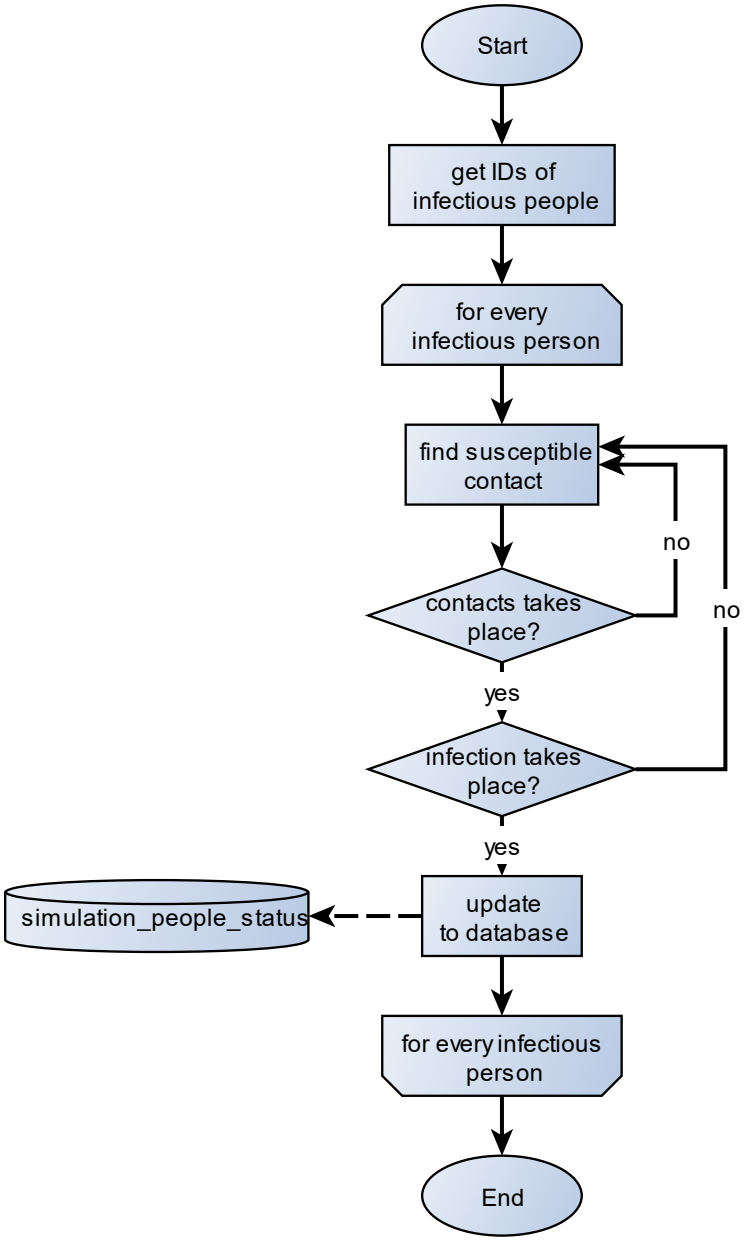

Figure 17: Simulating contacts

In order to make the simulation time as short as possible, infectious people are assigned a specific value. The application then only goes through the people with this value and their contacts without it. For every contact between a susceptible or removed person and an infectious person, it is then randomly determined if they meet and if the disease spreads to the infectable person.

3.5.6 Evaluating the Simulation

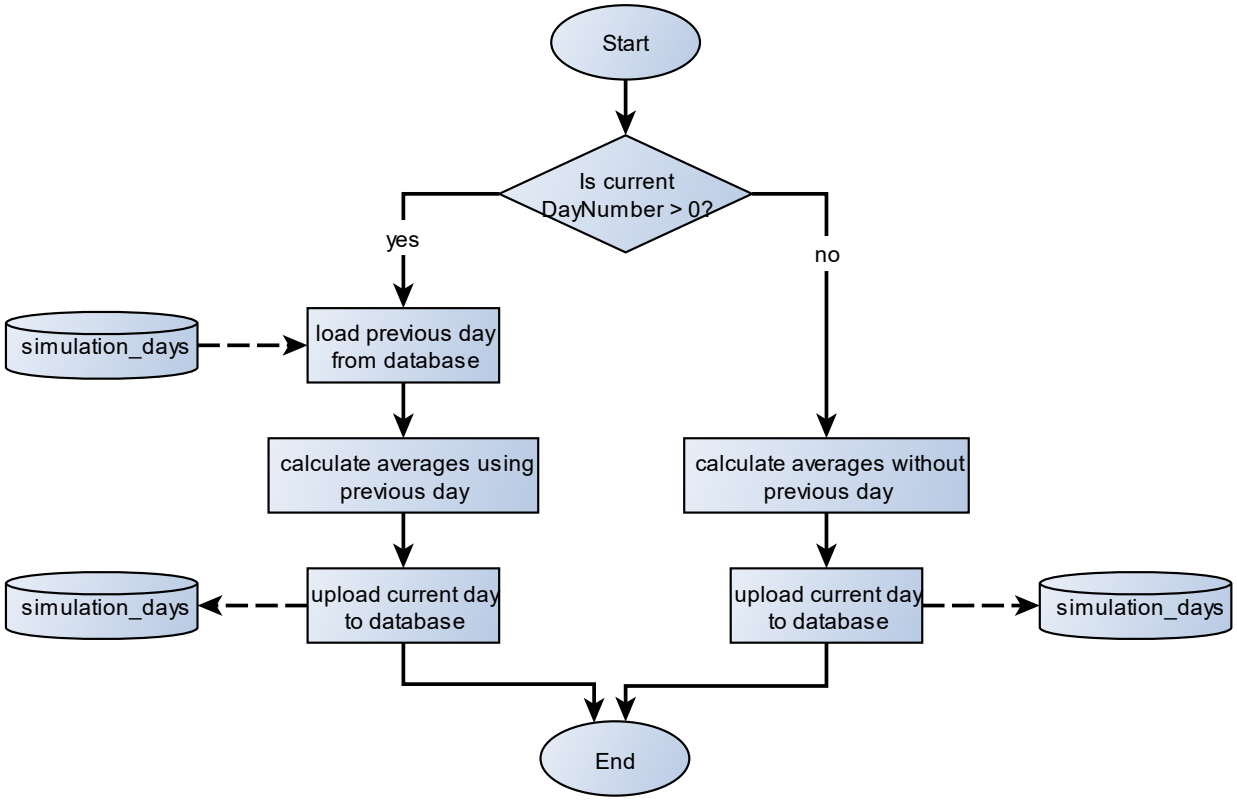

Figure 18: Evaluating the simulation

When evaluating a day, the program fist establishes a connection. There are then two conditions that need to be met. For one, the application checks whether the task is available or not. The next step represents multiple checks. First, it checks if the days associated with the evaluation are finished. The other part is made up of the previous day evaluations. After this, the task is grabbed. There is a split following this. If the day number is greater than zero, the day is evaluated using the previous day, otherwise it evaluates it without using the previous day. The day is then updated and finished in both paths.

3.6 The View Layer

3.6.1 Design of the User Interface

The user interface is made up of multiple compartments. The first one, at the very top, is the title bar. It contains the name of the application, and the currently used database. Below, there is the menu bar. It contains the different sections of the program. The space below is split into two different panels. On the left side, there is the left data panel. Here, the user can choose the object to be shown more details of. The data panel is found on the right side. It is where the information of the chosen item is shown. Further below, there is the action bar. Here, the user can interact with the chosen item. Lastly, there is the status bar, displaying an explanation of the part of the application hovered over.

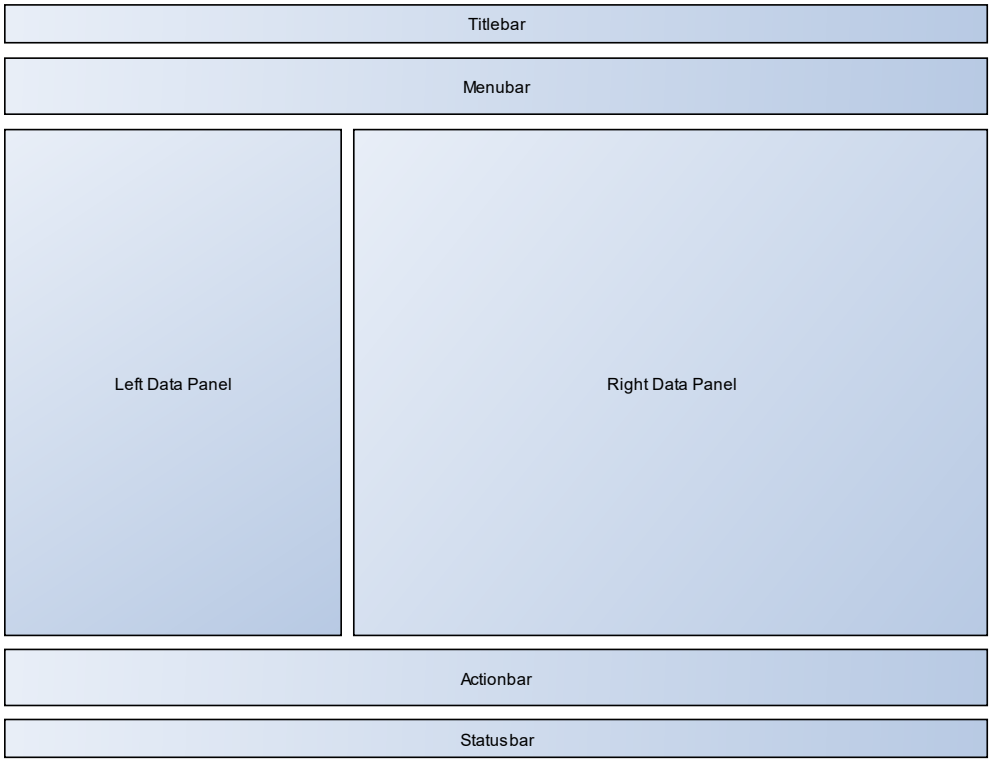

Figure 19: General User Interface Design

3.6.2 Units and Classes of the View

The inheritance overview of the view classes appears as follows:

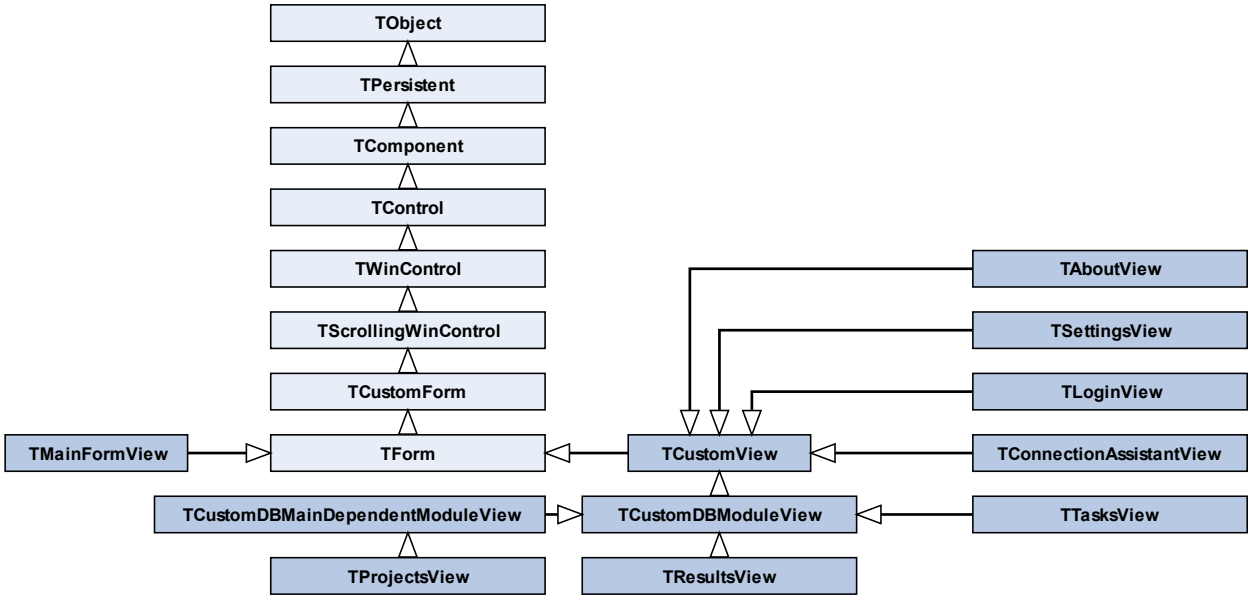

Figure 20: Inheritance Overview of the View Classes

Most of the units and classes are explained in the following chapter. However, some of them will be explained now.

TCustomView adds the option to validate different values. *TCustomDBModuleView* expands the visual aspect of the unit. It adds different buttons to interact with the program, a main source that can be selected, and details of the selected source. Finally, the *TCustomDBMainDependentModuleView* adds another step. When using a *TCustomDBMainDependentModuleView*, a main source and a record within the source can be selected. The details of this record can then be seen.

3.6.3 The Login View

When opening the program, the first view that appears is the Login View. With the Login view, the user can log into the program by entering a username and password. They can also specify the server type, server, port and database. The Connection Assistant can also be accessed via the Login View, Finally, the user opens Projects View with the connect button. Mainly, the functions and procedures are used to check whether the information put in by the user is valid, and to create a respond window either confirming or denying a login.

The Login View contains multiple important functions and procedures. The public functions include the actions the program takes if a button is clicked. Should the user click on the Assistant button, the application uses the function *AssistantBtnClick*.

The private procedures include *PositionInputPanel*, *UpdateInputBox*, *LoadLastConnectionSettings*, *RespondMsgConnectionFailed* and *RespondMsgConnectionSuccessfu*. *PositionInputPanel* is used to position the main panel of the Login View at the center of the screen. *UpdateInputBox* updates the boxes used to input details about the server used. It mainly applies to SQLite, as the server type needs less information in order to be used. *LoadLastConnectionSettings* loads the settings last entered by the user. *RespondMsgConnectionFailed* and *RespondMsgConnectionSuccessful* open the dialog windows confirming or denying a connection.

Finally, the protected function *ValidateData* checks whether the information entered by the user is valid.

3.6.4 The Connection Assistant View

When opening the application for the first time, the Connection Assistant View appears instead of the Login View. This view offers different options. The user can connect to an already existing database, which then opens the Login View. The user can create a new database, and finally, the user can choose to be shown some sample data. As both connecting to an already existing database and showing sample data open up other views, most of the functions and procedures are about the creation of new databases.

3.6.5 The Projects View

The Projects View is where the user sets up multiple simulation. It is the only part of the program the user can put information about the infection and population in. It is split into three main compartments. Details of the selected Project is where the user inputs most of the information that is used to randomly determine the population and specific attributes of the disease. Simulation Setup then shows a specific population. The user can access information about the different, randomly determined people in the population. Lastly, the Simulations compartment contains information about the simulation itself. This is also where the user puts in the number of runs that are to be simulated.

3.6.6 The Tasks View

The task view is an overview of the different tasks the program has to calculate. Each task represents a step in the simulation and calculation of the overall results. It contains information about task properties, such as type or description, and processing status. The processing status is, in essence, the completion status of the task. For example, the time of starting and finishing or task owner are showcased here.

3.6.7 The Results View

The Results View shows the actual results of the simulation. It contains multiple graphs showcasing the results.

Table 16: Description of the charts in the Results View

Table	**Explanation**
Susceptible – Infected – Removed (Proportions)	The Susceptible – Infected – Removed (Proportions) shows how the three compartments in- and decreased over the course of the simulations.
Susceptible – Infected – Removed (Deltas)	The Susceptible – Infected – Removed (Deltas) table instead shows how the different groups Susceptible, Infected, and Removed changed compared to the previous day.
7-Days-Incidence and New Infected	The 7-Days-Incidence and New Infected graph portrays the 7-Day-Incidence and the number of newly infected people on each day.
Infection Risk on Contacts	The Infection Risk on Contact graph shows how likely it is to get infected when meeting an infectious person.
Reproduction Rate	The Reproduction Rate graph showcases the development of how many people the average infectious person infected over the course of their disease.
Population Shares	The Population Shares graph shows how much of the entire population the different groups made up.
Infectious	The Infectious graph portrays the amount of people who can infect others.

3.6.8 The Settings View

The Settings View contains five different settings that can be activated or deactivated.

The first option is to *create queued tasks*. Tasks that are not yet started can either be “not started” or “queued”. This option initializes every task not yet started as “queued”. This option is enabled by default.

Hide finished tasks makes it so that tasks, upon being finished, are not shown to the user anymore. This is because visually, they do not have a lot of purpose to the user. This feature is also enabled by default.

If *Execute all runs* is enabled, a selected day will be executed across all runs. It is also enabled by default.

Keep people states and *Keep contact information*, when enabled, save either the people states or the contact information. These are disabled by default, as not deleting this information greatly increases the size of the database. These options are meant for evaluation purposes only.

3.6.9 The About View

The About View consists of a small window containing information about the program, such as the current version.

3.6.10 Dialog Forms

Dialog forms are simply the windows that open when notifying the user of something important. For example, when entering wrong login data, a dialog appears, denying the login. These dialog forms include confirmation, error, information, waiting and warning dialogs.

3.6.11 Messaging

Due to the design of the program, modules within it cannot communicate with each other and the *MainForm*. However, if it is necessary that they do so, messages are used. The master class subscribes to the messages. The module then sends messages out, and since the *MainForm* class is subscribed to the unit, it receives the message. The following messages are used:

- *Messsages.AboutViewClosed*
- *Messages.ConnectionAssistantRequested*
- *Messages.ConnectionFailed*
- *Messages.ConnectionSuccessful*
- *Messages.DBCreationFailed*
- *Messages.DBCreationSuccessful*
- *Messages.LoginRequested*
- *Messages.LoginSuccessful*
- *Messages.ModuleInfoChanged*
- *Messages.SettingsViewChanged*
- *Messages.SettingsViewSaved*

4 Realization of the Simulation and Analysis of the Results

4.1 Simulation of Known Epidemiological Models

The following diagrams are taken from the sample data included in the application.

First, popular generic epidemiological models were simulated. These included the SI-, SIS-, SIR-. SIRS-, SEIR- and SEIRS-models.

Simulating an SI-model revealed that at some point, the susceptible compartment reaches a value near zero. Reaching a point at which every single person in the simulation is infected is unlikely, especially with high numbers of people in the simulation. Therefore, despite not reaching the exact same state expected from an SI-model, the results are similar.

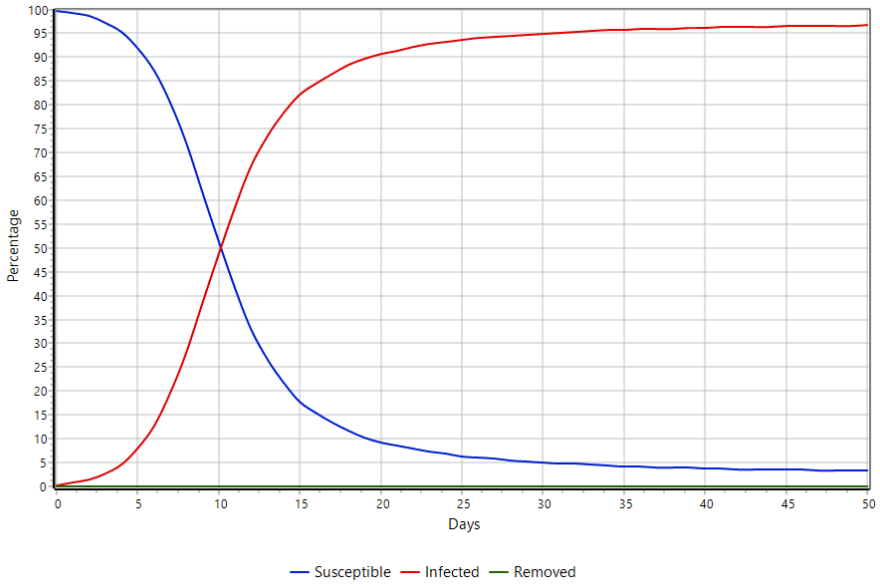

Figure 21: Results of the SI-model 1, setup 1.1, simulation 1.1.1

When simulating the SIS-model, the population reached a point at which both compartments stayed nearly the same. This indicates that the disease became endemic, which was the expected outcome of such a disease. It has, however, also to be noted, that in then simulations done as a part of this work, the population never reached a disease-free equilibrium. The amount of susceptible people was usually higher than the number of infected people as well.

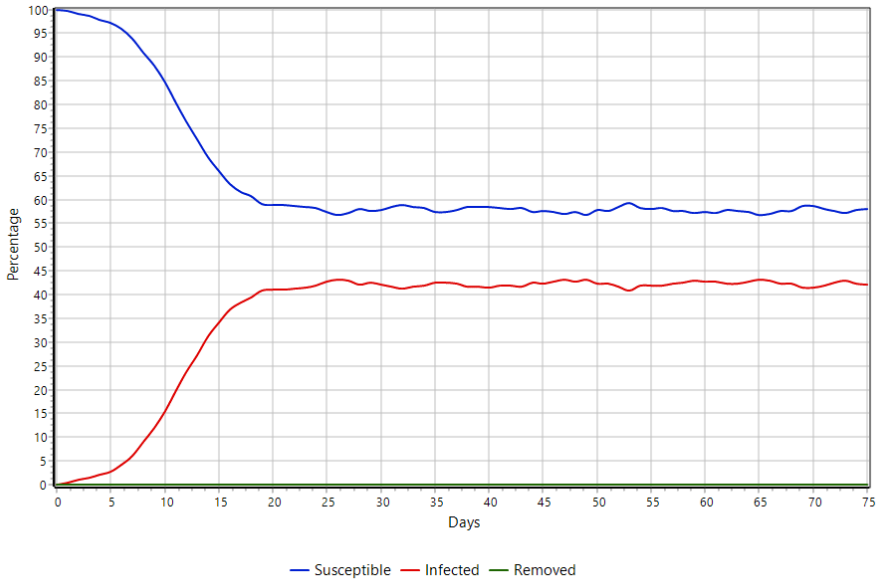

Figure 22: Results of the SIS-model 1, setup 1.1, simulation 1.1.1

In the simulation of the SIR-model, the amount of infected people outgrew the susceptible department, before finally becoming disease-free again. This is the course commonly associated with the SIR-model.

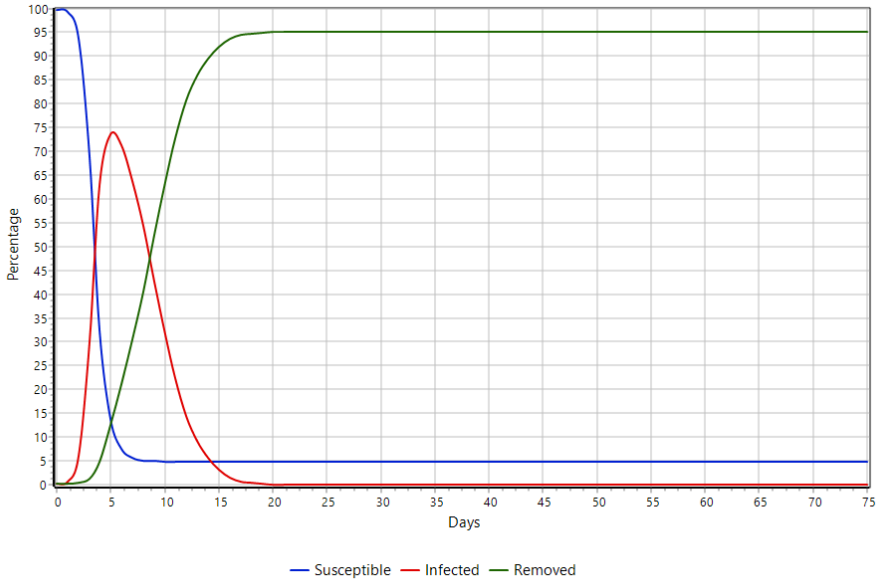

Figure 23: Results of the SIR-model 3, setup 3.1, simulation 3.1.1

The outcome of the SIRS-model also roughly matched the generally assumed one. The amount of infected people stagnated, although it did not reach zero.

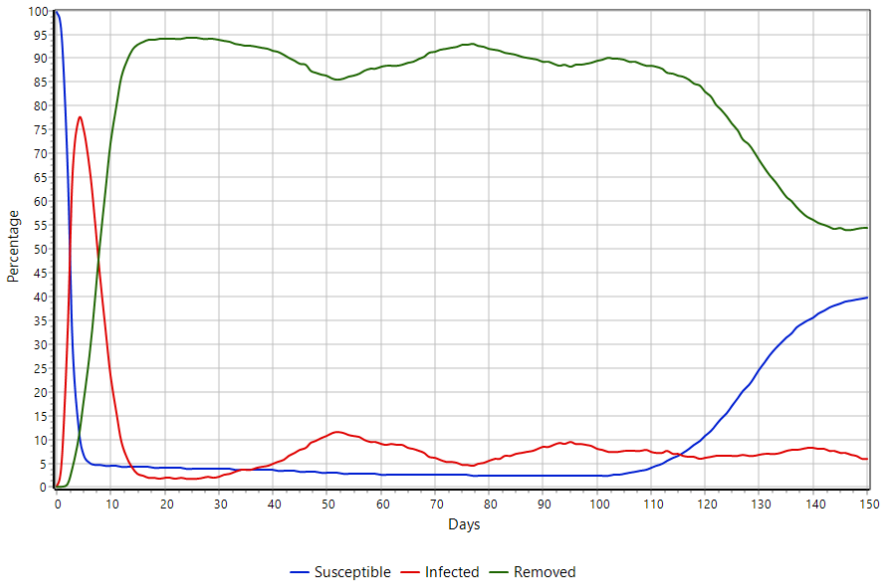

Figure 24: Results of the SIRS-model 1, setup 1.1, simulation 1.1.1

The SEIR- and SEIRS-models also followed the expected course. The SEIR-model ended up nearly the same as observed running the SIR-model, by only having a short delay before the steep decrease of susceptible people. The SEIRS-model also matched the SIRS-model. The proportion of infected people seemed to stagnate. It is noteworthy that smaller increases in infected people occurred, alongside decreasing- and later increasing numbers of removed people.

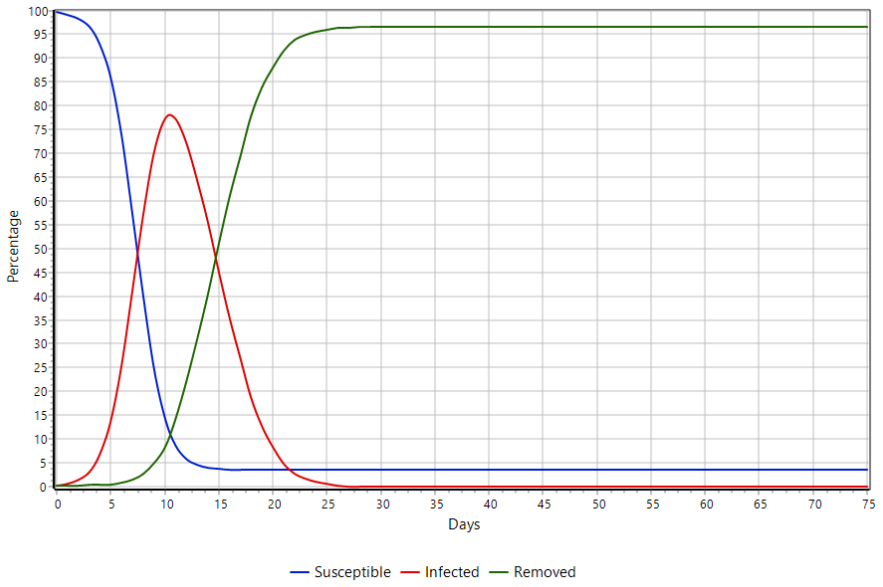

Figure 25: Results of the SEIR-model 1, setup 1.1, simulation 1.1.1

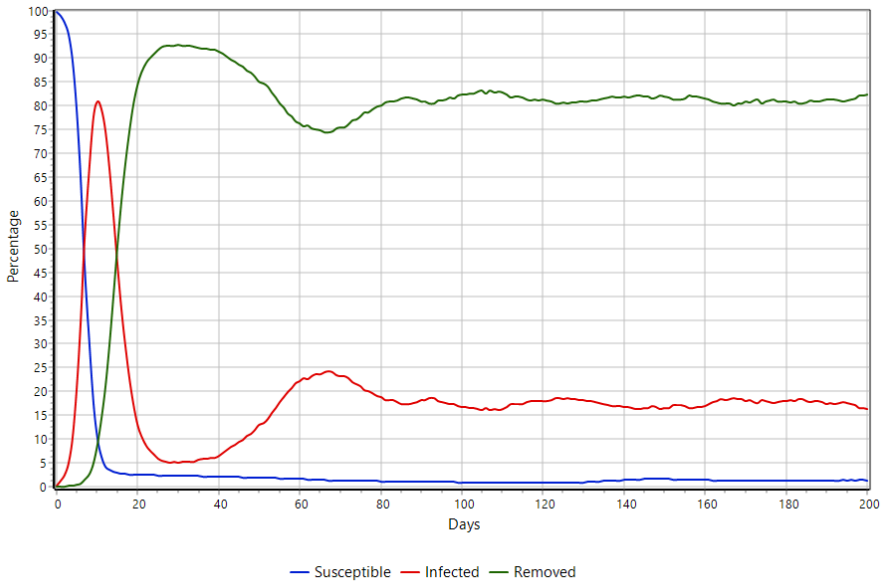

Figure 26: Results of the SEIRS-model 1, setup 1.1, simulation 1.1.1

4.2 Stability of the Results

In order to check the reproducibility of outcomes obtained with the simulation in a way allowing for evidence-based reporting of the simulated model to be reliably valid and the simulation process stable, different populations randomly generated and different simulations must yield similar results. The parameters entered by the user must be the same, but differences in randomly determined values must not be so important that the simulation for equal parameters creates vastly different results. To test this, multiple simulations across multiple populations were completed using the SI-model. As visible from the results, differences were only minor.

Table 17: Stability of the SIS-model

Simulation	**day when infected compartment exceeds susceptible**	**end values**
1.1.1	11	3.32% susceptible, 96.68% infected
1.1.2	12	3.50% susceptible, 96.50% infected
1.1.3	11	3.32% susceptible, 96.68% infected
1.1.4	11	3.39% susceptible, 96.61% infected
1.1.5	11	3.45% susceptible, 96.55% infected
average	11.2	3.40% susceptible, 96.60% infected

4.3 Impact of Differing Factors on Simulated Course of Disease

4.3.1 SIS-Models and Different Infectious Periods

The SIS-model was used twice more, using lower infectious periods. Whereas the normal SIS-simulation had an infectious period of five days, the others had four and three days re-

spectively. The shortening of the infectious period resulted in a prolonged time to achieve the maximum of the also lower number of infected people in the overall population until a defined time. It did not change the fact that the disease became endemic. The two graphs shown below are compared to the figure 23.

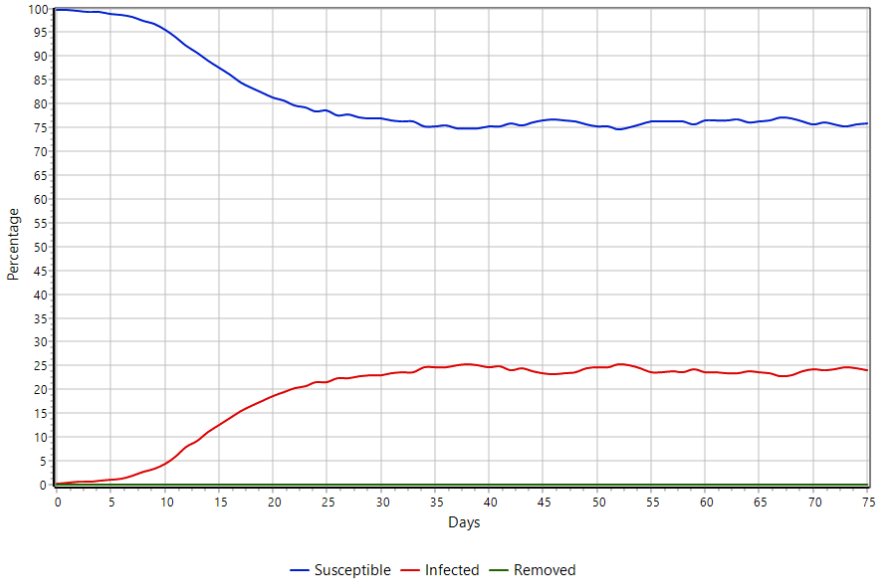

Figure 27: The SIS-model using an infectious period of 4 days

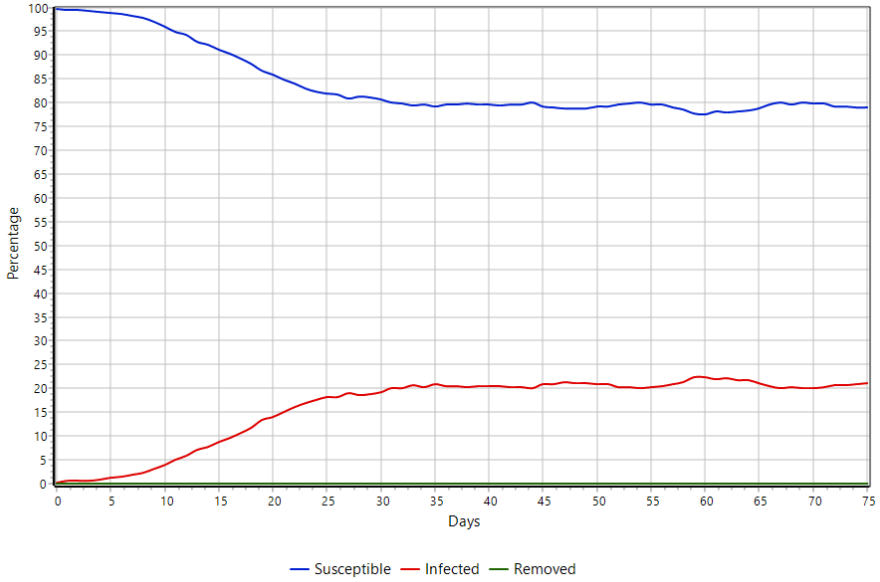

Figure 28: The SIS-model using an infectious period of 3 days

4.3.2 SIR-Models and Differing Infectiousness, as well as Differing Susceptibility

As mentioned before, the results of the initial simulation differed quite a bit from the usual SIR-model. This is true in terms of the course; however, the most relevant outcome of the simulation, achieving a disease-free equilibrium, stayed the same across all SIR-models. The difference between the initial simulation and following ones lied in the infectiousness of the disease and the susceptibility. Again, the course was changed, in that the amount of people who became infected before the disease died out differed, but the outcome stayed the same.

Table 18: Comparing the different results of the SIR-model

changed input	day of peak infected	final susceptible	final removed
20% susceptibility, 20% infectiousness	21	95.37%	4.63%
30% susceptibility, 30% infectiousness	9	32.88%	67.12%
40% susceptibility, 40% infectiousness	5	4.89%	95.11%

4.3.3 SIRS-Models and Different Immunity Periods

Across all SIRS-models, the percentage of removed people rapidly increases. Later, there is another shift. The amount of removed people rapidly decreases again, and the number of susceptible increases. The major difference across the simulations is the point at which the removed people rapidly decrease again.

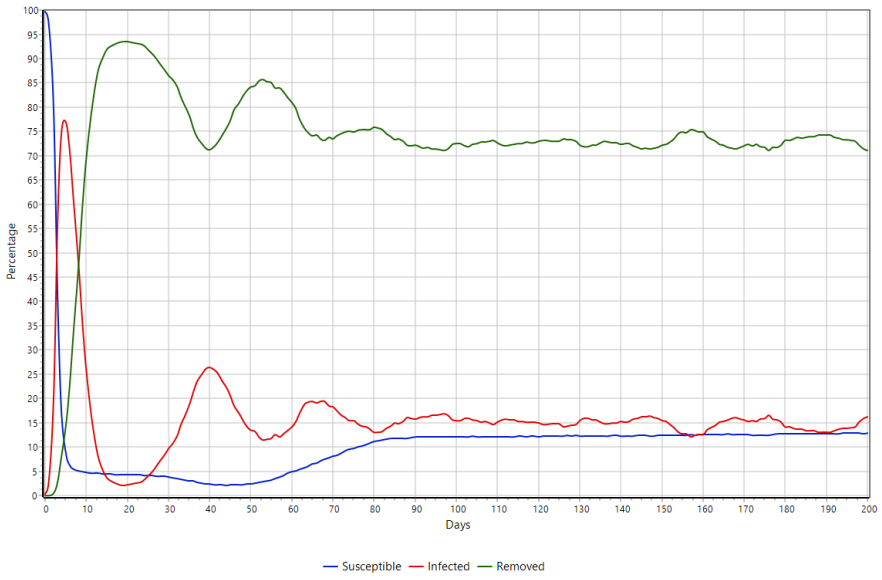

Figure 29: The SIRS-model using a halving period of 30 days and a detectable period of 60 days

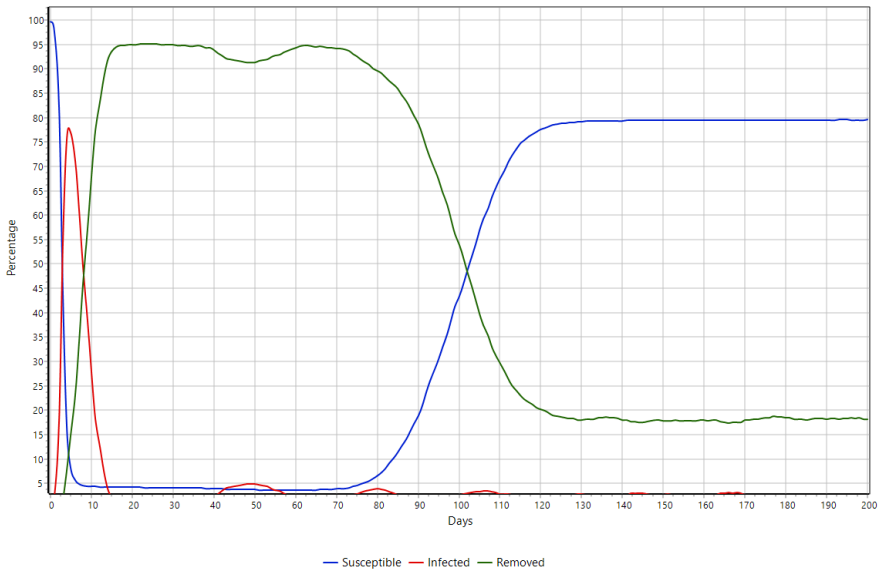

Figure 30: The SIRS-model using a halving period of 45 days and a detectable period of 90 days

4.3.4 SEIR-Models and Different Contact Probabilities

As the result of a generic SIR-model can be compared to that of an SEIR-model, so can the results of these simulations. Changing the contact probability had similar results to changing the infectiousness and susceptibility. The speed at which new infections were made, or even the maximum number of infections, changed.

Table 19: Comparing the differing results of the SEIR-models

changed input	day of peak infected	final susceptible	final removed
25% contact probability	16	20.27%	79.73%
50% contact probability	10	3.48%	96.52%
75% contact probability	8	1.50%	98.50%

4.3.5 SEIRS-Models and Differing Initial Immunity

When changing the initial immunity in the SEIRS-models, the most easily visible change was the percentage at which the different compartments stagnated. Using a high initial immunity, the number of infected people stayed rather low, while it was definitely increased when using low initial immunities.

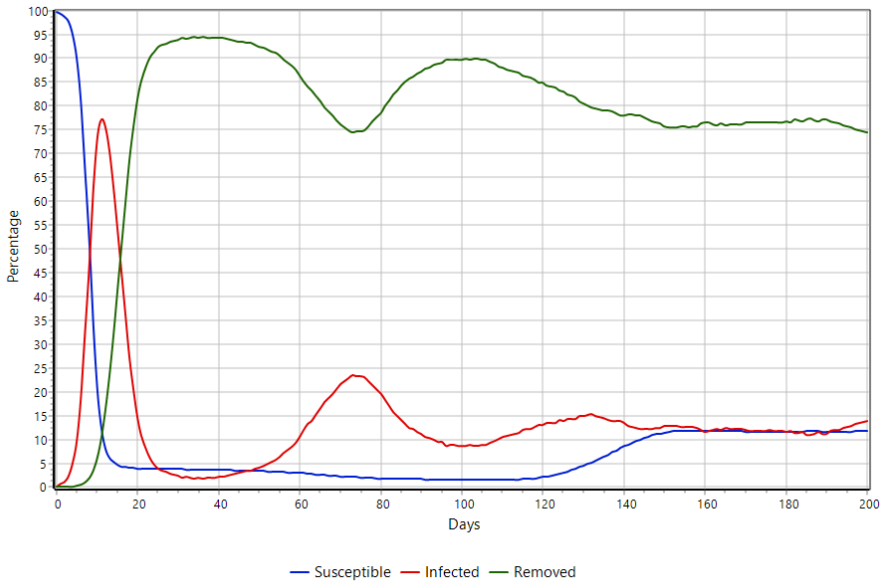

Figure 31: The SEIRS-model using an initial immunity of 0.95

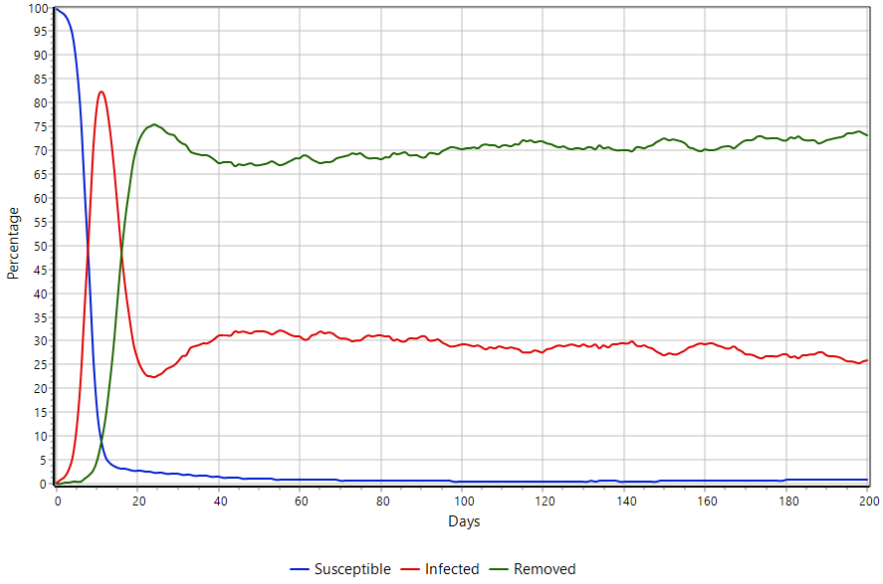

Figure 32: The SEIRS-model using an initial immunity of 0.8

5 Summary and Conclusion

The newly developed software provides an organized graphical user interface for entry of published or arbitrarily chosen data for relevant parameters describing the course of infection over time based on commonly used characteristics applied in epidemiology. These models are the SI, SIS, SIR, SIRS, SEIR, and SEIRS models. All models were used in various simulations and the results of these simulations show that the various models each returned the outcome that is expected from such models. Of course, the specific course depends very much on the particular disease and further characteristics of the population and distribution of relevant immunological properties (genotype and phenotype) as well as age and general health but also the spatial distribution of the population over time. To reflect such very relevant parameters appropriately, measures for susceptibility, infectiousness/virulence, incubation time (delay before becoming infectious after being infected), duration of disease (the number of days being contagious, that means being able to conduct the disease), and time to acquiring and maintaining immunity can be entered with their error margins used for building random distribution patterns. Besides demonstrating utilizability of the software to simulate the course of a disease according to the above-mentioned models, the software could be used successfully to demonstrate the impact of modified values of certain parameters on the distribution of cases between the compartments representing the states susceptible, infected, and removed.

Still, there are some possible developments I would propose:

The first one would be the ability to implement countermeasures. Countermeasures would include actions like widespread vaccination, or reduced contact probabilities, for instance, achieved via lockdowns. These would have an impact on the population in different ways. For example, vaccination would increase the number of removed people, whereas lockdowns would decrease either contacts or contact probability.

Another idea would be to make the software more user-friendly and adaptable so that the parameters for the disease and the population are entered independently of each other, allowing to define a specific population including their characteristics respective to age, sex, and comorbidities could be used to simulate the interaction with another disease. As it is, that would require the user to create an entirely new population.

The third idea would be to implement a Windows service. That would mean that the current program is just used to manage the calculations done by the Windows service. Tasks are created as not started. The client then marks them as queued. The service’s purpose is to automatically calculate all tasks marked as queued.

Overall, facilitating disease modeling by the software developed and used within the here presented may help in quickly deriving useful information about the disease before fully available information about their characteristics and in this regard adequately adapt to potential threats.

Sources

AMRandom. In https://www.esbconsult.com/download.htm [last seen 30 October 2021]

Brauer, Fred; Castillo-Chavez, Carlos; Feng, Zhilan: Mathematical Models in Epidemiology, New York 2019

Britch, David; Schonning, Nick; Dunn, Craig; Osborne, John: The Model-View-ViewModel Pattern. In https://docs.microsoft.com/en-us/xamarin/xamarin-forms/enterprise-application-patterns/mvvm [last seen 09/05/2021]

Celentano, David D.; Szklo, Moyses: Gordis Epidemiology. 6. Edition, Philadelphia 2019

Delphi Community Edition. In https://www.embarcadero.com/de/products/delphi/starter [last seen 30 October 2021]

Delphi Language Reference. In https://docwiki.embarcadero.com/RADStudio/Sydney/en/Delphi_Language_Reference [last seen 30 October 2021]

Differential Equations. In http://www.maths.surrey.ac.uk/explore/vithyaspages/differential.html [last seen 5 September 2021]

Embarcadero Softwarelizenz- und Supportvertrag. In https://www.embarcadero.com/de/products/rad-studio/rad-studio-eula [last seen 30 October 2021]

GNU General Public License. In https://raw.githubusercontent.com/HeidiSQL/HeidiSQL/master/out/gpl.txt [last seen October 2021]

GNU General Public License, Version 2. In https://www.gnu.org/licenses/old-licenses/gpl-2.0.html [last seen 30 October 2021]

GNU Lesser General Public License, Version 2.1. In https://www.gnu.org/licenses/old-licenses/lgpl-2.1.de.html [last seen 30 October 2021]

GNU General Public License, Version 3. In https://www.gnu.org/licenses/gpl-3.0.de.html [last seen 30 October 2021]

HeidiSQL. In https://www.heidisql.com [last seen 30 October 2021]

Herold, Helmut; Lurz, Bruno; Wohlrab, Jürgen: Grundlagen der Informatik, München 2007

Icons8. In https://icons8.com [last seen 30 October 2021]

Icons8 license. In https://icons8.com/license [last seen 30 October 2021]

Kouraklis, John: MVVM in Delphi. London 2016

MariaDB. In https://mariadb.org [last seen 30 October 2021]

MySQL Community Edition. In https://www.mysql.com/de/products/community/ [last seen 30 October 2021]

Norell, Staffan E.: A Short Course in Epidemiology. Philadelphia 1992

Papula, Lothar: Mathematik für Ingenieure und Naturwissenschaftler Band 3, Wiesbaden 2021

Saake, Gunter; Sattler, Kai-Uwe: Algorithmen und Datenstrukturen, Heidelberg 2002

SI and SIS models. In https://docs.idmod.org/projects/emod-generic/en/latest/model-si.html [last seen 5 September 2021]

Steema. In https://www.steema.com/product/vcl [last seen 30 October 2021]

SQLite. In https://www.sqlite.org/index.html [last seen 30 October 2021]

SQLite is Public Domain. In https://www.sqlite.org/copyright.html [last seen 30 October 2021]

TeeChart Pro VCL / FMX Components. In https://www.steema.com/?/linkIn/VCL_license [last seen 30 October 2021]

The MIT license. In https://opensource.org/licenses/MIT [last used 30 October 2021]

VCL. In https://docwiki.embarcadero.com/RADStudio/Sydney/en/VCL [last seen 30 October 2021]

Weiß, Christel: Basiswissen Medizinische Statistik. 7., vollständige und überarbeitete Auflage, Berlin 2019

yEd graph editor. In https://www.yworks.com/products/yed [last seen 30 October 2021]

ZeosLib. In https://sourceforge.net/projects/zeoslib/ [last seen 30 October 2021]