Forge4Flame (F4F): User Guide

Contents

1 Introduction

This document serves as a user guide for ***Forge4Flame (F4F)***. Specifically, Section 2 provides various instructions for downloading the repository, *F4F*, FLAME GPU 2, the Docker images, and Slurm, while Section 3 explains how to run them in different ways. Section 4 introduces the main features of the application, using an example as a guide. Section 5 provides additional details about the generic middle school environment discussed in the main document. Finally, Section 6 briefly explains the integration between *F4F* and FLAME GPU 2.

2 How to download

This section provides instructions for downloading the repository, the *F4F* R package, FLAME GPU 2, and the Docker images and configuring the HPC environment for Slurm.

2.1 Repository

Cloning the repository is optional, especially for users intending to use the *F4F* package, dockers, or Slurm—see the following sections for more details. The GitHub repository is available at `https://github.com/qBioTurin/FORGE4FLAME`. To clone it, use the following Bash command:

```
git clone --recurse-submodules \
          https://github.com/qBioTurin/FORGE4FLAME.git
```

The main directory, `FORGE4FLAME`, contains the R Shiny application, while the `inst/FLAMEGPU-FORGE4FLAME` directory stores the FLAME GPU 2 template used by *F4F*.

2.2 F4F

To install the *F4F* R package it is possible to use the R package *devtools* [32] or *remotes* [14], through these R commands:

```
if(!requireNamespace("devtools", quietly = TRUE))
    install.packages("remotes")

library(devtools)
install_github("https://github.com/qBioTurin/FORGE4FLAME")
```

or:

```
if(!requireNamespace("remotes", quietly = TRUE))
    install.packages("remotes")

library(remotes)
install_github("https://github.com/qBioTurin/FORGE4FLAME")
```

2.2.1 Dependencies

F4F is implemented in R Shiny [11]. It requires several CRAN packages to operate: *shinydashboard* [10], *shinyjs* [4], *jsonlite* [19], *dplyr* [33], *shinythemes* [12], *colourpicker* [2], *glue* [16], *readr* [34], *zip* [13], *sortable* [27], *shinyalert* [3], *shinybusy* [18], *shinyBS* [7], *stringr* [31], *ggplot2* [29], *tidyr* [30], *DT* [35], *shiny* [9], *shinyWidgets* [24], *shinyFiles* [22], and *htmltools* [25]. These packages are automatically installed.

Furthermore, it also requires the EBImage [21] and the EBImageExtra [20] packages. The user must install these packages manually using the following R commands:

```
if(!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("EBImage")


if (!requireNamespace("remotes", quietly = TRUE))
    install.packages("remotes")

library(remotes)
install_github("ornelles/EBImageExtra")
```

2.3 FLAME GPU 2

To install FLAME GPU 2 dependencies, refer to the official documentation here.

2.4 Dockers

Users can download the Docker images for both *F4F* and FLAME GPU 2 to avoid any potential dependency-related issues. In this context, the user must have Docker installed on their computer. For more information, refer to this document: `https://docs.docker.com/engine/installation/`. Additionally, the user needs to ensure they have the necessary permissions to run Docker without using sudo. To create the Docker group and add the user on a Unix system, follow these steps:

- Create the docker group:

  ```
  $ sudo groupadd docker
  ```

- Add user to the docker group:

  ```
  $ sudo usermod -aG docker $USER
  ```

- Log out and log back in so that group membership is re-evaluated.

Additionally, the user must have the NVIDIA driver (installation instructions can be found here) and the NVIDIA container toolkit (installation instructions can be found here) installed. To download the Docker images, run the following Bash commands:

```
docker pull qbioturin/forge4flame
docker pull qbioturin/flamegpu2
```

An alternative method for downloading Docker images is via *F4F*, provided that the user has installed the R package (refer to Section 2 for installation instructions):

```
library(FORGE4FLAME)
FORGE4FLAME::downloadContainers()
```

2.5 Slurm

To run FLAME GPU 2 simulations on an HPC system, the user must install Slurm on it (more information here). Generally, on HPC systems, Docker cannot be used due to privacy concerns. Therefore, the user must install all necessary FLAME GPU 2 dependencies on the user's system before running simulations (see Section 2.3 for more details). However, on HPC4AI [1] (more information here), FLAME GPU 2 can also be executed using Docker, thanks to a tool that addresses privacy concerns.

3 How to run

This section describes how to run *F4F* and FLAME GPU 2, both with and without Docker, using the three approaches outlined in the main document (see Figure 2).

3.1 F4F

3.1.1 Without Docker

To run *F4F* without using Docker, the user can open the `FORGE4FLAME.Rproj` file with RStudio—after cloning the repository—and run it through the *Run App* button or run the following R commands after installing the package (see Section 2.2):

```
library(FORGE4FLAME)
FORGE4FLAME::FORGE4FLAME.run()
```

3.1.2 With Docker

To run it using Docker, execute the following Bash command (if running on a server, ensure that port 3839 is exposed and accessible via `http://<server-hostname>:3839`):

```
docker run -p 3839:3838 qbioturin/forge4flame
```

In this case, there is no need to download the *F4F* repository or install the package.

3.2 FLAME GPU 2

3.2.1 Without Docker

To run FLAME GPU 2 without using Docker, run the following Bash command (inside the `FORGE4FLAME/inst/FLAMEGPU-FORGE4FLAME` directory, after cloning the *F4F* repository):

```
# Executing a single run without using the FLAME GPU 2
# 3D visualization:

./abm.sh -expdir NameOfTheModel

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

# Executing a single run using the FLAME GPU 2
```

```
# 3D visualization:

./abm.sh -expdir NameOfTheModel \
         -v ON

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

# Executing n runs without using the FLAME GPU 2
# 3D visualization:

./abm_ensemble.sh -expdir NameOfTheModel

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

# Visualize the helper:

./abm.sh -h
./abm_ensemble.sh -h
```

In particular, `NameOfTheModel` must be the name of the desired model and must correspond to a directory within `FORGE4FLAME/inst/FLAMEGPU-FORGE4FLAME/resources/f4f` that contains both a JSON file and an RDs file. If the user utilizes the *Link the model to FLAME GPU 2* button (see Section 4.10 for more details), the directory with the provided name will be automatically saved in the correct location. Otherwise, if the user downloads the model using the *Save the model* button, they must manually copy the unzipped directory into `FORGE4FLAME/inst/FLAMEGPU-FORGE4FLAME/resources/f4f`.

3.2.2 With Docker

To run FLAME GPU 2 using Docker, run the following Bash commands:

```
# Executing a single run without using the FLAME GPU 2
# 3D visualization:

docker run --user $UID:$UID \
           --rm \
           --gpus all \
           --runtime nvidia \
           -v AbsolutePathToTheDirectoryWithTheModel:/home/
              docker/flamegpu2/FLAMEGPU-FORGE4FLAME/resources
              /f4f/CustomModel \
           -v $(pwd):/home/docker/flamegpu2/FLAMEGPU-
              FORGE4FLAME/flamegpu2_results \
           qbioturin/flamegpu2 /usr/bin/bash -c "/home/docker
              /flamegpu2/FLAMEGPU-FORGE4FLAME/abm.sh -expdir
              CustomModel"

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

# Executing n runs without using the FLAME GPU 2
# 3D visualization:
```

```
docker run --user $UID:$UID \
           --rm \
           --gpus all \
           --runtime nvidia \
           -v AbsolutePathToTheDirectoryWithTheModel:/home/
              docker/flamegpu2/FLAMEGPU-FORGE4FLAME/resources
              /f4f/CustomModel \
           -v $(pwd):/home/docker/flamegpu2/FLAMEGPU-
              FORGE4FLAME/flamegpu2_results \
           qbioturin/flamegpu2 /usr/bin/bash -c "/home/docker
              /flamegpu2/FLAMEGPU-FORGE4FLAME/abm_ensemble.sh
               -expdir CustomModel"
```

In this case, there is no need to download the *F4F* repository or install the package. `AbsolutePathToTheDirectoryWithTheModel` represents the absolute path to the local directory that contains the model to run (the JSON and the RDs files). These files will be saved in a directory named `CustomModel` inside the Docker (in `FLAMEGPU-FORGE4FLAME/resources/f4f`). Results will be saved in a directory named `results/CustomModel` within the current directory.

3.3 Docker Compose

To use the Docker Compose, the user must download the YAML file here. To start both *F4F* and FLAME GPU 2 containers, navigate to the directory containing the YAML file and run the following Bash command (if running on a server, ensure that port 3839 is exposed and accessible via `http://<server-hostname>:3839`; if running locally, access to `http://localhost:3839`):

```
docker compose up -d --build
```

To run a FLAME GPU 2 simulation using Docker Compose, the user must use the **Run** page of *F4F*—see Section 3 for more details. Results will be saved in a directory named `results/CustomModel` within the current directory.

To stop the containers, run the following Bash command:

```
docker compose down
```

3.4 Slurm

3.5 How to reproduce the results

To reproduce the results presented in the main document—comparing FLAME GPU 2 and NetLogo using the ITP and the alert scenarios—run the following Bash commands (note that this process may take time, especially for NetLogo):

```
git clone --recurse-submodules \
          git@github.com:qBioTurin/FLAMEGPU-FORGE4FLAME.git
git checkout School

# Reproduce results on comparison
# between FLAME GPU 2 and NetLogo:
./reproduce_comparison.sh
```

```
# Reproduce results on alert
# scenarios:
./reproduce_alert.sh
```

4 Forge4Flame application

In this section, we provide a detailed overview of all functionalities available in the *F4F* application, explaining each tab individually.

4.1 Homepage

Figure 1 illustrates the **Homepage** of *F4F*, providing a clear overview of the application. The left-hand menu organizes *F4F*'s main functionalities, each designed to streamline various stages of model creation.

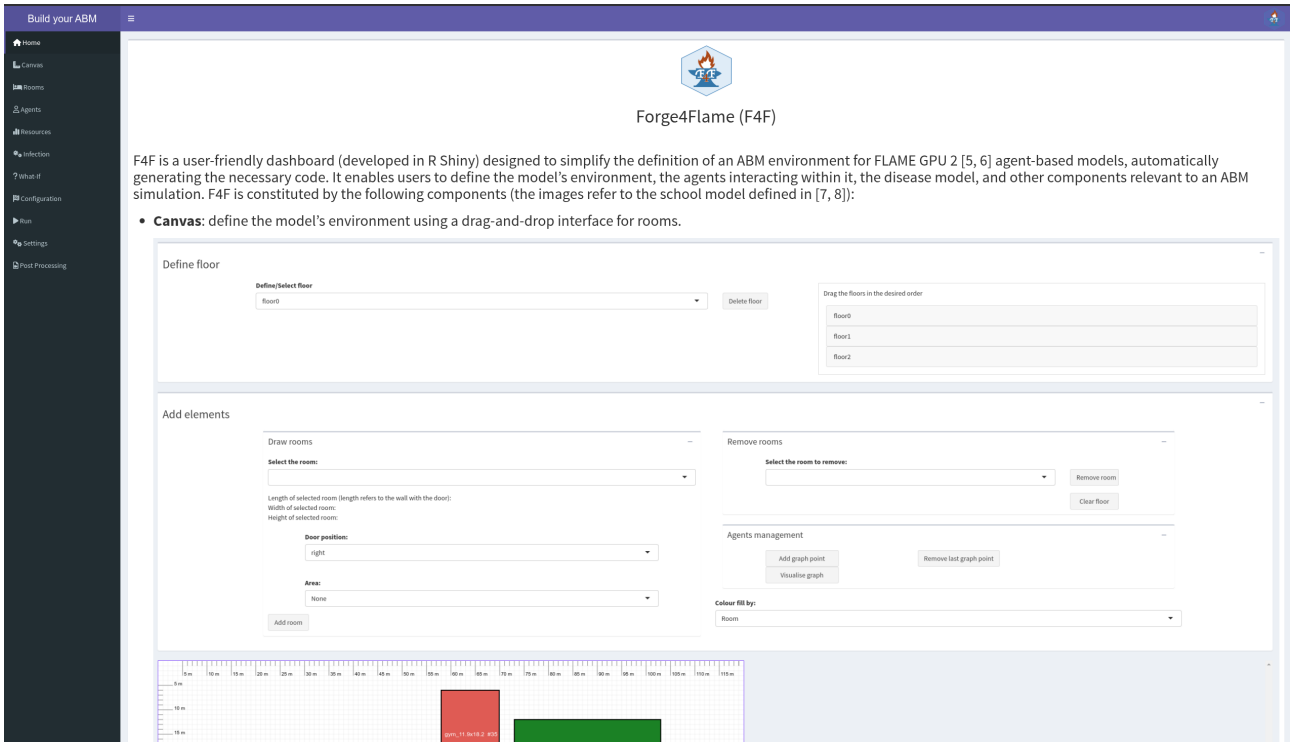

Figure 1: The **Homepage** page.

4.2 Canvas

Figure 2 shows the initial **Canvas** page. In the panel `Define floor`, a drop-down menu can be used to select a previously defined floor or to create a new floor giving it a name. The button *Delete floor* deletes the selected floor.

In the section *Draw rooms*, inside the panel `Add elements`, the user can add a previously defined type of room inside the canvas—by clicking on the button *Add room*—specifying the position of the door (none, bottom, left, top, right) and the area to which the room belongs (the user can define new areas or assign a room to no area). In the section *Remove rooms* the user can remove a room from the canvas by selecting it in the drop-down menu and clicking on the button *Remove room*. It is also possible to remove all the elements in the selected floor through the button *Clear floor*. The *Agent management* section is useful to create the graph of

movements used by agents and to visualize it—see Section 6 for more details on the graph of movement. The *Add graph point* button allows the user to insert a new node into the agents' movement graph. It is particularly useful to insert these graph points near the corners of the "corridors". Additionally, each room's door and room's centre area are represented by a node in the graph. In addition, it is possible to remove the last inserted graph point by clicking on the *Remove last graph point* button and to visualize all the graph's edges by clicking on the *Visualise graph* button. The drop-down menu named *Colour fill by* allows the user to colour rooms based on names, types, or areas. Finally, under this panel, there is the canvas, initially empty. Figure 3 shows the just created canvas associated with floor *floor0*—with the panel `Add elements` closed. Each square in the canvas represents 1x1 m^2.

Figure 4 shows the canvas of floor *floor0* with 16 rooms and 4 graph points. In particular, in this case, the colour of the room represents a previously defined room (that is, in the example light green represents the *room0*, dark blue the *room1*, gray the *fillingroom*, dark purple the *stairs*, and red the *spawnroom*). Moreover, Figure 5 shows the canvas of floor *floor1* with 15 rooms and 4 graph points (here we have two more defined rooms: dark pink representing the *room2*, and dark green the *room3*).

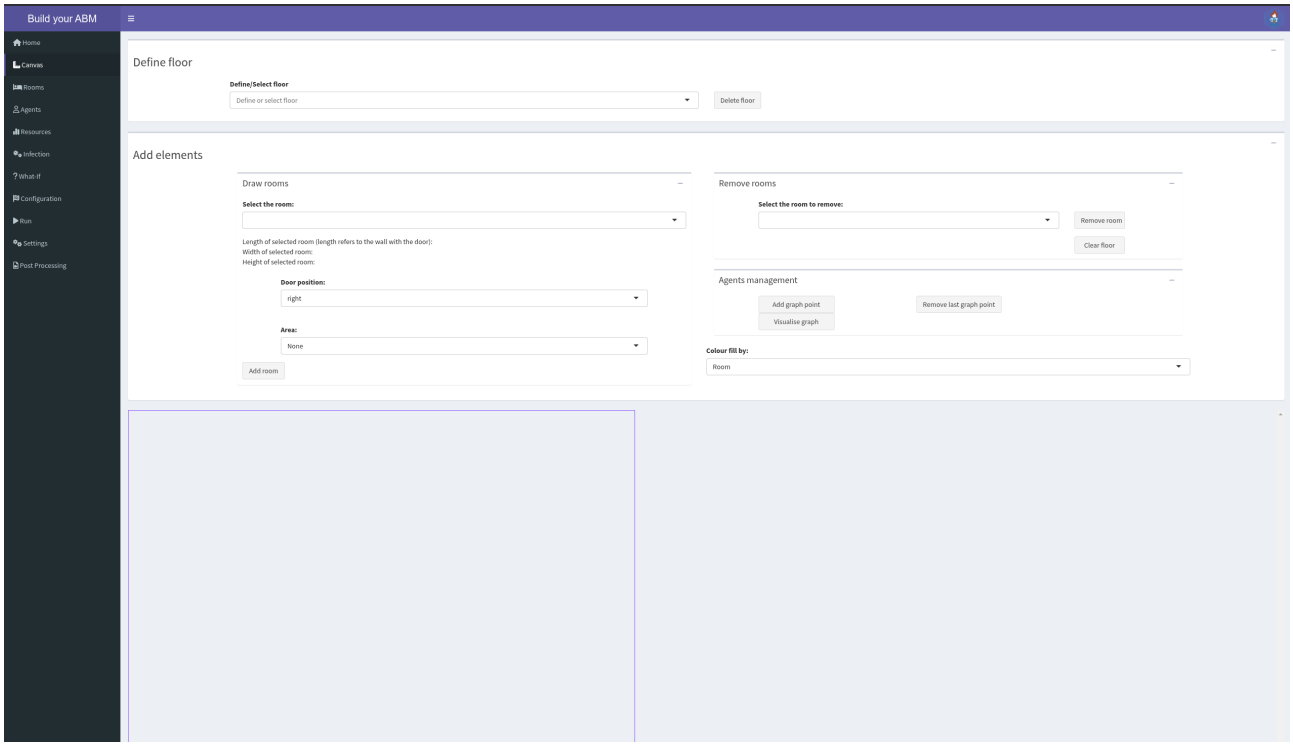

Figure 2: The **Canvas** page.

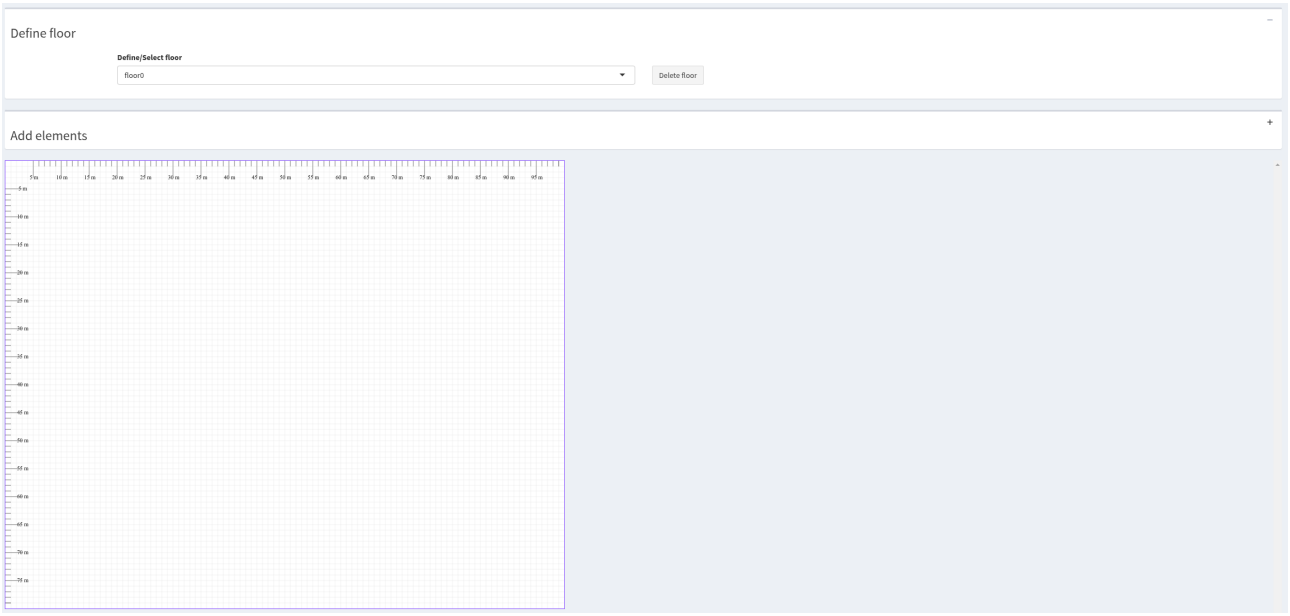

Figure 3: The **Canvas** page with the empty canvas of the *floor0* visualized.

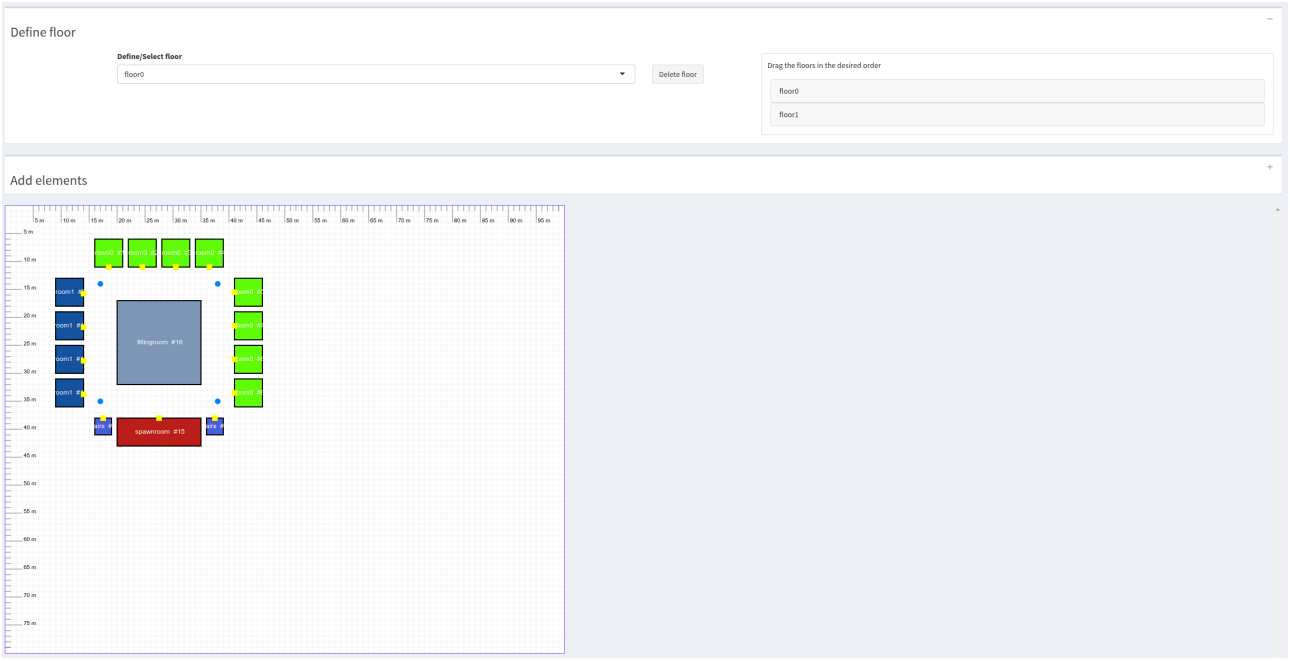

Figure 4: The **Canvas** page with some rooms and graph points inside the canvas *floor0*. Rooms are coloured by names.

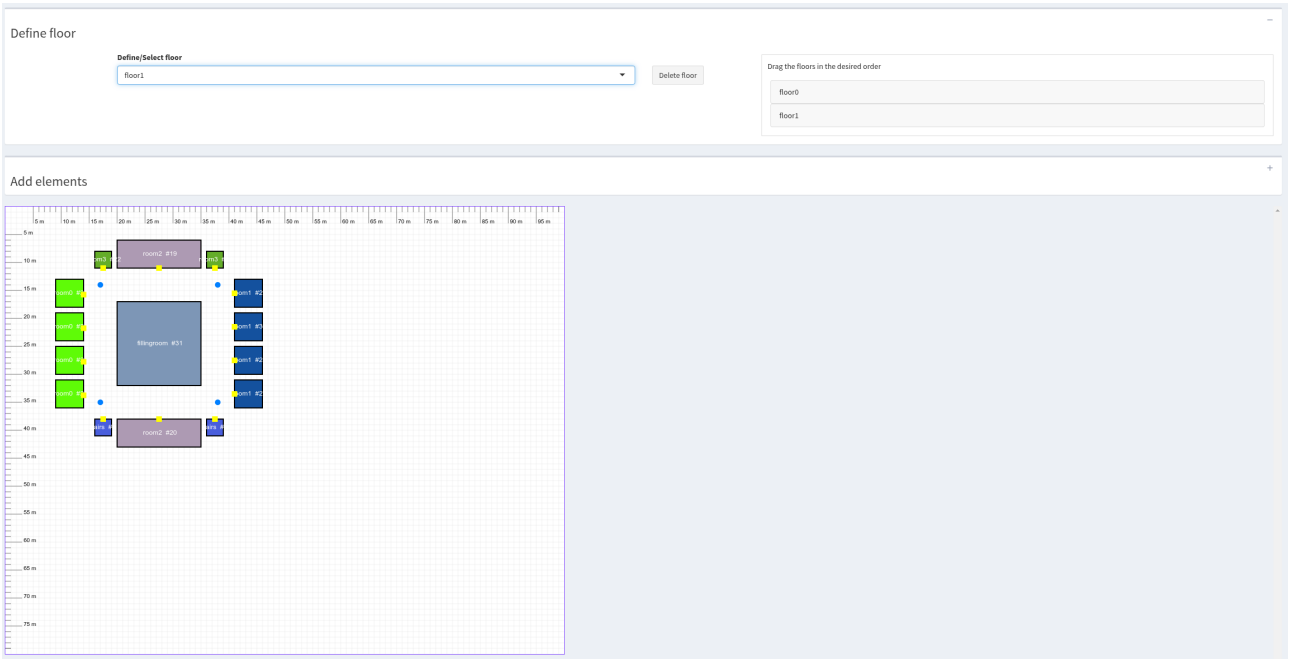

Figure 5: The **Canvas** page with some rooms and graph points inside the canvas *floor1*. Rooms are coloured by names.

4.3 Rooms

The **Rooms** page allows the user to define the rooms that will be placed inside the canvas. There are two panels inside this page, as shown in Figure 6.

The first, `Define a new room`, allows creating the room from scratch and the second, `Set colour legend`, allows modification of the colours associated with each defined room. To define a new room the user must specify a *Name*—the name of a room is case insensitive, i.e. *room1* and *ROOM1* represent the same room—, the *Length*, the *Width*, and the *Height* of the room. Notice that the length refers to the wall with the door. The height of a room will not be visible in the canvas but will be displayed in the FLAME GPU 2 visualization. It is essential for accurately implementing the aerosol contagion process—see Section 4.6 for more details.

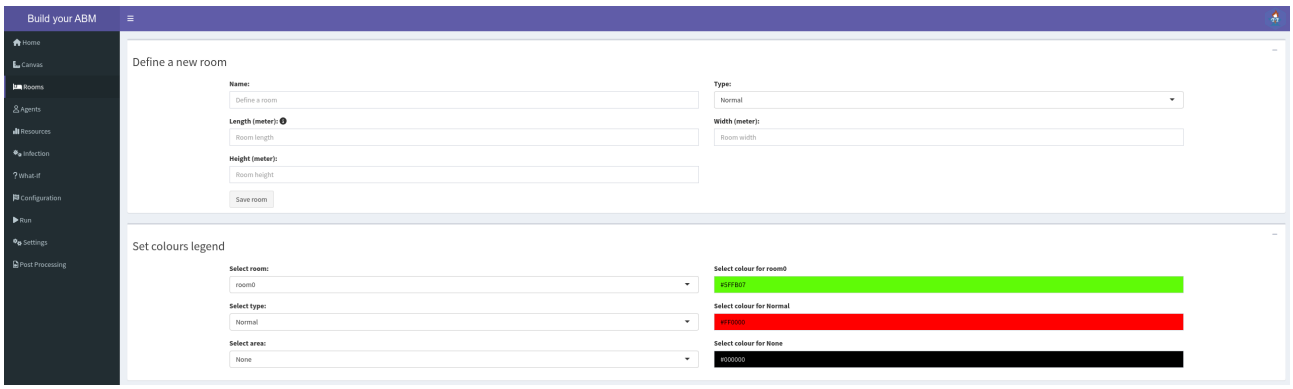

Figure 6: The **Rooms** page.

Rooms can be grouped into clusters based on a specific *Type*. This allows for the definition of a particular room type, instructing the agents to visit every room of that type. For instance, the user may define the generic type **bathroom**, defining a *small_bathroom* and a *large_bathroom*, instructing the agents to visit the generic **bathroom**. An example of the definition of the room

small_ bathroom can be seen in Figure 7. A set of default types are already defined inside our application:

- *Normal*: represents a generic room type.
- *Stairs*: room type used to move agents from one floor to another; stairs must be approximately in the same position on each floor.
- *Spawnroom*: room type from which agents enter and exit from the environment; currently, only one Spawnroom can be placed inside the canvas.
- *Fillingroom*: room type with no door; can be used to simulate the walls and rooms of no interest.
- *Waitingroom*: room type used by agents as a waiting area before entering another room when no resources are available.

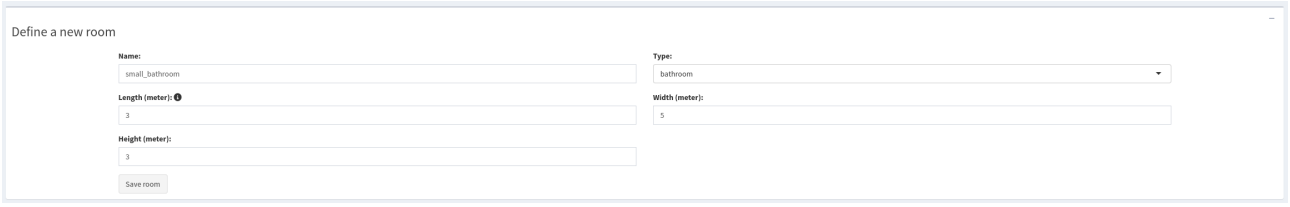

Figure 7: Definition of a room called *small_ bathroom* associated to a type called **bathroom**.

The panel `Set colour legend` allows the user to define the colours of individual rooms, rooms of a specific type, or rooms in a particular area, based on the chosen option from the drop-down menu. The associated colour will be displayed next to the selected item. To change the colour, simply click on it. A colour map will appear, enabling the user to select their preferred colour, as shown in Figure 8. These colours will be the ones the user sees on the canvas and in the FLAME GPU 2 simulation.

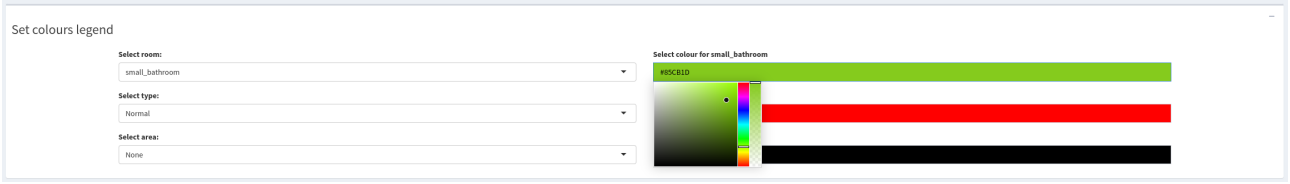

Figure 8: Selection of a particular colour for the room *small_ bathroom*.

4.4 Agents

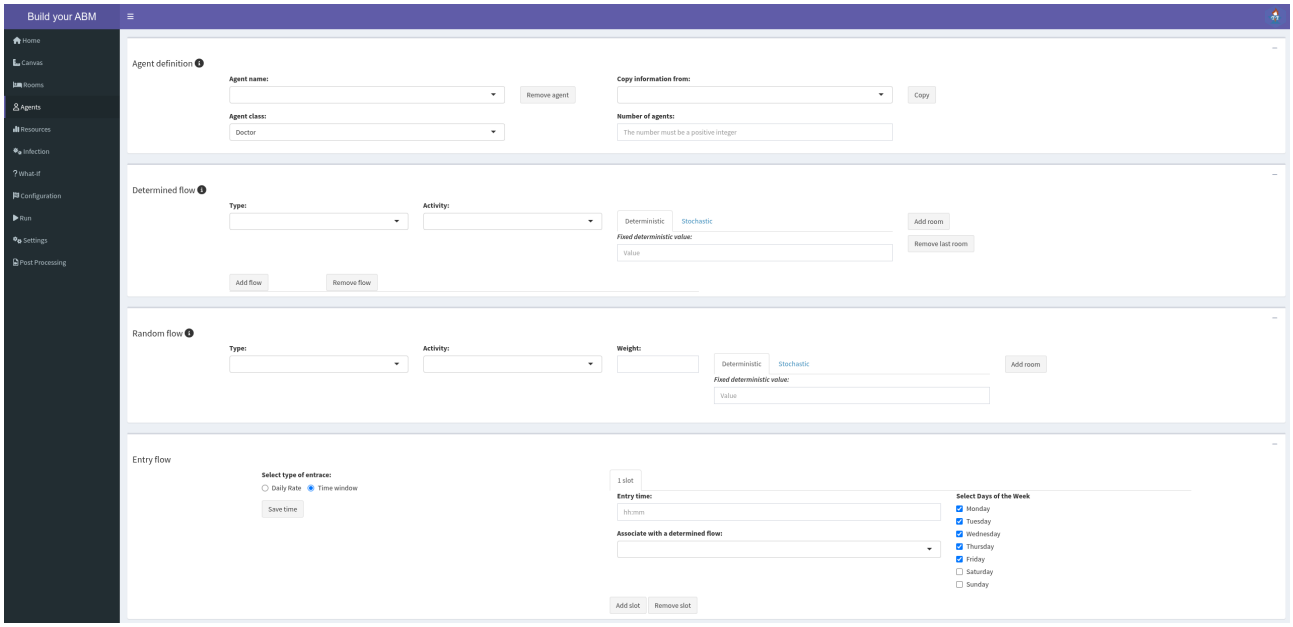

Figure 9: The **Agents** page.

The **Agents** page serves to define the agents that will interact with the environment. It comprises several panels—see Figure 9:

- In the `Agent definition` panel, the user can define all possible agent types. Each agent is defined by a name (*Agent name*) and can belong to a class (*Agent class*). A class is a category that groups different types of agents. Defining a broader class, such as *Doctor*, can be useful for organizing related agents like *Paediatrician*, *Neurologist*, *Surgeon*, and so on. Within this agent structure, the user can specify subclass-specific agents that belong to the same overarching group. For each specific agent, the user can define details like determined and random flows and entry times, which will be covered in the following sections. The user can also copy all information concerning an agent using the drop-down menu *Copy information from*, which allows the user to select an existing agent and then modify it as needed. Additionally, the user can specify the number of agents to include in the model by entering the desired amount in *Number of agents*. An example of this process is shown in Figure 10.

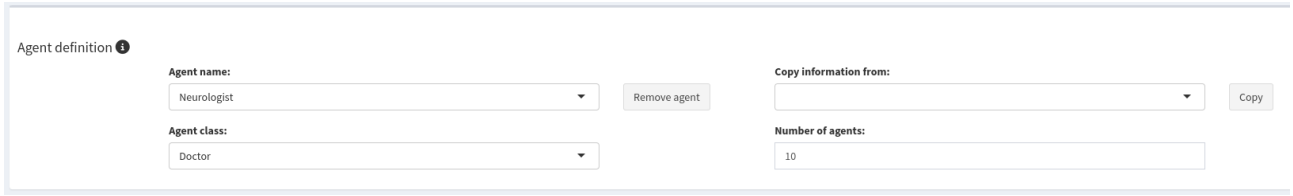

Figure 10: An example of the definition of an agent named *Neurologist*, which belongs to the class *Doctor*, with a total of 10 agents of that type in the model.

- In the `Determined flow` panel, the user can define one or more flows that an agent must follow within the model. There are three key elements to configure:

1. In the *Type* drop-down menu, the user can select the room to add to the flow. These rooms are those that have been added to the canvas—by type and area.
2. In the *Activity* drop-down menu, the user can select the intensity of the activity that the agent performs in the selected room. This setting is crucial for calculating the virus concentration within a room: higher activity intensity leads to greater virus spread.
3. The third and last element defines the duration of the task performed by the agent. The user can specify a deterministic time, which represents the exact number of minutes the agent spends on the task, or define the duration using a stochastic distribution—i.e., Exponential, Uniform, or Truncated Positive Normal. The distribution type can be selected from a drop-down menu, and its parameters can be entered below.

Rooms are added to the flow using the *Add Room* button, and the entire flow will be displayed below. The first room in the flow will appear at the top, with the agent following the subsequent rooms in descending order. Both the first and last rooms in the flow **must** be the room with type *Spawnroom* and area *None* to allow the agent to enter and exit the environment—the time associated the last *Spawnroom* is **negligible**, it will be automatically computed based on the next entry time. The order of rooms can be adjusted by selecting and moving them up or down within the flow. Additionally, the last room can be removed using the *Remove last room* button. Multiple flows can be defined for a single type of agent—except for agents with a daily rate, which can have only one flow, as explained below. An example of a defined flow is shown in Figure 11.

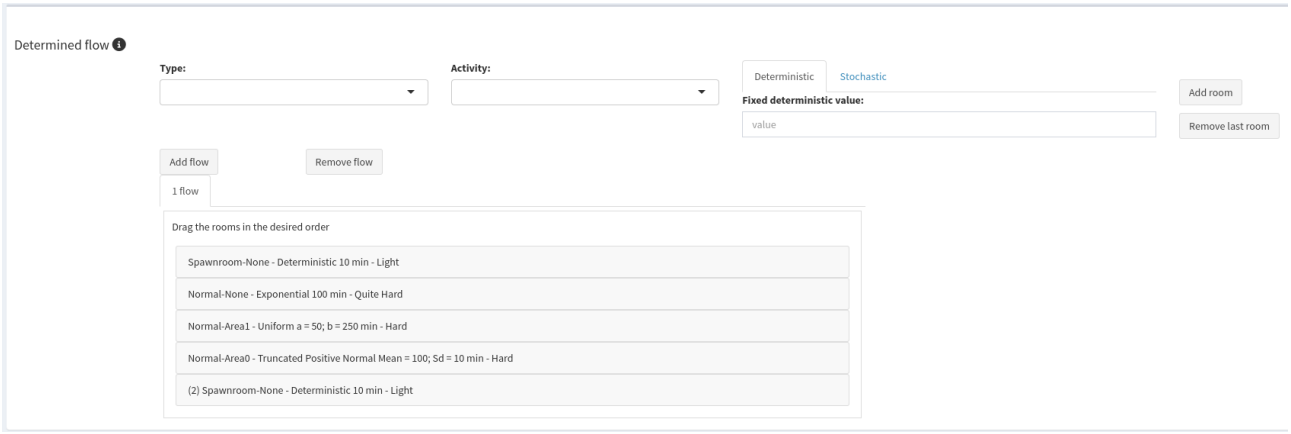

Figure 11: An example of a determined flow. Imagine that the agent is scheduled to enter the environment at 10:00 A.M. Then, it will *i)* remain outside the environment until 10:10 A.M, while engaging in a light activity, *ii)* move to a room of type *Normal* with area *None*, where it will perform a quite hard activity for a duration determined by an exponential distribution with an average of 100 minutes, *iii)* proceed to a room of type *Normal* with area *Area1*, where it will perform a hard activity for a duration determined by a uniform distribution with $a = 50$ minutes and $b = 250$ minutes, *iv)* move to a room of type *Normal* with area *Area0*, where it will perform a hard activity for a duration determined by a truncated positive normal distribution with a mean of 100 minutes and standard deviation of 10 minutes, and finally *v)* leave the environment.

- In the `Random flow` panel, the user can add random events that interrupt the agent's determined flow, i.e. directing the agent to an additional room—a random event should

occur infrequently and last for a short duration (few minutes). The elements are similar to those in the previous section, including the *Type* drop-down menu for selecting rooms—by type and area—, the *Activity* drop-down menu for selecting the activity type, and the definition of the time spent in that room. The *Weight* field allows the user to specify the probability of the event occurring by entering a number between 0 and 1. By default, the agent will follow the determined flow without any events. When a new event (e.g., going to the toilet) is added with a probability p, the probability of following the determined flow in each simulation step becomes $1-p$. A random event should happen rarely and last only a few minutes. If an event lasts too long, it may be cut short due to the completion of a block in the determined flow. For example, consider an event e with a duration of t_e, while the agent is required to remain in its current position for t_{df}—where df represents the current block in the determined flow. If $t_e > t_{df}$, the agent will experience the event for t_{df} to maintain consistency with the determined flow. In each simulation step, at most one event can occur and the probabilities are to be considered **per simulation step**—for more details on the simulation step, see Section 4.8. An agent is restricted from triggering new events while actively handling an ongoing event to avoid excessively long or infinite event loops. New events can only be initiated once the current event has been completed. In addition, an event can be removed by simply clicking on it—an alert will appear asking for confirmation to delete or cancel the action. An example of a random flow is shown in Figure 12.

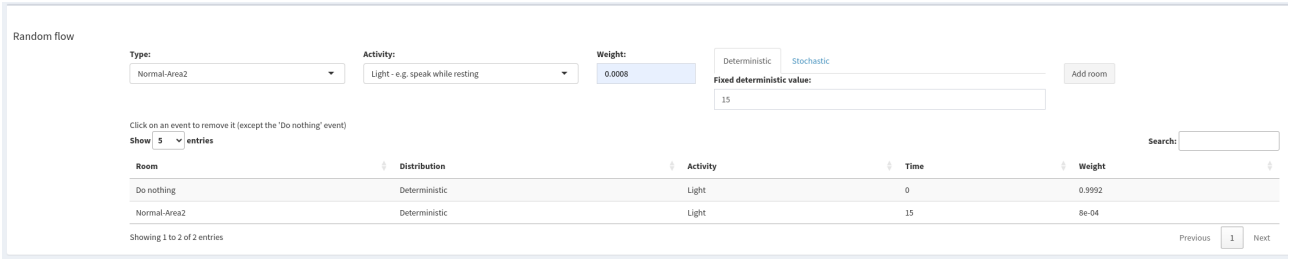

Figure 12: An example of random flow. At each simulation step, the agent has a probability of 0.0008 to move to a room of type *Normal* with area *Area2* with light activity and will spend exactly 15 minutes there.

- The last panel is called `Entry flow`. It allows the user to define when the agent enters the environment and starts its determined flow. There are two ways to define an agent's entry flow:
  - By a **daily rate**: the user can define an entrance rate for the agent using either a deterministic value or a stochastic distribution, along with specifying an initial and a final generation time. Agents will be generated inside this time window. If a deterministic value is used, the exact number of agents specified will be generated, otherwise, the number of agents created will be determined according to the stochastic distribution parameters. Generated agents will follow the determined flow and leave the environment upon completion—these agents can have only one determined flow. Additionally, the user can set various entry rates for different time windows and specify particular days of the week for agent generation. See Figures 13 and 14 for an example of a definition of a daily rate entry flow.
  - By a **time window**: in this case, the number of agents is specified in the `Agent definition` panel—see Figure 10. The user sets the entry time and the specific days of the week when the agent will enter the environment—it is possible to define

multiple time slots. Additionally, the user can define different flows—see Figure 11—associating one of them to each time slot. These agents will repeat the selected flow each week throughout the simulation. The total duration of a determined flow can only be estimated (especially if the user employs stochastic distributions). We check for possible overlaps among the entry flows using the average of the selected distributions. In some cases, overlaps may occur. In these cases, when the agent finishes the current determined flow, it will start the next one with some delay. Our suggestion is to avoid placing different entry flows too close to each other. See Figure 15 for an example of the definition of a time window entry flow.

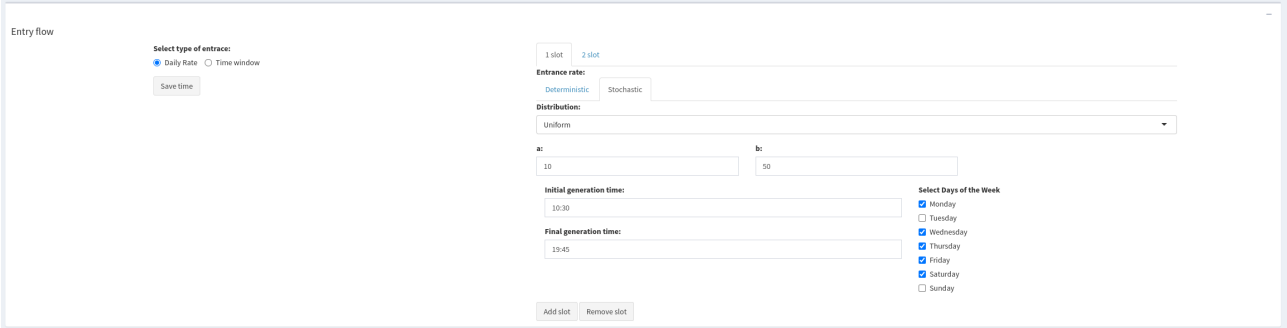

Figure 13: An example of a daily rate entry flow (slot 1). Agents will be generated between 10:30 A.M. and 7:45 P.M. on every weekday except Tuesday and Sunday, with their number determined stochastically using a uniform distribution with $a = 10$ agents and $b = 50$ agents.

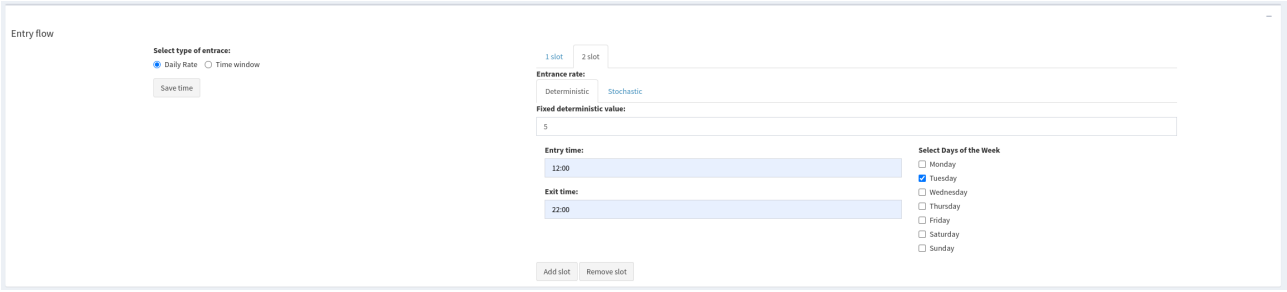

Figure 14: An example of a daily rate entry flow (slot 2). Five agents will be generated between 12:00 A.M. and 10:00 P.M. on Tuesday and Sunday.

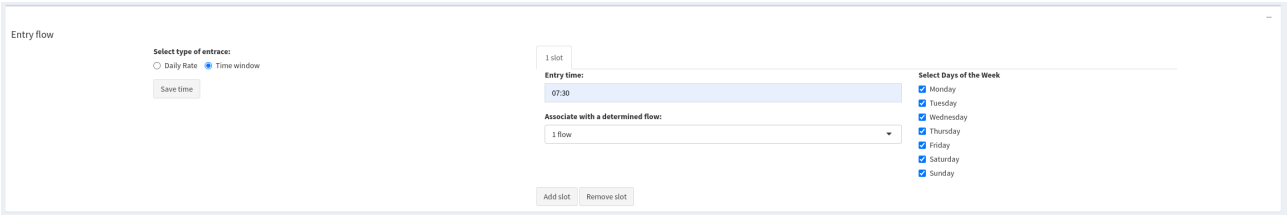

Figure 15: An example of a time window entry flow. Agents will enter the environment at 7:30 A.M. every day of the week, following the first flow.

4.5 Resources

The **Resources** page allows the user to specify the maximum number of agents that can occupy a room simultaneously—see Figure 16. It also provides options to set limits based on the type of agent.

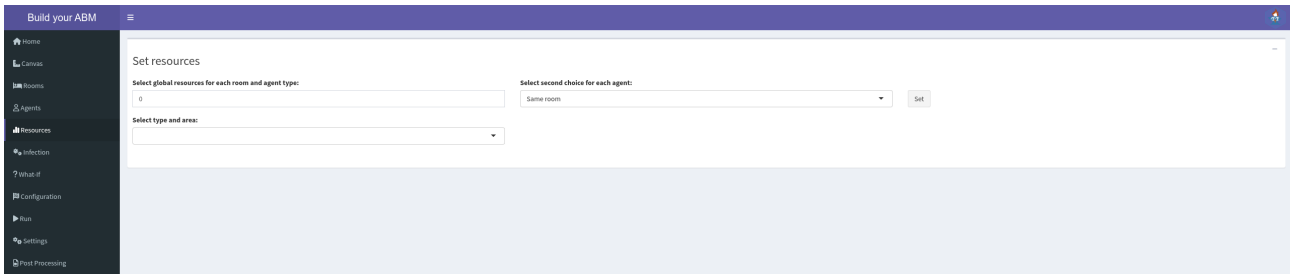

Figure 16: The **Resources** page.

This page consists solely of the `Set Resources` panel, as illustrated in Figure 17. Within this panel, two global values can be configured:

- **Maximum number of agents:** users can specify globally the maximum number of agents allowed to enter each room in the environment. This is particularly useful when the user is not interested in modelling resource limitations and prefers to set them to high values.
- **Global alternative for room entry:** if an agent is unable to enter a room in its flow due to capacity constraints, the user can define an alternative option based on the associated resources. The available alternatives include:
  - *Same room*: search for another room of the same type and area.
  - *Skip room*: proceed to the next room in the flow or skip the event (in the case an event occurred).
  - *Another room*: select a different room with a different type and area.

To set this value, simply enter the desired number in the input field and press the *Set* button.

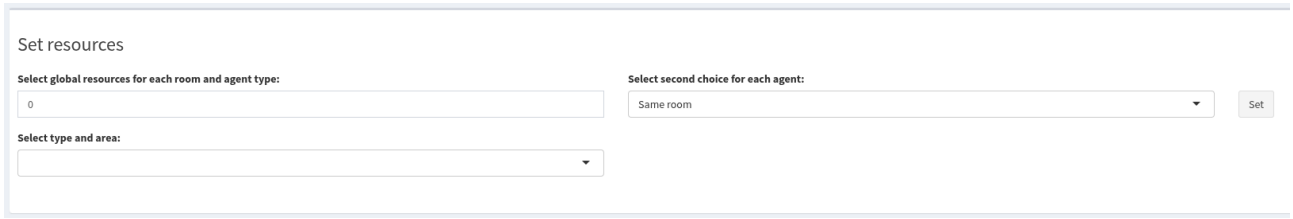

Figure 17: The **Resources** page.

The third element of the section is a drop-down menu labelled *Select Type and Area*, which allows the user to choose a room to associate resources with, categorized by type and area. By selecting a room based on its type and area, the user can modify the associated resources. As shown in Figure 18, a table displays all rooms of the selected type an d area that are associated with at least one agent's flow.

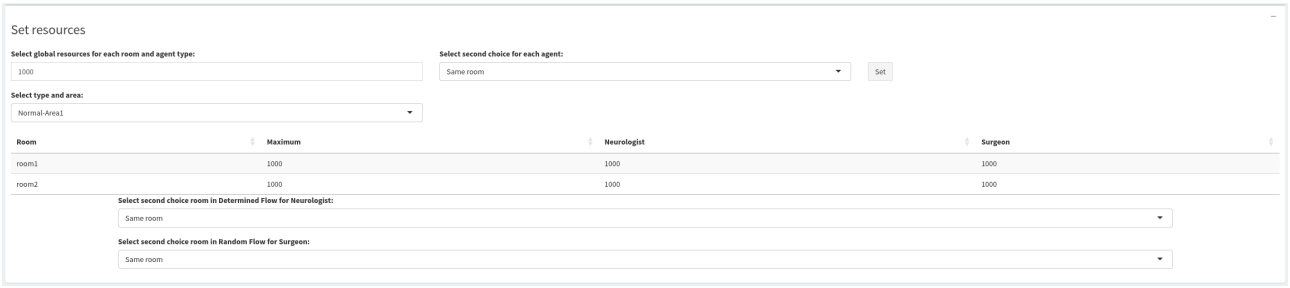

Figure 18: The resources associated with rooms of type *Normal* and area *Area1* are displayed. This room appears in the predefined flows of the *Neurologist* agent and in the random flow of the *Surgeon* agent.

For each room, the user can view and modify the number of agents allowed to enter by double-clicking the respective field. The total number of agents permitted in the room can be adjusted in the *Maximum* column. Additionally, the user can specify the allowed number of agents for each type, with each type displayed in a separate column corresponding to the room. If an agent finds the room full when attempting to enter, the user can specify an alternative room using the *Select second choice room in Determined/Random Flow for* ***Agent*** drop-down menu for each agent. The user can assign different alternative rooms for an agent's determined and random flow, as shown in Figure 18.

4.6 Infection

The **Infection** page—see Figure 19—allows the user to choose from eight different disease compartmental models:

- SIR: Susceptible-Infected-Recovered.
- SIRD: Susceptible-Infected-Recovered-Deceased.
- SEIR: Susceptible-Exposed-Infected-Recovered.
- SEIRD: Susceptible-Exposed-Infected-Recovered-Deceased.
- SIRS: Susceptible-Infected-Recovered-Susceptible.
- SIRDS: Susceptible-Infected-Recovered-Deceased-Susceptible.
- SEIRS: Susceptible-Exposed-Infected-Recovered-Susceptible.
- SEIRDS: Susceptible-Exposed-Infected-Recovered-Deceased-Susceptible.

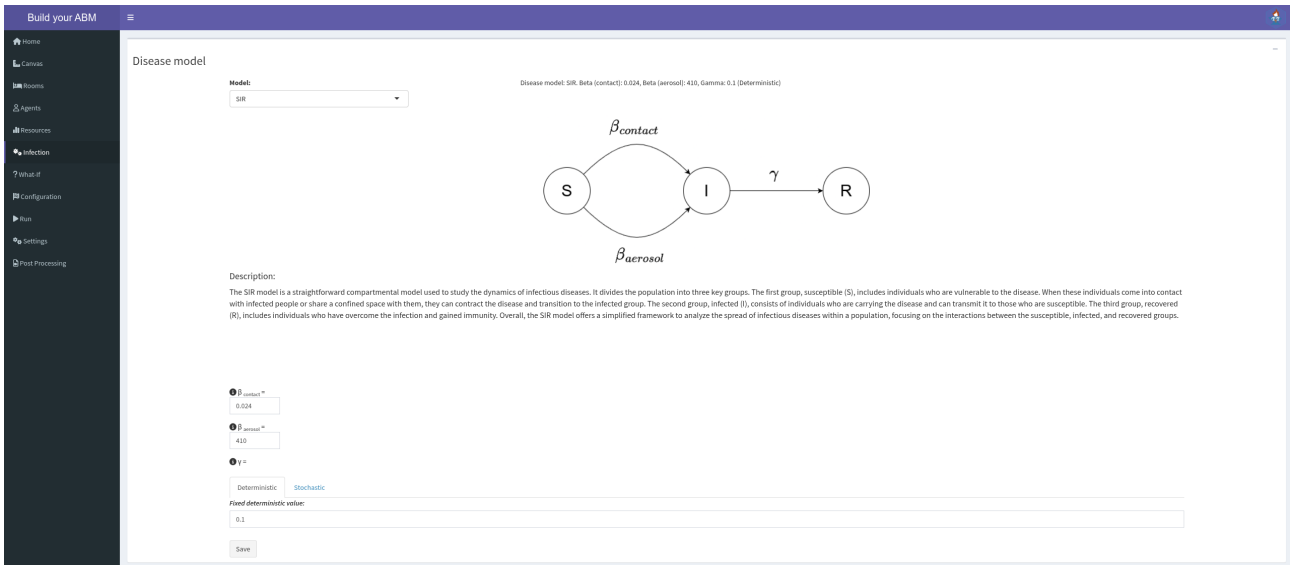

Figure 19: The **Infection** page with the SIR compartmental model shown.

A brief description and a graphical schema are provided for each disease model. Based on the chosen model, the user must specify values for the various parameters of that model. In particular, all the disease models share three parameters: *i)* $\beta_{contact}$, which represents the contamination risk associated with infection from close-range contacts, as modelled in [17]—for example, for the COVID-19 disease the value to use is 0.024—, *ii)* $\beta_{aerosol}$, which represents the risk constant associated with infection due to aerosol, as modelled in [15]—for example, for the COVID-19 disease the value to use is 410—, and *iii)* γ, which represents the recovery rate. **Note that** $\beta_{contact}$ **and** $\beta_{aerosol}$ **are not the infection rates.** For each parameter, except $\beta_{contact}$ and $\beta_{aerosol}$, the user can choose a deterministic value or a stochastic distribution. After specifying the parameters, click on the *Save* button to save them.

In Figure 20 the SEIRDS model is shown. This case is characterize by six parameters: *i)* $\beta_{contact}$, *ii)* $\beta_{aerosol}$, *iii)* γ, *iv)* α, which represents the incubation rate, *v)* λ, which represents the fatality rate, and *vi)* ν, which represents the end-of-immunization rate.

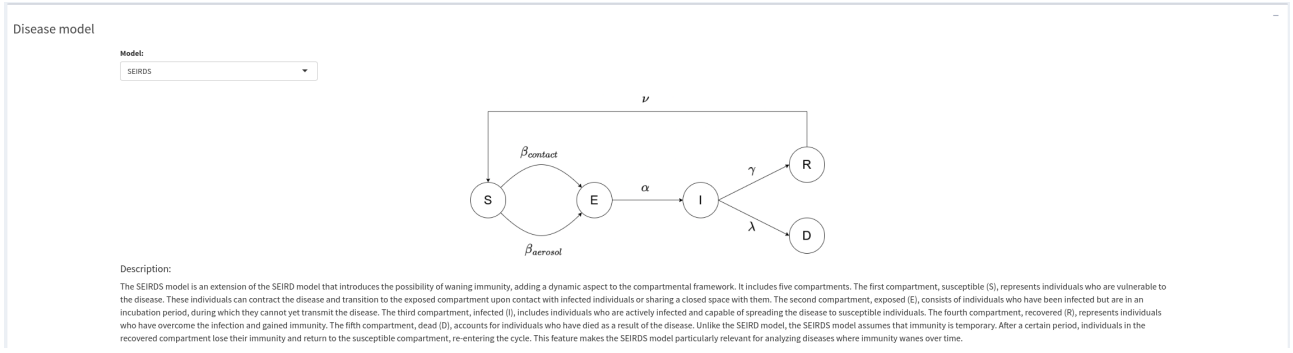

Figure 20: The SEIRDS compartmental model.

The virus exposure is modelled through two different drivers: either close-range contacts [17] between susceptible and infected agents or through aerosol [15] within an indoor environment.

In the former case, the probability of successful contagious contact is associated with each susceptible agent (S_i) that remains within an area (**A**) of $3.46\,m^2$ surrounding an infected agent—in contrast to the model presented in [5, 6], we considered a circle with a radius of $1.05\,m$ around each agent, instead of a square with area $2.1 \cdot 2.1\,m^2$; we chose $1.05\,m$ because it

corresponds to the apothem of the square, but we are considering making this value a parameter that the user can specify through the application. This probability is determined by the following function [17]:

$$P_{S_i} = \beta_{contact} \frac{C_{S_i}}{\mathbf{A}} \tag{1}$$

where $\beta_{contact}$ represents the *contamination risk*, and C_{S_i} denotes the total time (in minutes) that the S_i agent remains within the vicinity of an infected agent.

In contrast, the spread of the pathogen via aerosol is calculated based on the *quanta* concentration in each room. A quanta is defined as the dose of airborne droplet nuclei necessary to infect 63% of susceptible individuals [8]. Following the approach in [15], we determine the number of quanta inhaled by each agent in a given room to compute the probability of infection without direct contact with an infected agent. This probability is given by the following function:

$$P_{S_i} = 1 - e^{-\frac{N_{\text{virus}}}{\beta_{aerosol}}} \tag{2}$$

where $\beta_{aerosol}$ represents the reciprocal of the probability that a single pathogen particle will trigger an infection [15, 28], and N_{virus} denotes the number of quanta inhaled by each susceptible agent during a time interval Δt. This value is determined by solving the following differential equation:

$$dN_{virus} = (1 - \eta_{mask} f_{\text{mask}})\ C(t) Q_{inh}\ dt \tag{3}$$

where η_{mask} is the efficacy of the masks, f_{mask} represents the proportion of agents correctly wearing a mask, $C(t)$ denotes the quanta concentration at time t, and Q_{inh} is the inhalation rate. Observe that quantum concentration at time t mainly depends on the number of people within a room, the number of infections, and the room's ventilation.

4.7 What-If

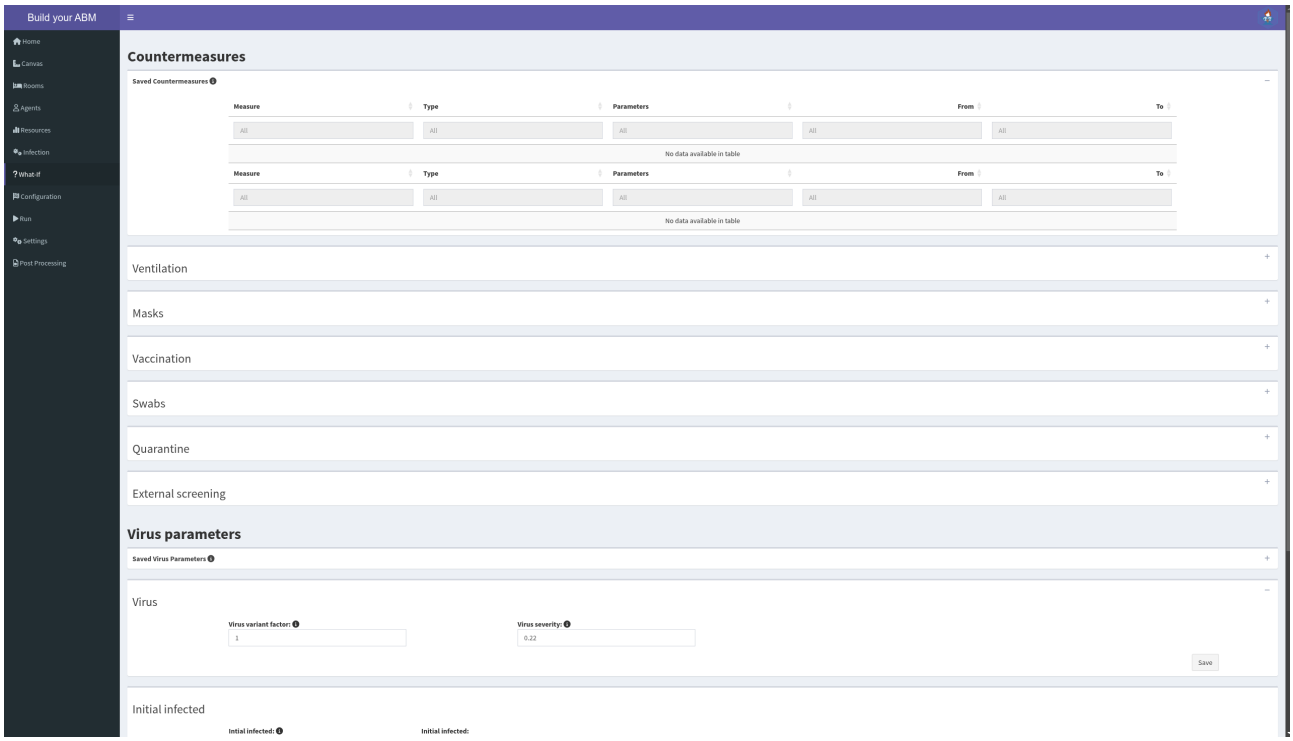

Figure 21: The **What-If** page.

The **What-If** page—see Figure 21—enables the user to conduct various what-if scenarios considering different *i)* levels of environmental ventilation, *ii)* masks policies, *iii)* levels of vaccination coverage and efficacy, *iv)* screening policies, *v)* quarantine policies, *vi)* external screening policies, *vii)* virus variants and severity of symptoms, *viii)* initial infected types and numbers, and *ix)* levels of outside contagion. The page is divided into two macro-sections: **Countermeasures** and **Virus Parameters**. If the user does not specify a particular countermeasure, it will not be applied. For the virus parameters, the default values will be used—see below.

For each countermeasure, the user must specify the period during which it will be in effect. For instance, the user can specify a particular type of mask with a defined fraction of agents wearing it from day d_1 to d_2 and apply a different setting from $d_2 + 1$ to d_3—with $d_1 < d_2 < d_3$. Clicking the *Save* button the specified settings will be saved—this applies to both the countermeasures and the virus parameters. In details the **Countermeasures** section contains:

- The `Ventilation` panel allows the user to define either a global ventilation level—Figure 22—, which is the same for each room, or specific ventilation levels—Figure 23—which can differ for each room [15]. There are 7 possible ventilation levels in ACH[1], i.e. 0 (no ventilation), 0.3 (poorly ventilated), 1 (domestic), 3 (offices/schools), 5 (well-ventilated), 10 (typical maximum), and 20 (hospital setting). The ventilation level has an impact on the aerosol contagion process—see Section 4.6.

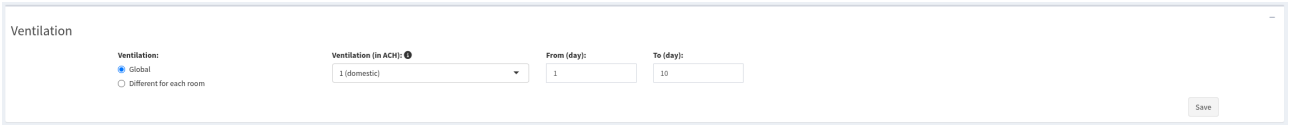

Figure 22: Setting ventilation globally from day 1 to 10.

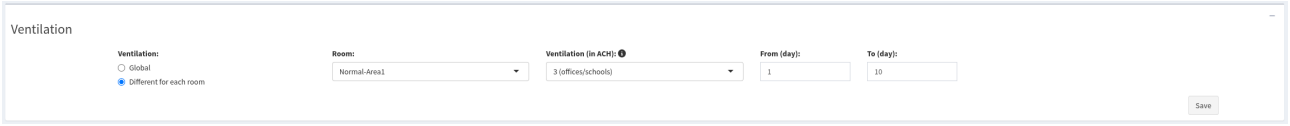

Figure 23: Setting ventilation specifically to rooms with type *Normal* and area *Area1* from day 1 to 10.

- The `Masks` panel allows the user to define either a global mask policy—Figure 24—, which is the same for each agent, or a specific mask policy—Figure 25—which can differ for each agent. There are 3 different types of masks [23], i.e. no mask (0% of exhalation and inhalation efficacy), surgical mask (59% of exhalation and inhalation efficacy) and FFP2 mask (90% of exhalation and inhalation efficacy). The user can also specify the fraction of agents that will wear the selected mask, either globally or for a specific agent.

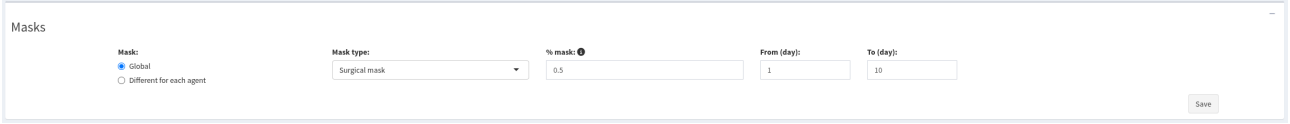

Figure 24: Setting mask policy globally from day 1 to 10.

[1]For instance, 3 Air Changes per Hour (ACH) means that in 1 hour 300.000 L (or analogous 300 squared meters) of external air are entered into the considered room.

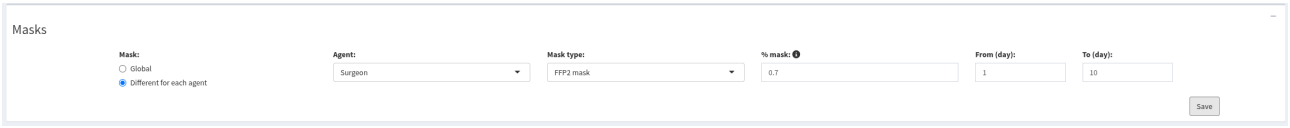

Figure 25: Setting mask policy specifically to agents with type *Surgeon* from day 1 to 10.

- The `Vaccination` panel allows the user to define either a global vaccination policy—Figure 26—, which is the same for each agent, or a specific vaccination policy—Figure 27—which can differ for each agent. The user must specify the fraction of agents that will be vaccinated and the efficacy of the vaccines. We define vaccination efficacy as the proportion of vaccinated agents that are effectively immune to infection, while the remainder remain susceptible despite being vaccinated. It is also necessary to set a vaccine coverage, which determines the number of days the vaccination will protect the agent. The user can configure this using either a deterministic value or a stochastic distribution, as described previously. In this case, the countermeasure will be applied on the specified day. For instance, if the user specifies a fraction f of vaccinated agents with an efficacy e and a coverage c on day d, those agents will be vaccinated on that day. In this case, there is no *From* and *To* period.

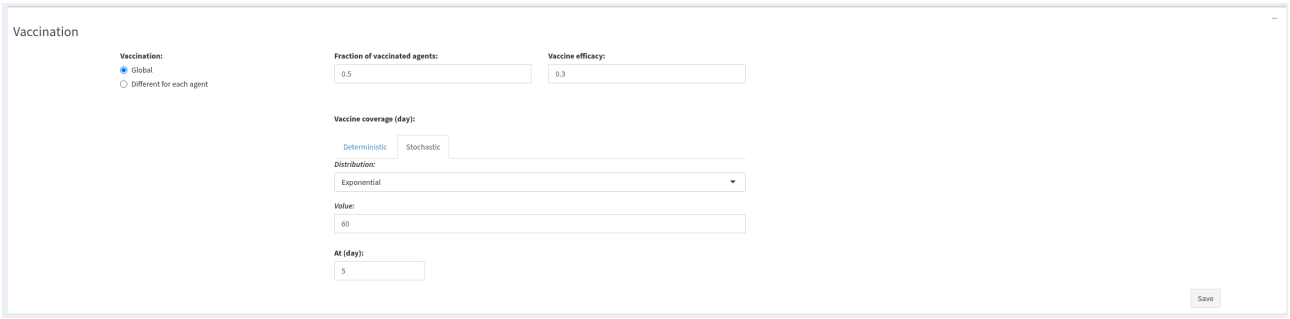

Figure 26: Setting vaccination policy globally at day 5.

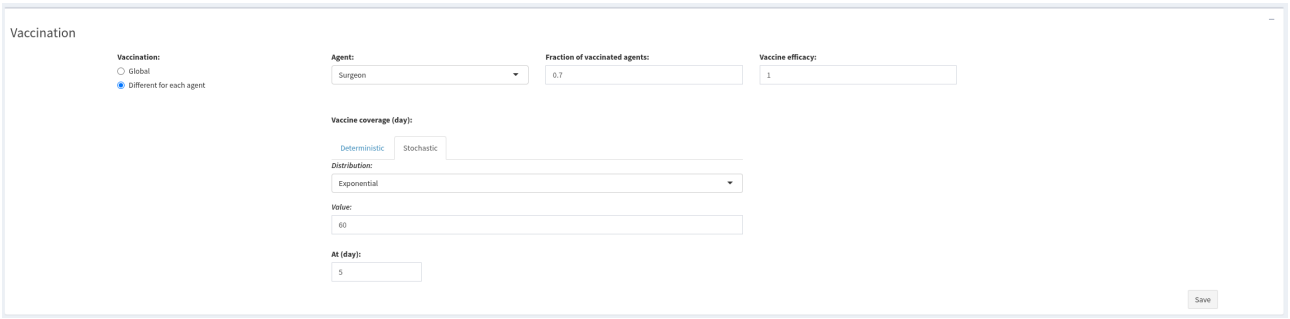

Figure 27: Setting vaccination policy specifically to agents with type *Surgeon* at day 5.

- The `Swabs` panel allows the user to define either a global policy—Figure 28—, which is the same for each agent, or a specific policy—Figure 29—which can differ for each agent. The user must specify the sensitivity and specificity of the swabs, as well as the frequency (deterministic or stochastic) at which swabs are administered to each agent.

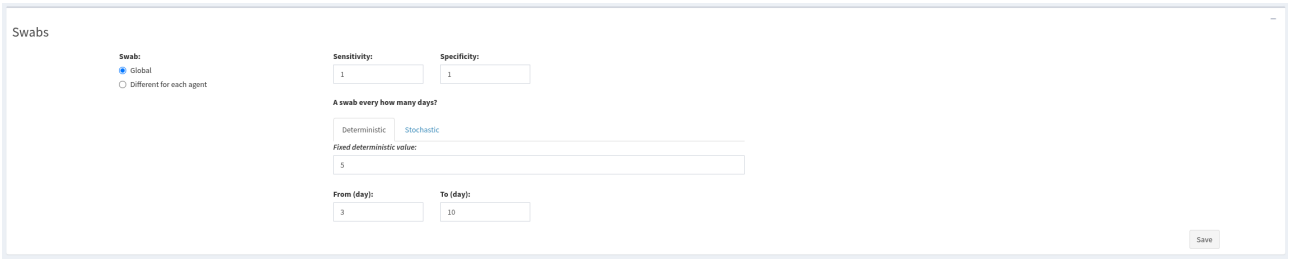

Figure 28: Setting swab policy globally from day 3 to 10.

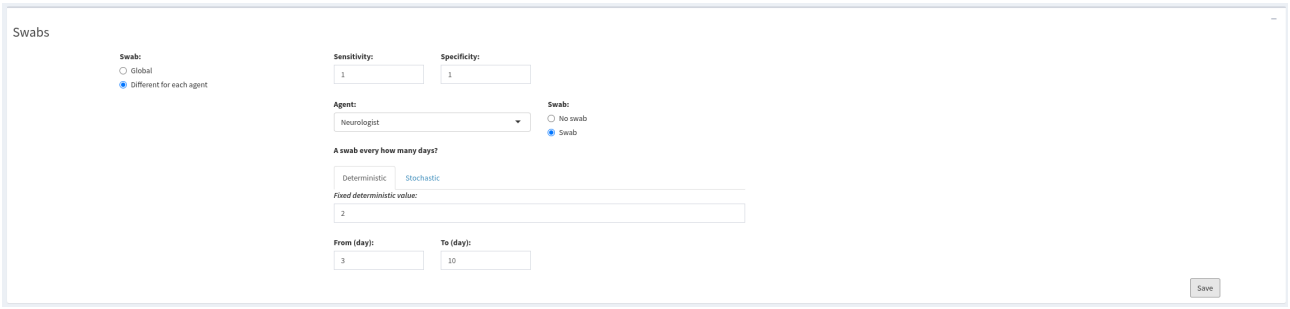

Figure 29: Setting swab policy specifically to agents with type *Neurologist* from day 3 to 10.

- The `Quarantine` panel allows the user to define either a global policy—Figure 30—, the same for each agent, or a specific policy—Figure 31—which can differ for each agent. The user must specify the duration of the quarantine in days (deterministic or stochastic) and the room used to put in quarantine agents with severe cases—the default is outside the environment. The user can also choose a swab policy, enabled only during quarantine, by specifying the frequency (deterministic or stochastic) at which swabs are administered to each agent, the sensitivity and the specificity.

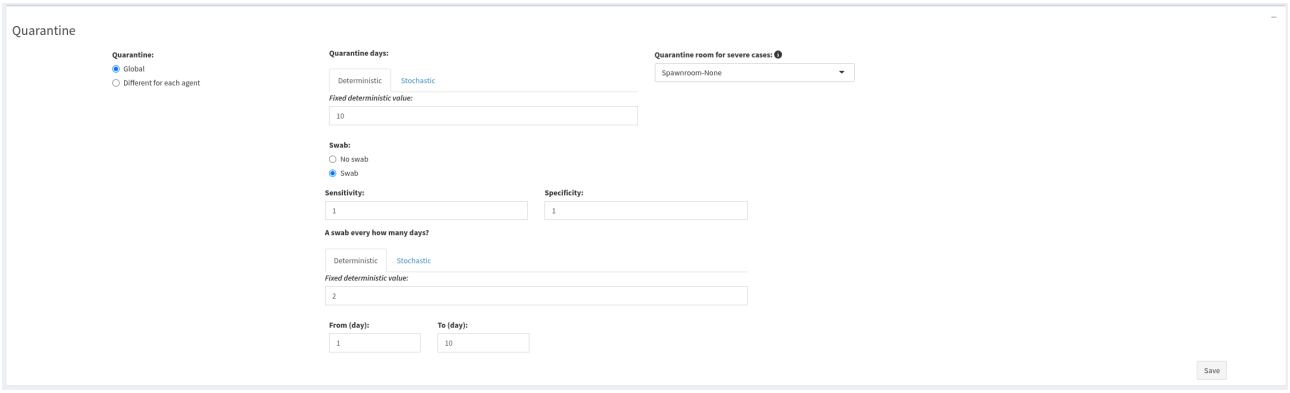

Figure 30: Setting quarantine policy globally from day 1 to 10.

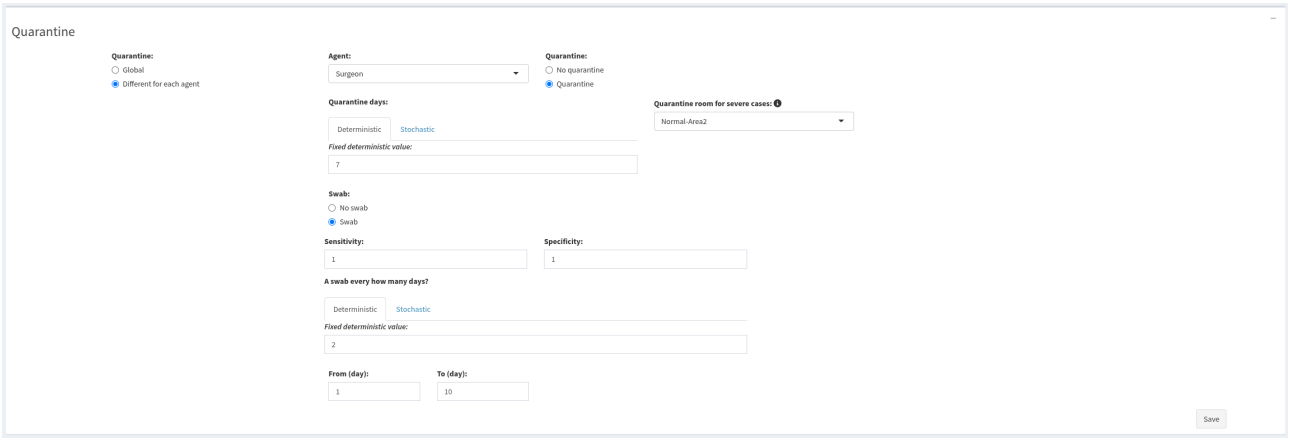

Figure 31: Setting quarantine policy specifically for agents with type *Surgeon* from day 1 to 10.

- The `External screening` panel allows the user to define either a global policy—Figure 32—,the same for each agent, or a specific policy—Figure 33—which can differ for each agent. In particular, two different probabilities of external screening can be applied to agents each day: the probability of testing an agent due to activities that involve screening campaigns (e.g., sports practice) and the probability of testing an agent outside the environment due to the presence of symptoms.

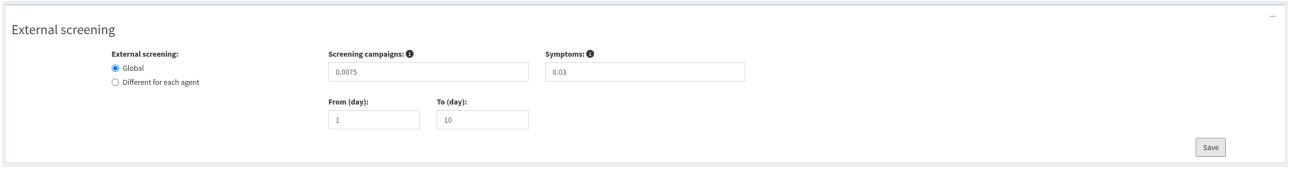

Figure 32: Setting external screening policy globally from day 1 to 10.

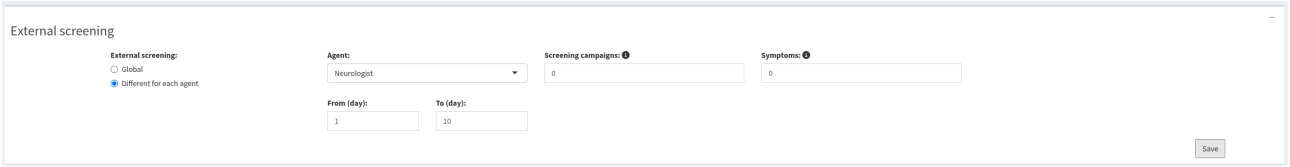

Figure 33: Setting external screening policy specifically for agents with type *Neurologist* from day 1 to 10.

At the beginning of the **What-If** page, all countermeasures set by the user are displayed, as shown in Figure 34, in the *Saved Countermeasures* panel. The first table contains countermeasures related to the rooms (i.e., ventilation), while the second table contains countermeasures related to agents. To remove one of the countermeasures, the user must click on the relevant entry and an alert will appear asking for confirmation to delete or cancel the action. It is also possible to display only specific countermeasures using the filter associated with each column. An example is shown in Figure 35, where only the mask policy is displayed. **Specific countermeasures will take precedence over global ones.** For instance, as shown in Figure 34, 50% of agents will wear a surgical mask from day 1 to day 10, with the exception of surgeons, 70% of whom will wear an FFP2 mask.

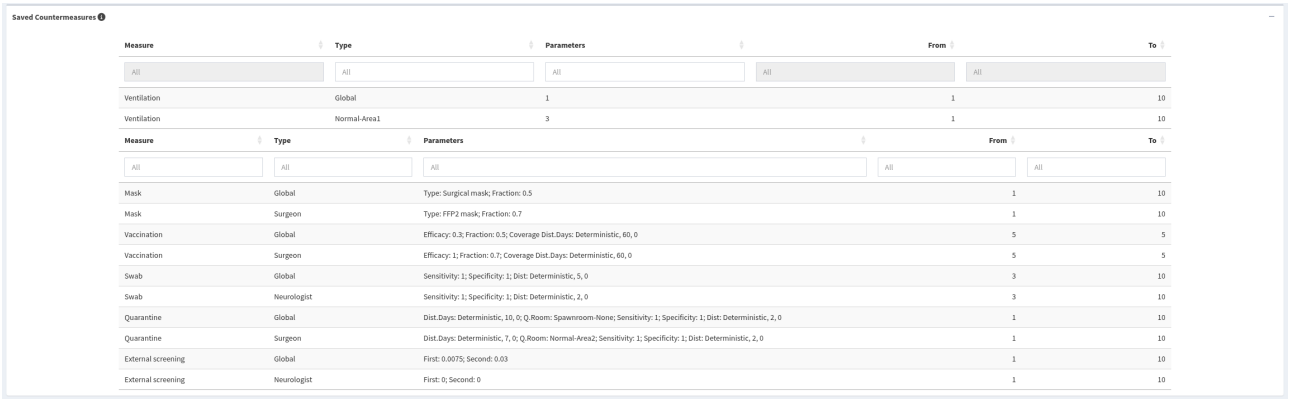

Figure 34: Saved countermeasures listed in the panel *Saved Countermeasures*.

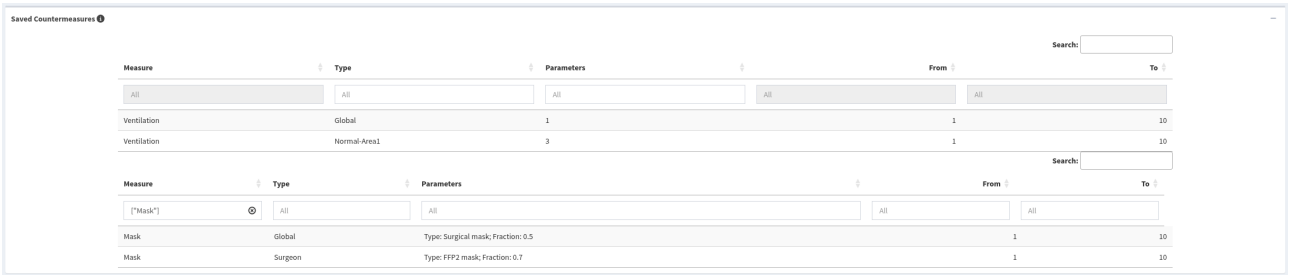

Figure 35: Example of filter on the *Measure* column.

On the other hand, the **Virus parameters** section contains:

- The `Virus` panel allows the user to define the virus—or the variant of the virus—to model and the severity of symptoms—Figure 36. Use the value 1 as *Virus variant factor* if there are no variants or if the user wants to model the base variant—in [23] the user can find some examples for COVID-19. The *Virus severity* represents the probability of showing severe symptoms—in [26] the user can find an example of COVID-19.

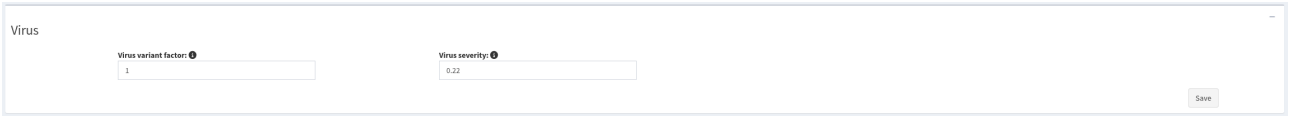

Figure 36: Definition of virus parameters.

- The `Initial infected` panel allows the user to define the number of agents that will be initially infected. There are three options:

  1. Random: if the number of initial infected is n, then these n agents will be randomly selected from all existing agents—Figure 37.
  2. Global: if the number of initial infected is n, then n agents of each agent type will be initially infected—Figure 38.
  3. Specific: for each agent type, the user can indicate the number of agents of that type that will be initially infected—Figure 39.

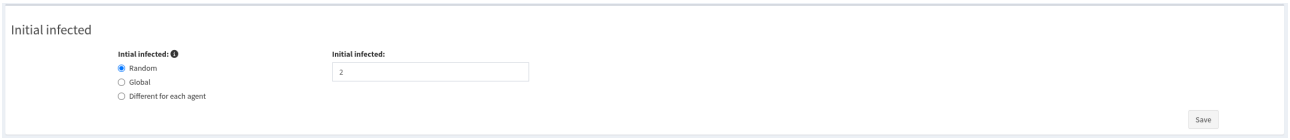

Figure 37: Setting random initial infected.

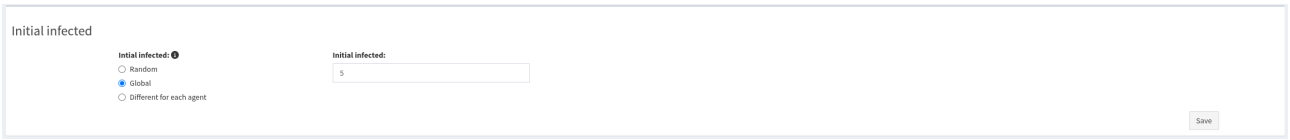

Figure 38: Setting initially infected globally.

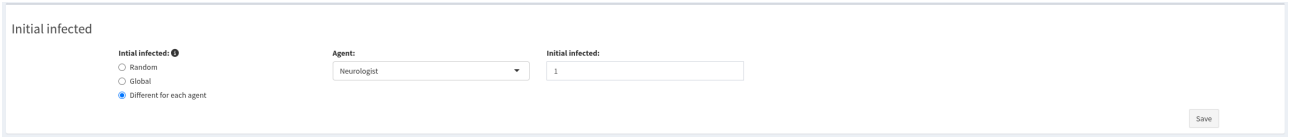

Figure 39: Setting initially infected specifically.

- The `Outside contagion` panel—Figure 40—allows users to model infections occurring outside the environment as a stochastic arrival process. This setup aligns the rate of external infections with the prevalence of the disease in the broader population. In particular, a probability is applied to each susceptible agent every day, representing the possibility of infection outside the environment. If the user wishes to model the infection outside the environment, they must upload a CSV file containing two columns: *day*, representing the simulation day, and *percentage_ infected*, which indicates either the actual number of infected individuals in the considered population or the probability of acquiring the infection outside the environment on that day. If the former is provided, the user must specify the total population in the *Population* field; otherwise, they should set the population to 1. A plot is displayed after the file is uploaded—Figure 41.

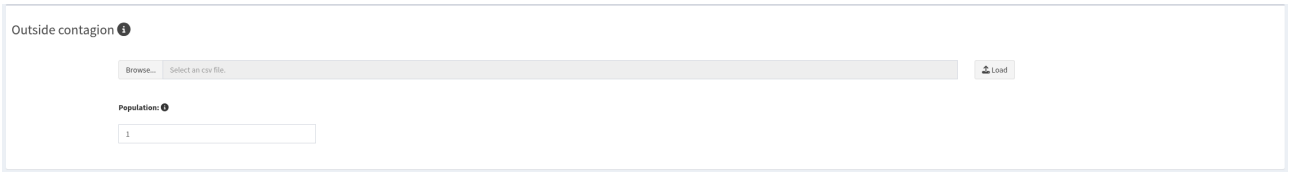

Figure 40: Outside contagion panel.

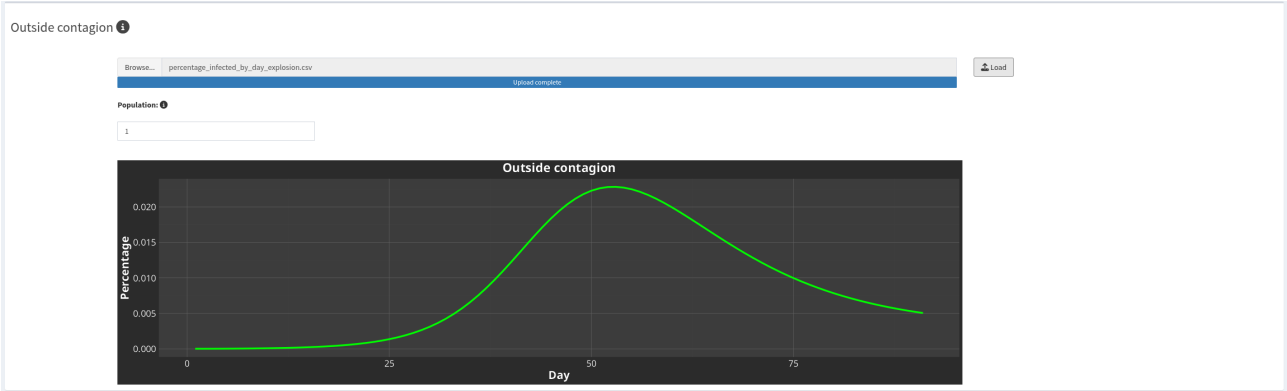

Figure 41: Example of a plot generated after uploading an outside contagion file.

At the beginning of the *Virus Parameters* section, all configured virus parameters are displayed in the *Saved Virus Parameters* panel, as shown in Figure 42. The first table shows the virus variant factor and virus severity, while the second one displays the initial infection set by the user. Entries can only be removed from the second table by clicking on them. An alert will appear, prompting the user to confirm the deletion or cancel the action. **Specific initial infections will take precedence over global ones, while randomly assigned infections are considered separately.**

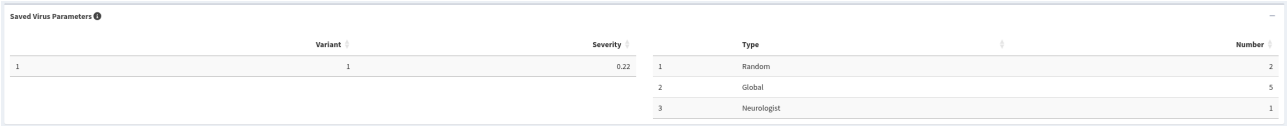

Figure 42: Saved countermeasures listed in the panel *Virus parameters*

4.8 Configuration

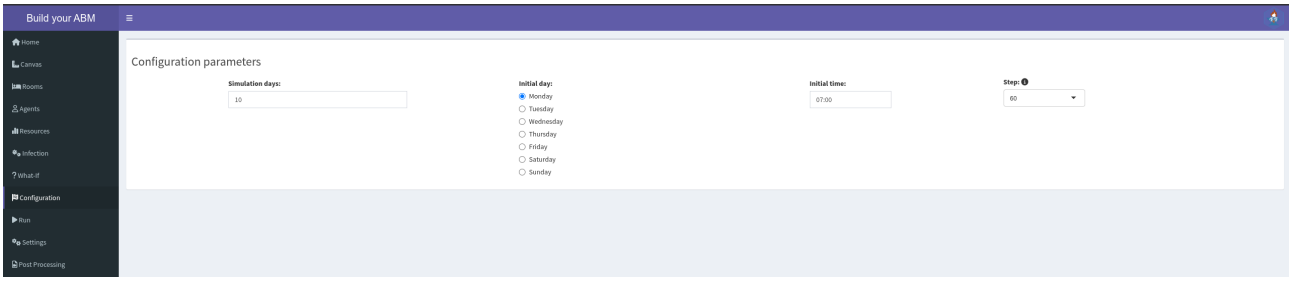

Figure 43: The **Configuration** page.

On this page—see Figure 43—, the user can configure various settings for the simulation:

- *Simulation days*: define the duration of the simulation in days.
- *Initial day*: set the start day of the week for the simulation.
- *Initial time*: determine the start time of the day for the simulation.
- *Step*: define the duration of a step in seconds; possible values are dividers of 60 seconds.

4.9 Run

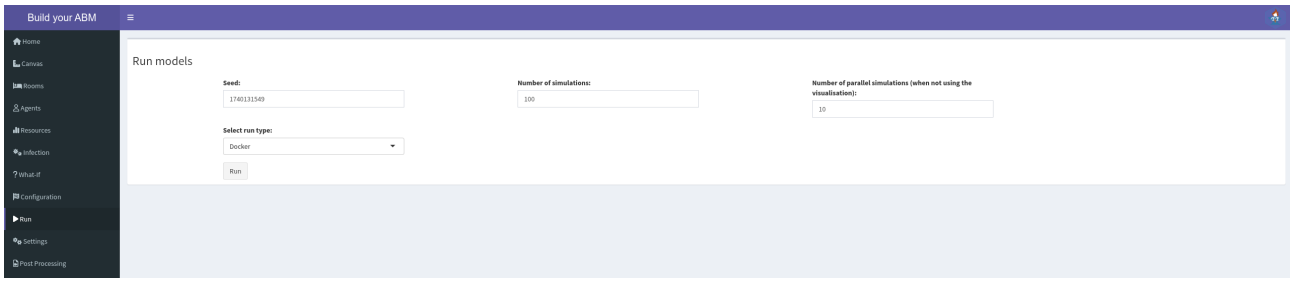

Figure 44: The **Run** page.

On this page—see Figure 44—, the user can configure various settings for the simulation:

- *Seed*: specify the starting seed for the simulation.
- *Number of simulations*: specify the number of simulations to run with a specific configuration of parameters.
- *Number of parallel simulations*: specify the number of simulations to run in parallel to speed up the computation.

Moreover, as shown in Figure 45, the user can choose how to run the simulation they have just created using FLAME GPU 2. There are three options:

- Launch the simulation using Docker (see Section 2.4); in this case, visualization will not be available.
- Launch the simulation locally with visualization.
- Launch the simulation locally without visualization.

These three options are available only when running *F4F* without Docker. If *F4F* is run using Docker Compose, the only available option will be *Docker*.

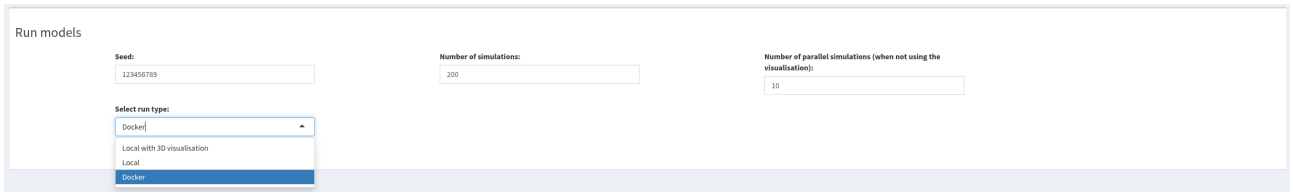

Figure 45: Section where the user can select how to run their simulation.

4.10 Settings

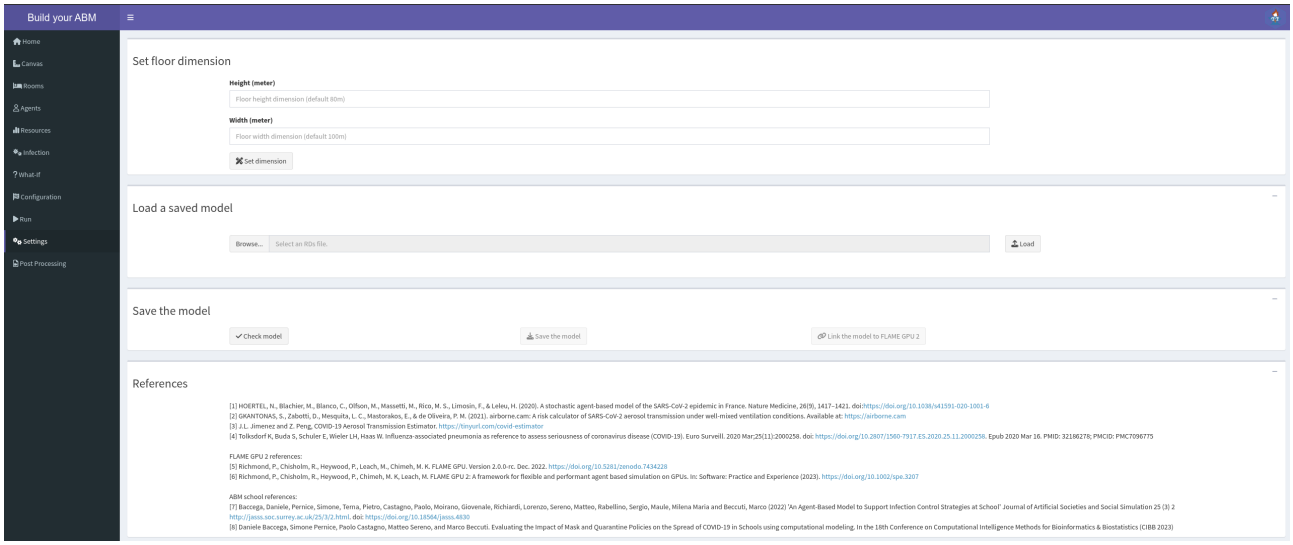

Figure 46: The **Settings** page.

Inside the **Settings** page—see Figure 46—the user can find three different panels:

- In the `Set floor dimension` panel, the user can adjust the dimensions of the canvas displayed on the Canvas page. The dimensions are specified in meters. The default dimensions are 80 meters in height and 100 meters in width. To set the desired dimensions, simply enter the values and press the *Set dimension* button.
- In the `Load a saved model` panel, the user can load a previously defined model. Click the *Browser* button to navigate through files and select the desired file. Once the correct file is chosen, click the *Load* button to import the model into the application.
- In the `Save the model` panel, the user can save the defined model. Use the *Check model* button to perform basic validation checks before saving. Once the checks are complete, the user can save the entire model by clicking the *Save the model* button. Additionally, the user can link the newly created environment in the *F4F* application to FLAME GPU 2 by using the *Link the model to FLAME GPU 2* button, giving it a name. In this way, the model will be saved in every directory named `FORGE4FLAME/inst/FLAMEGPU-FORGE4FLAME/resources/f4f` within the user's home directory. **To run a model in FLAME GPU 2, it must be saved in this location.** The user can create a subdirectory within this path, placing the JSON file and the required RDs downloaded using the *Save the model* button. This process is automated when using the *Link the model to FLAME GPU 2* button.
- In the `References` panel, the user will find the sources and references that were used as a basis for the development of the tool.

4.11 Post Processing

On this page—see Figure 47—users can analyse and visualise simulation results, filter specific simulations of interest, and observe virus evolution in each room, among other aspects. The section is organized into several panels, described below.

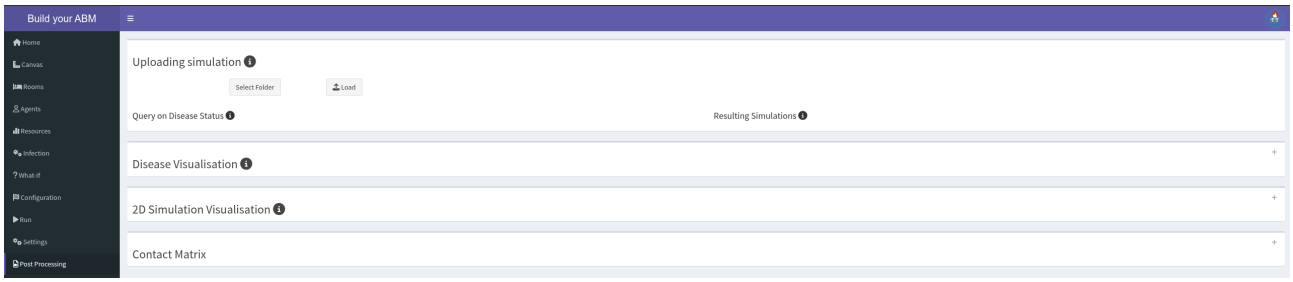

Figure 47: The **Post Processing** page.

4.11.1 Uploading simulation

Here, the user must upload the folder containing all the files and directories generated by the simulation—see Section 3 for more details on where results are saved. The required files, automatically generated by FLAME GPU 2, include `rooms_mapping.txt`, `seed.txt`, and all directories named `seed*`, each containing the following files: `evolution.csv`, `counters.csv`, `AEROSOL.csv`, `CONTACTS_MATRIX.csv`, `CONTACT.csv`, `INFO.csv`, and `AGENT_POSITION_AND_STATUS.csv`. For more details on these files, see Section 6.

Once the directory is chosen, a set of sliders will appear, as shown in Figure 48. These sliders allow users to select a specific subset of simulations—listed on the right side of the figure—by querying them. Users can refine their selection by limiting the simulation days and filtering based on the number of Susceptible, Exposed, Infected, Recovered, and Deceased agents. Based on these filters, the *Resulting Simulations* section will display directories containing simulations that meet the user's specified ranges. For instance, in Figure 48, only the simulations in which, between days 3 and 6, there were 20-30 susceptible agents, 0 exposed, 3-7 infected, 0-10 recovered, and 0 deaths were selected.

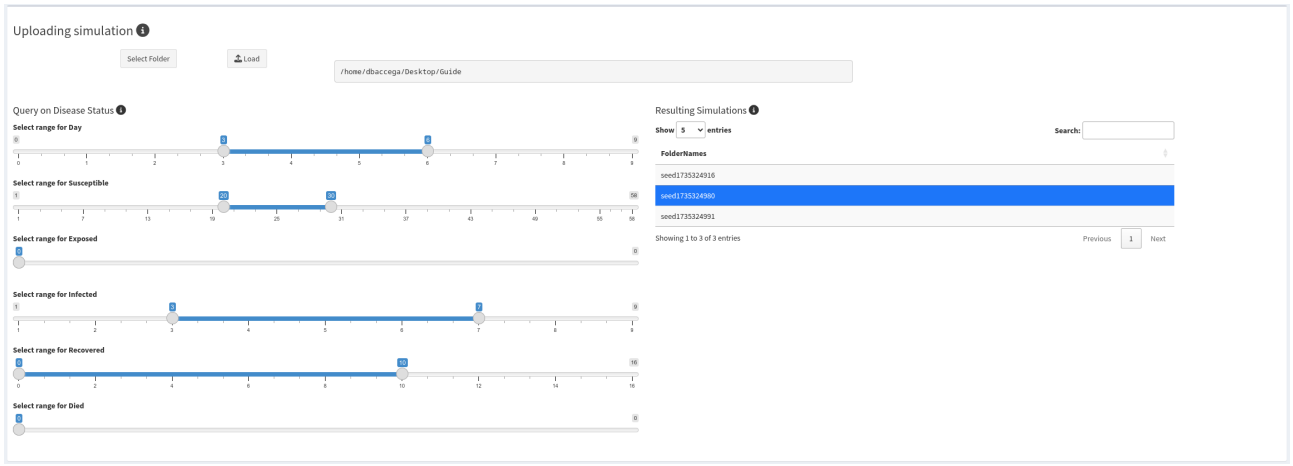

Figure 48: Example of query.

By clicking on each listed directory, the corresponding plots will appear below. As shown in Figure 49, these plots illustrate the daily evolution of the disease, along with additional data such as the number of agents birth or deaths (in the case of agents with a birth rate), the number of agents in quarantine, the number of agents infected outside the environment, and the number of swabs conducted. Additionally, the user can select a specific room and observe how contacts between susceptible and infected agents, as well as the virus concentration, have varied on an hourly basis—a contact between two agents is defined as the situation where they remain close to each other for a certain number of steps without ever separating. Finally, for

each of these plots, the user can display the mean curves along with the areas representing all simulations.

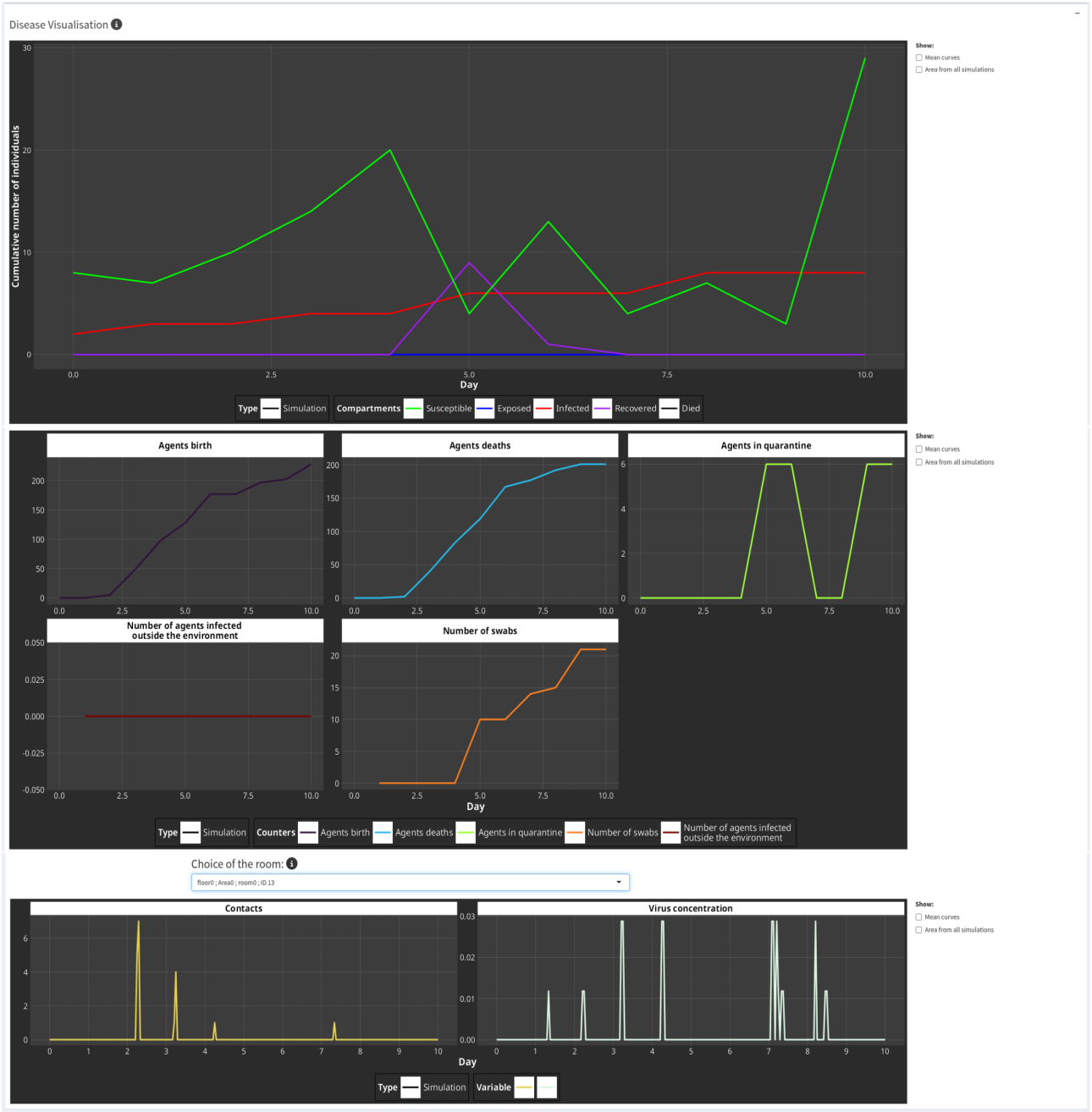

Figure 49: Plots generated after selecting a listed directory.

4.11.2 2D Simulation Visualisation

In this panel—see Figure 50—the user can navigate through two main sections:

- In the *Features* panel, the user can configure various display options:
  - Choose which floors to display (default: all floors).
  - Choose which agent types to display (default: all agent types).
  - Choose room colours based on name, type, area, cumulative number of contacts, current or cumulative virus concentration (default: name).

– Enable additional visual elements, such as room IDs, names, types or areas and agent IDs.

Figure 51 illustrates an example based on the model described in previous sections. Additionally, when an agent type is selected, the user can filter and display specific agents by their IDs—see Figure 52.

- In the *2D Visualisation* panel, the dynamics of the simulation are visualized in a 2D plot. The user can observe agent movements and changes in disease states using a time slider. In particular:

  – Different shapes represent different agent types.

  – Colours indicate disease states.

  – If a shape or colour is missing in the legend, it means no agent with those attributes is currently visible on the selected floor(s).

  The animation step can be adjusted via the input field next to the slider. The user can advance the simulation using the *Next* button, which increments the time step by the specified value and updates the 2D visualisation automatically, eliminating the need for manual adjustments. Finally, Figure 53 shows the cumulative virus concentration at the final time.

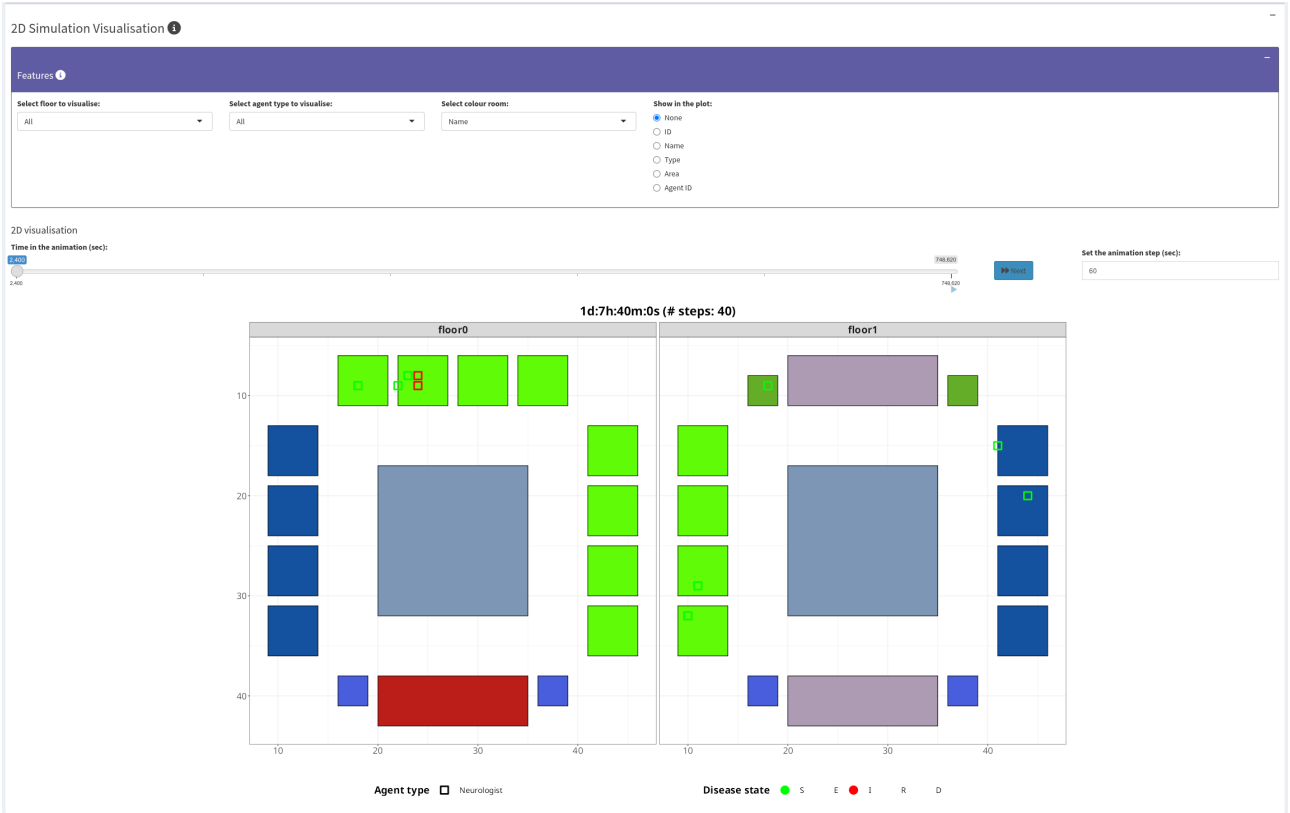

Figure 50: The 2D visualisation section.

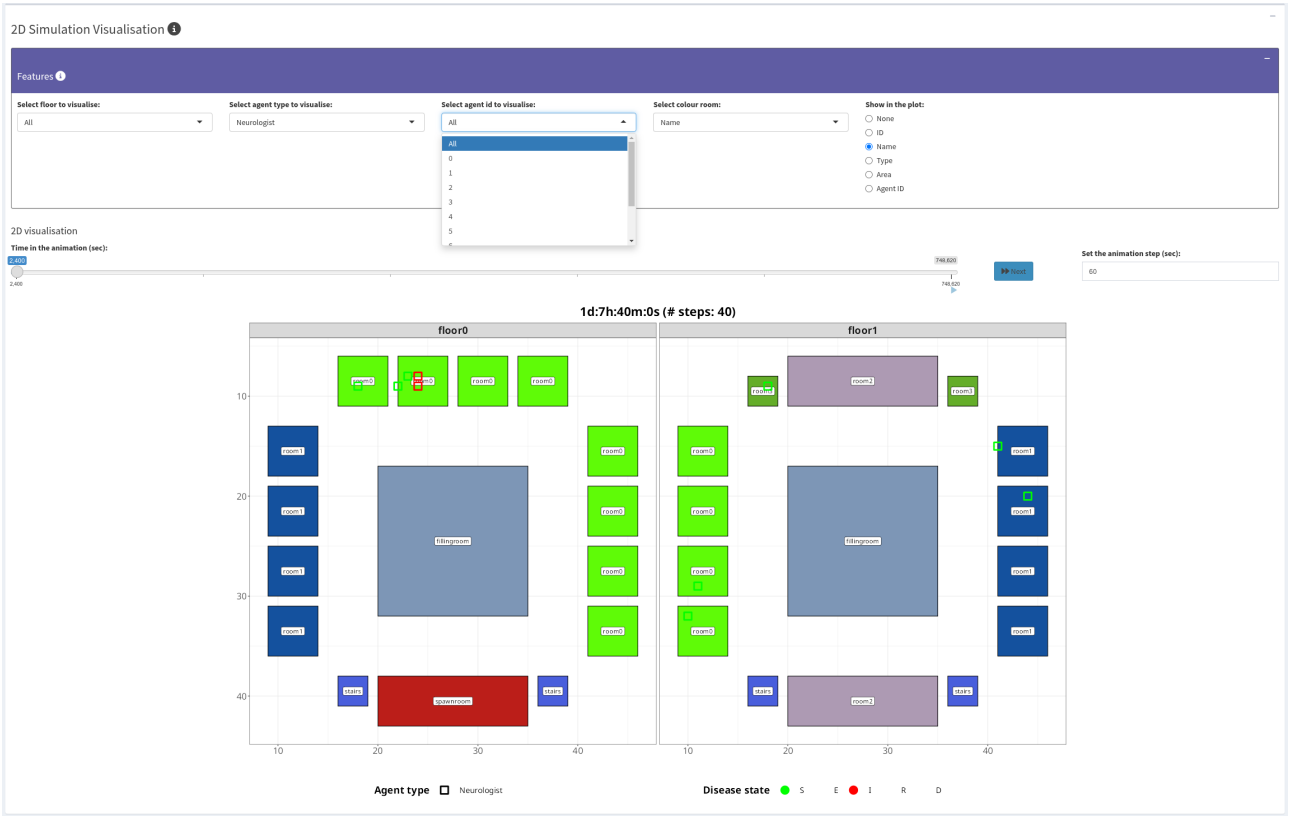

Figure 51: The 2D visualization section in which rooms' names and only the agent type *Neurologist* are displayed.

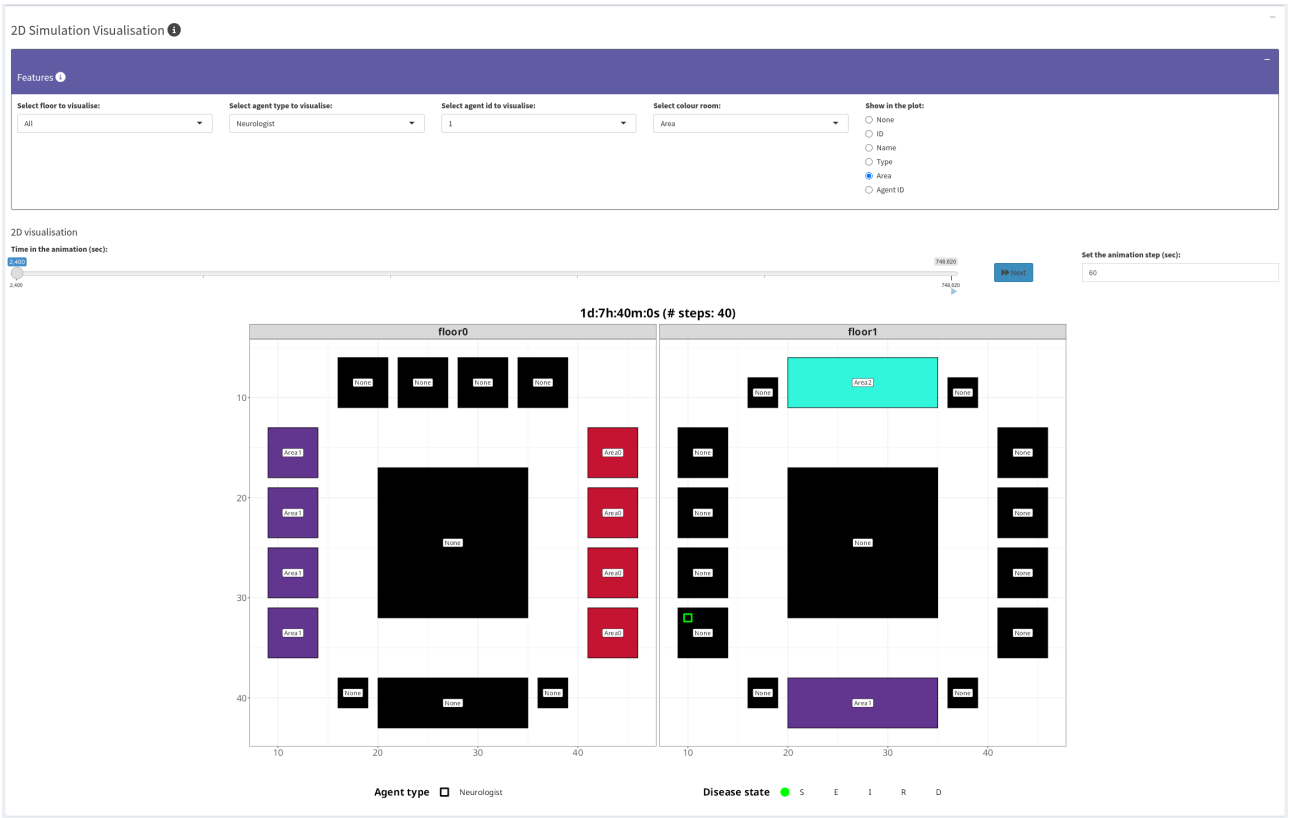

Figure 52: The 2D visualization section in which rooms' areas and only the agent type *Neurologist* with id 1 are displayed.

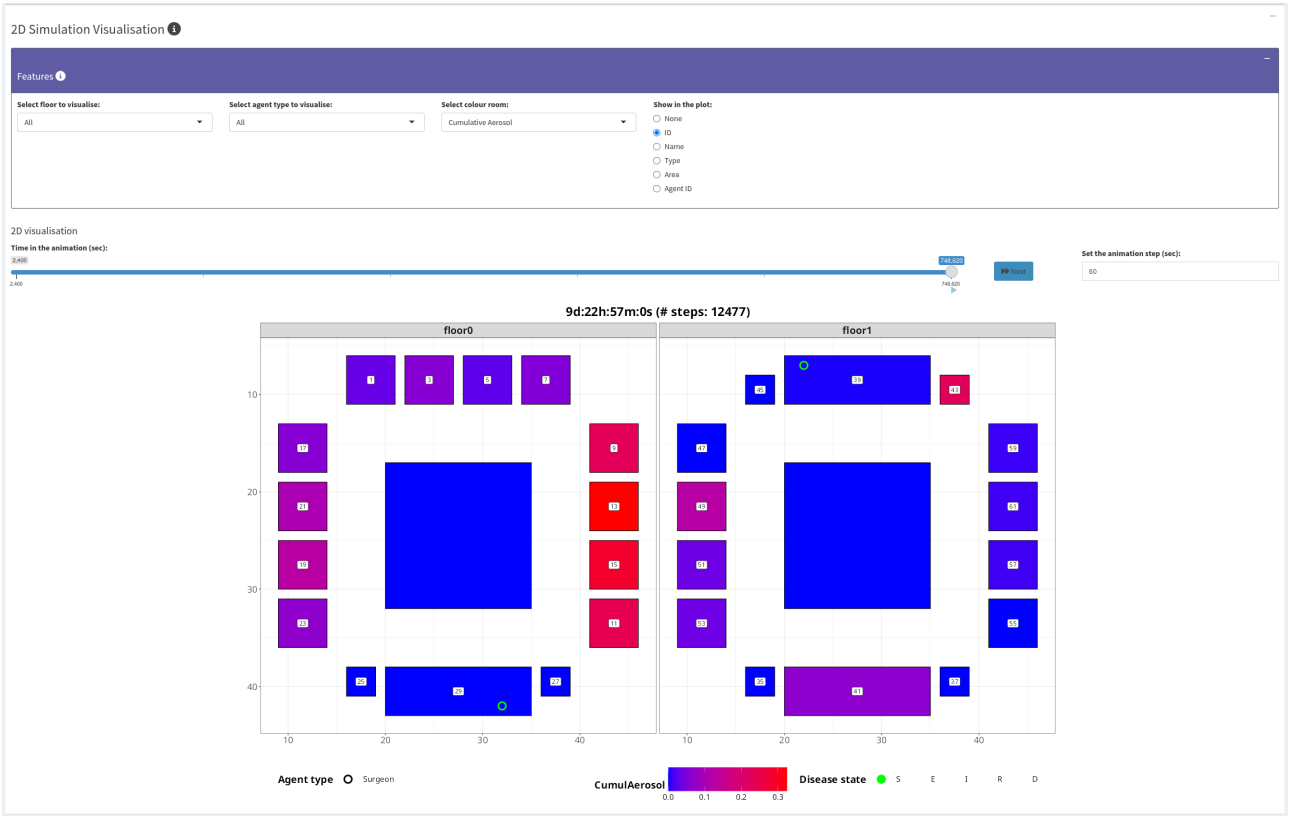

Figure 53: The 2D visualization section in which the cumulative virus concentration at the final time is shown together with rooms' IDs.

4.11.3 Contact Matrix

In this panel—see Figure 54—the lower triangular contact matrix shows the hourly average number of contacts among the different agent types.

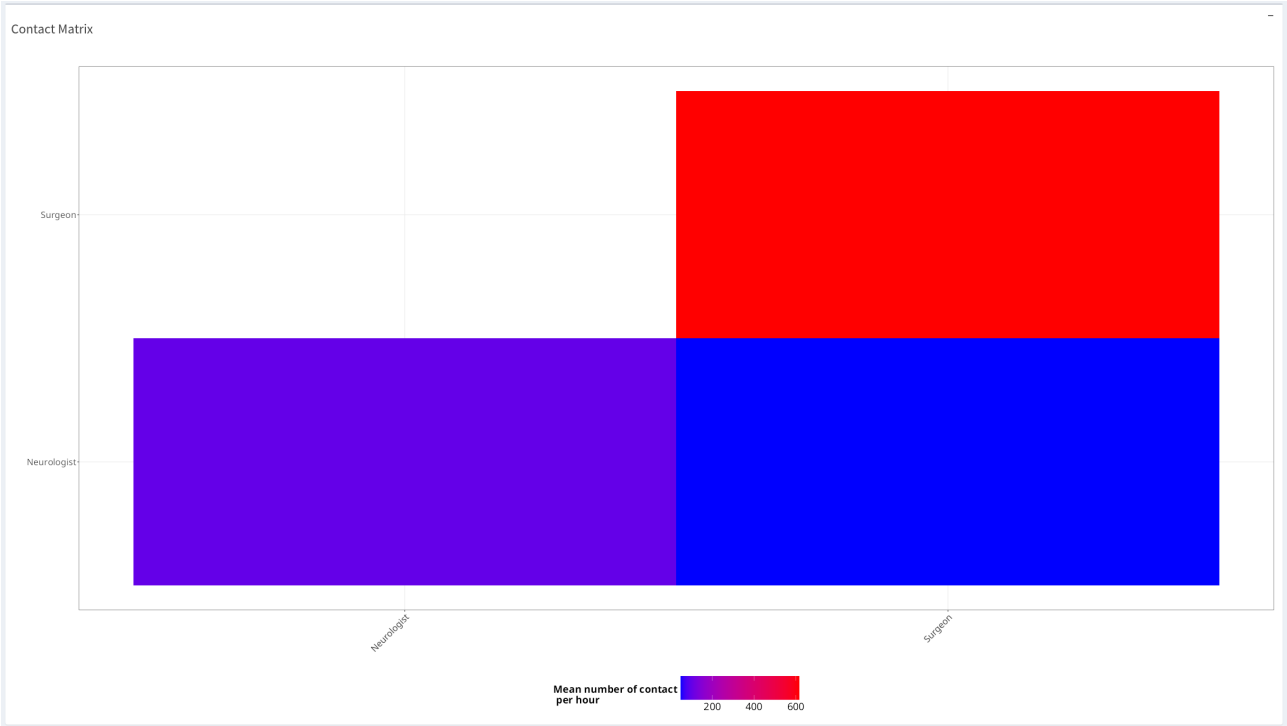

Figure 54: The *Contact Matrix* panel.

5 School

In this section, we provide further details about the generic middle school environment described in the main document. Figures 55, 56, and 57 illustrate the three floors of the school defined using *F4F*. Specifically, *floor0* includes the spawn-room where agents enter the environment, along with four classrooms, two bathrooms—one for students and one for teachers—the gym, the teacher's room, the principal's office, the temperature measurement room, the stairs connecting the floors, a couple of filling rooms, and four graph points in the hallway. On *floor1* and *floor2*, there are four classrooms, a student bathroom, stairs, some filling rooms, and two graph points in the hallway.

Figures 58, 59, and 60 illustrate the definition of the agent *student_1A*. Figure 58 presents one of the two determined flows associated with the agent, where the agent spends 300 minutes in classroom 1A, with two 15-minute breaks—one between the second and third classes and another between the fourth and fifth classes. Figure 59 highlights the random events—two random events are possible: going to the bathroom or visiting the principal's office. Finally, Figure 60 depicts one of the two entry flows linked to the first determined flow, which occurs every day except on Thursday, Saturday, and Sunday.

Figures 61, 62, and 63 illustrate the definition of the agent *teacher_0*. Figure 61 presents one of the three determined flows associated with the agent, where the agent will spend 115 minutes in classroom 1A. Figure 62 highlights the random events—only one possible random event includes going to the bathroom. Finally, Figure 63 depicts one of the three entry flows linked to the first determined flow, which occurs on Monday and Tuesday.

Figure 64 illustrates the infection parameters of the SEIRS compartmental model used in the alert experiments in the main document. Specifically, we set the average incubation and infection periods to 5 days and the average immunization period to one year and a half.

Figures 65, 66, 67, and 68 present screenshots from the *Post Processing* page considering the scenario *No Countermeasures* introduced in the main paper. The selection criteria for

these simulations required that, between days 20 and 40, the number of agents fell within the following ranges: 100–200 susceptible, 30–60 exposed, 20–40 infected, 0–150 recovered, and 0 deaths. Figure 66 illustrates the trajectories of one of the 56 simulations that meet the query conditions shown in Figure 65. In the 2D visualisation of the school—Figure 67—, the snapshot captures the day 25$^{\text{th}}$ of the simulation, depicting students in classrooms, the principal in his office, and a few teachers in the teacher's room. Additionally, the colour of each room represents the hourly virus concentration. Figure 68 shows the contact matrix among the different types of agents. Finally, we do not include screenshots of the *What-If*, *Configuration*, and *Settings* pages, as users have the flexibility to customize parameters as needed. Furthermore, no restrictions were imposed on room access based on resource availability, as each room was globally assigned 1,000 resources.

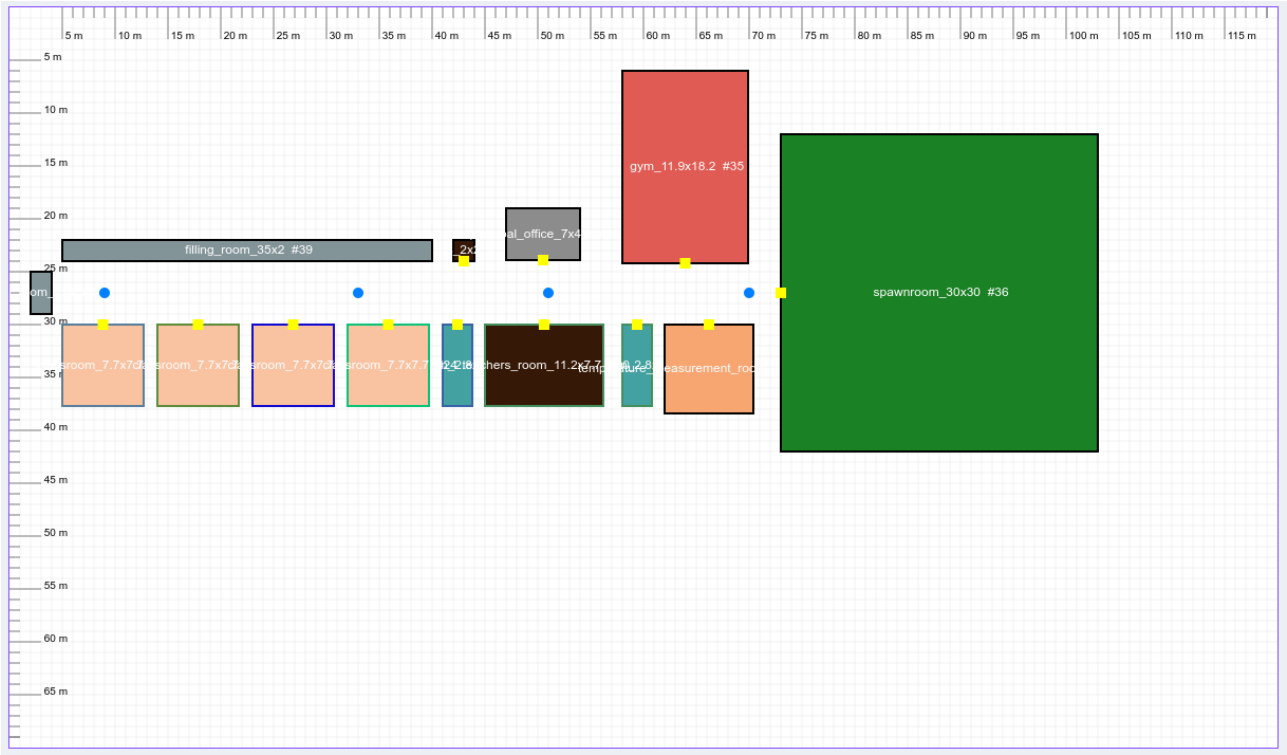

Figure 55: *floor0* of the school environment visualized in the canvas of *F4F*.

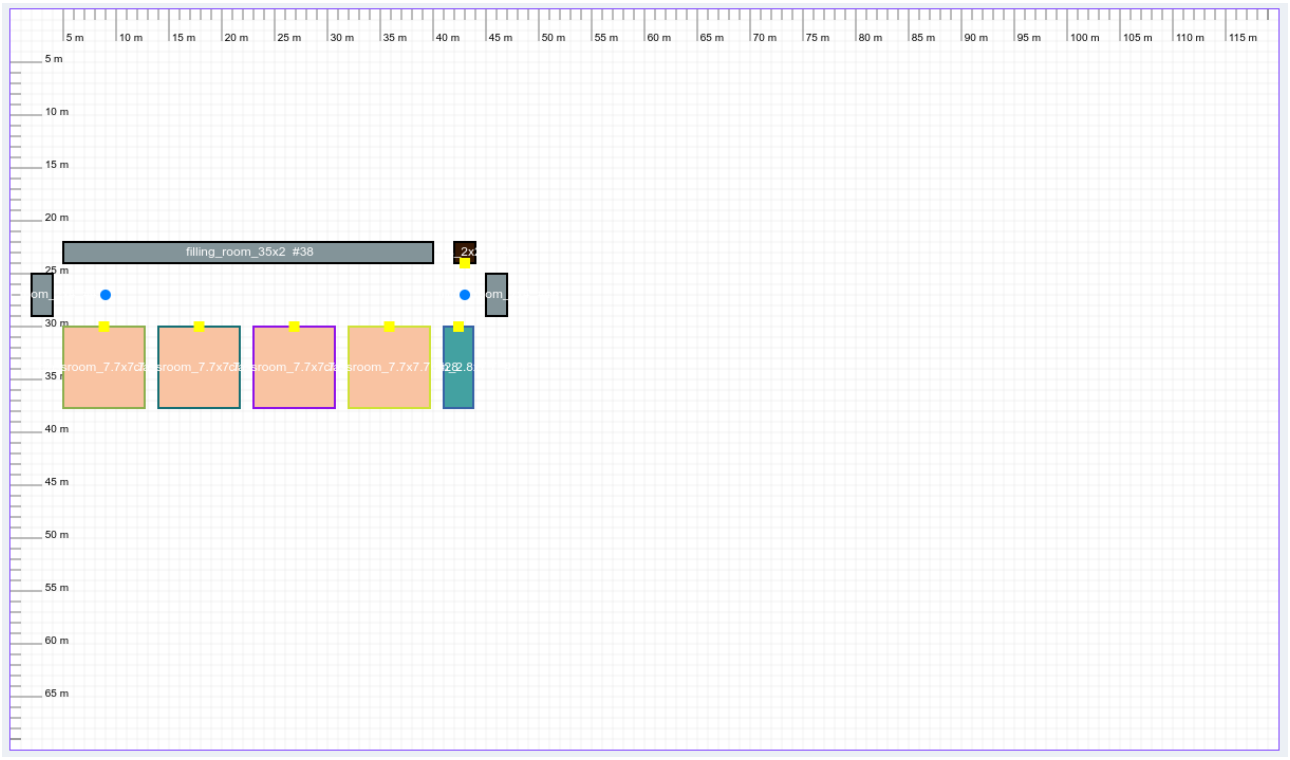

Figure 56: *floor1* of the school environment visualized in the canvas of *F4F*.

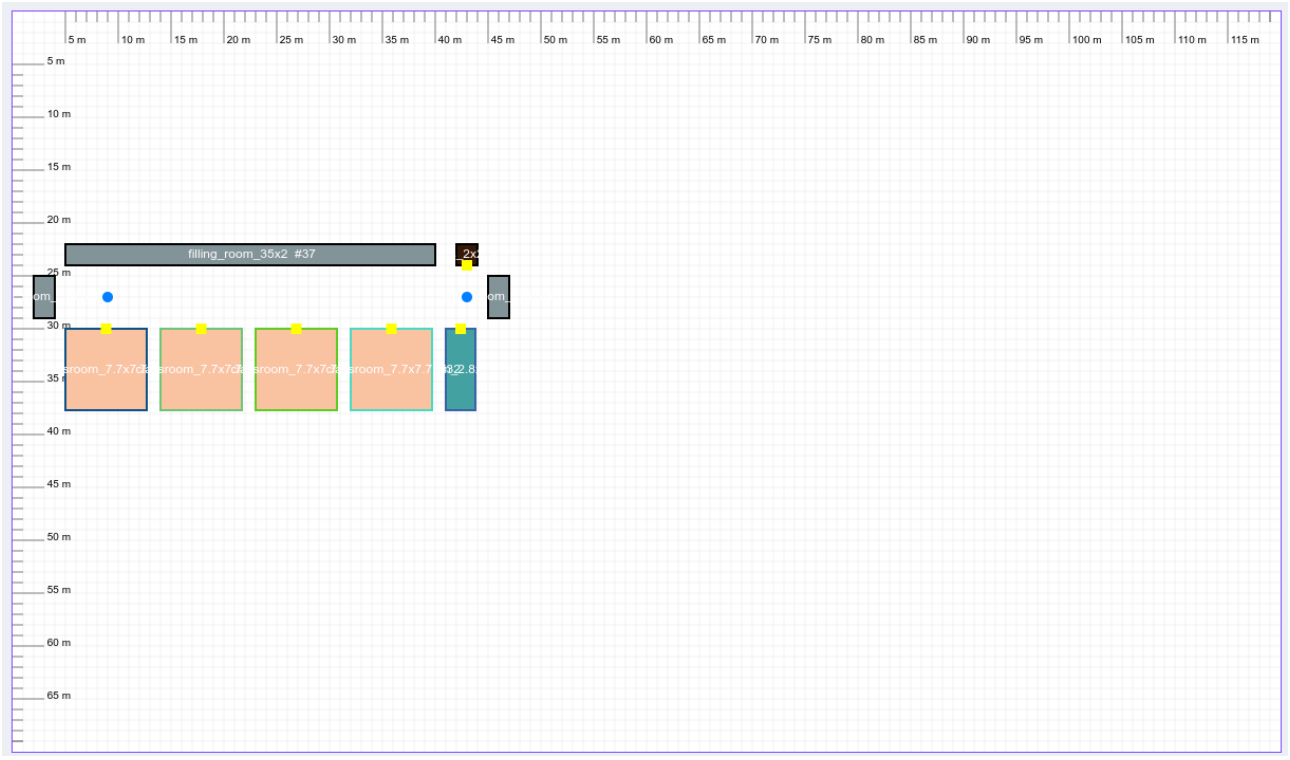

Figure 57: *floor2* of the school environment visualized in the canvas of *F4F*.

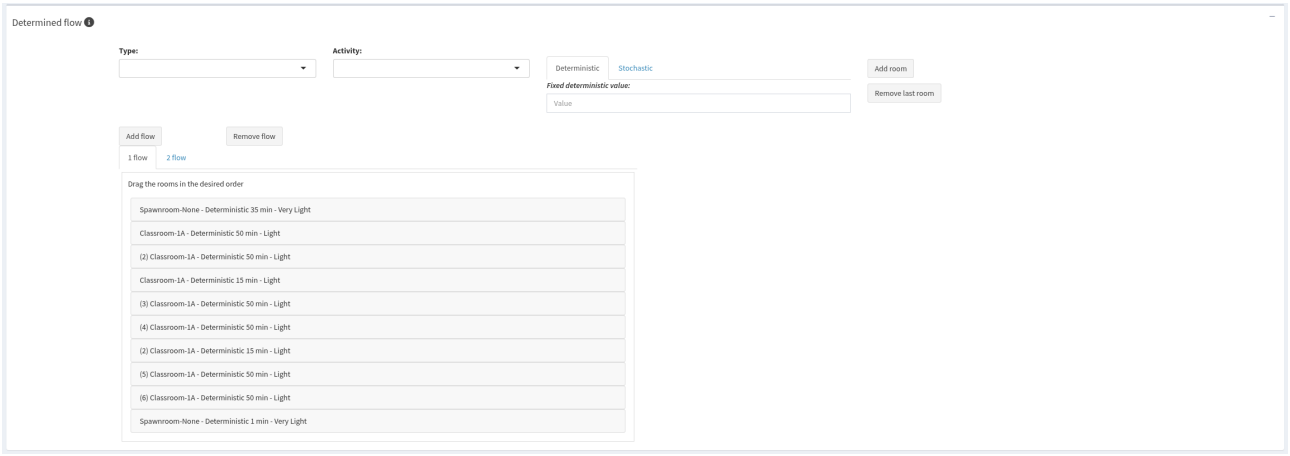

Figure 58: Determined flow of agent *student_1A*.

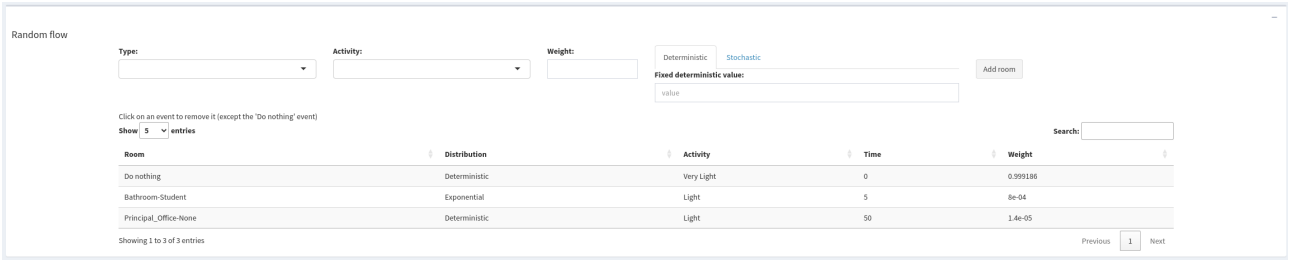

Figure 59: Random flow of agent *student_1A*.

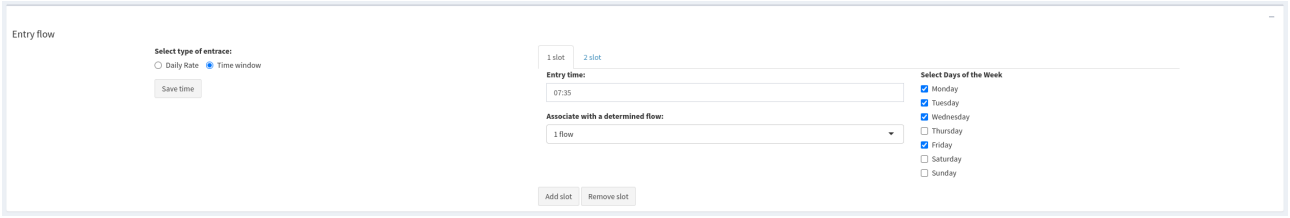

Figure 60: Entry flow of agent *student_1A*.

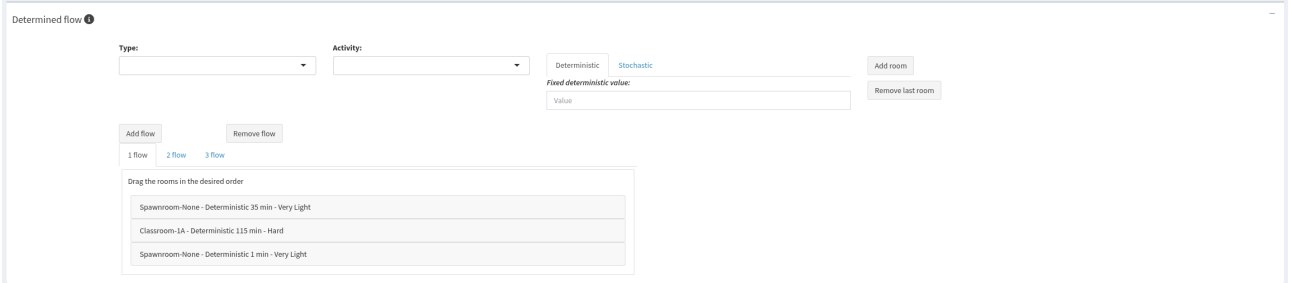

Figure 61: Determined flow of agent *teacher_0*.

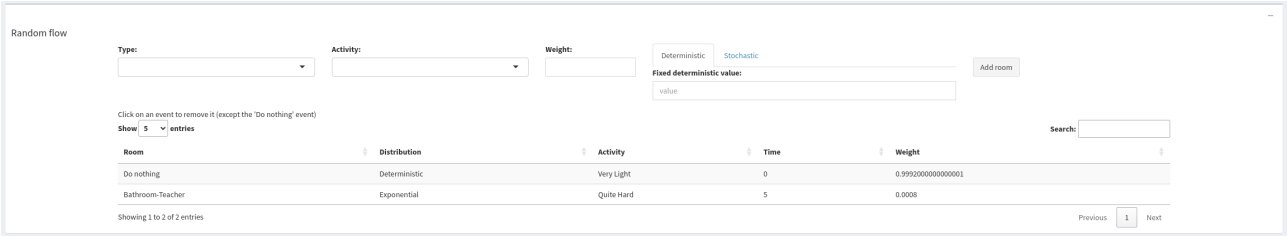

Figure 62: Random flow of agent *teacher_0*.

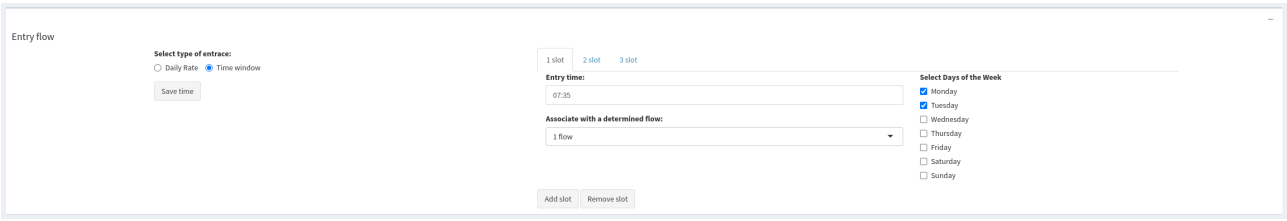

Figure 63: Entry flow of agent *teacher_0*.

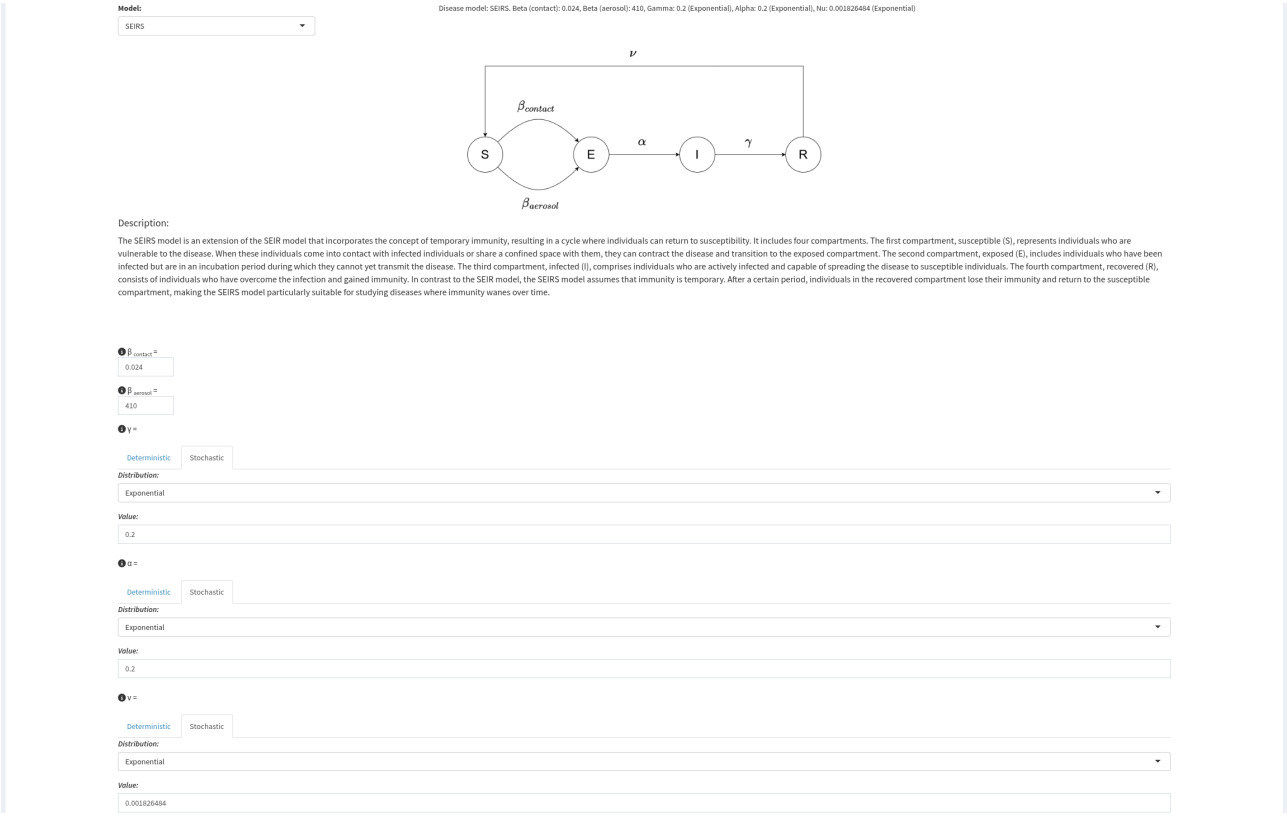

Figure 64: Infection parameters used in the alert experiments.

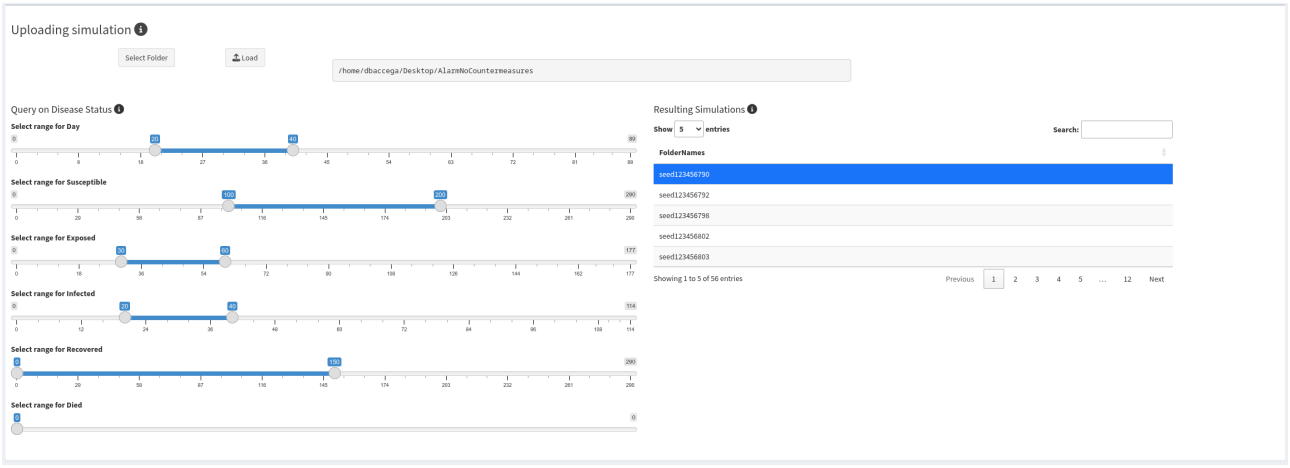

Figure 65: Example of query on the results obtained using the school model on the *No Countermeasures* scenario.

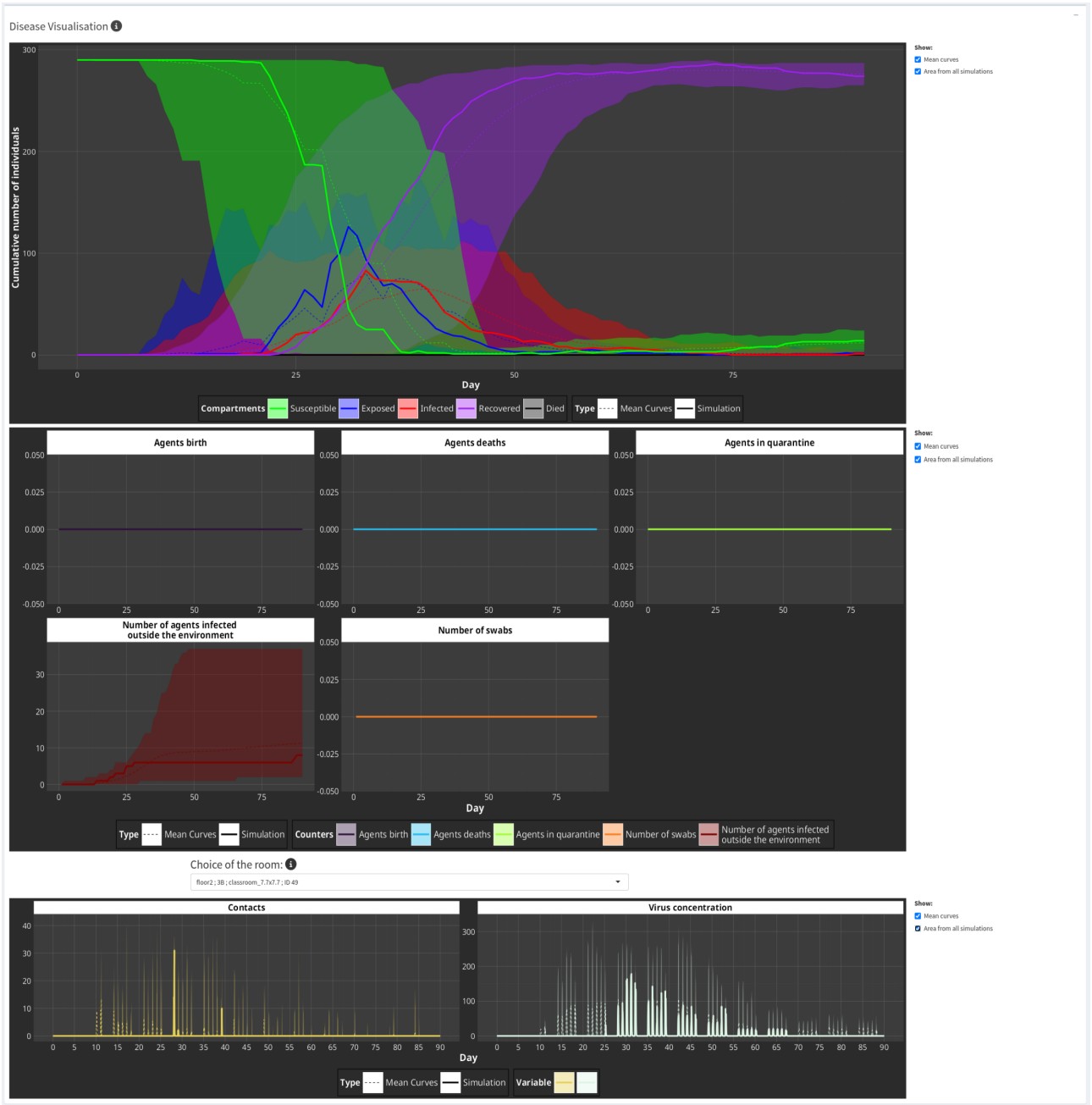

Figure 66: Plots generated after selecting a listed directory using the school model on the *No Countermeasures* scenario (with means and areas).

Figure 67: 2D visualization of the school where the colour of each room represents the hourly virus concentration on the *No Countermeasures* scenario.

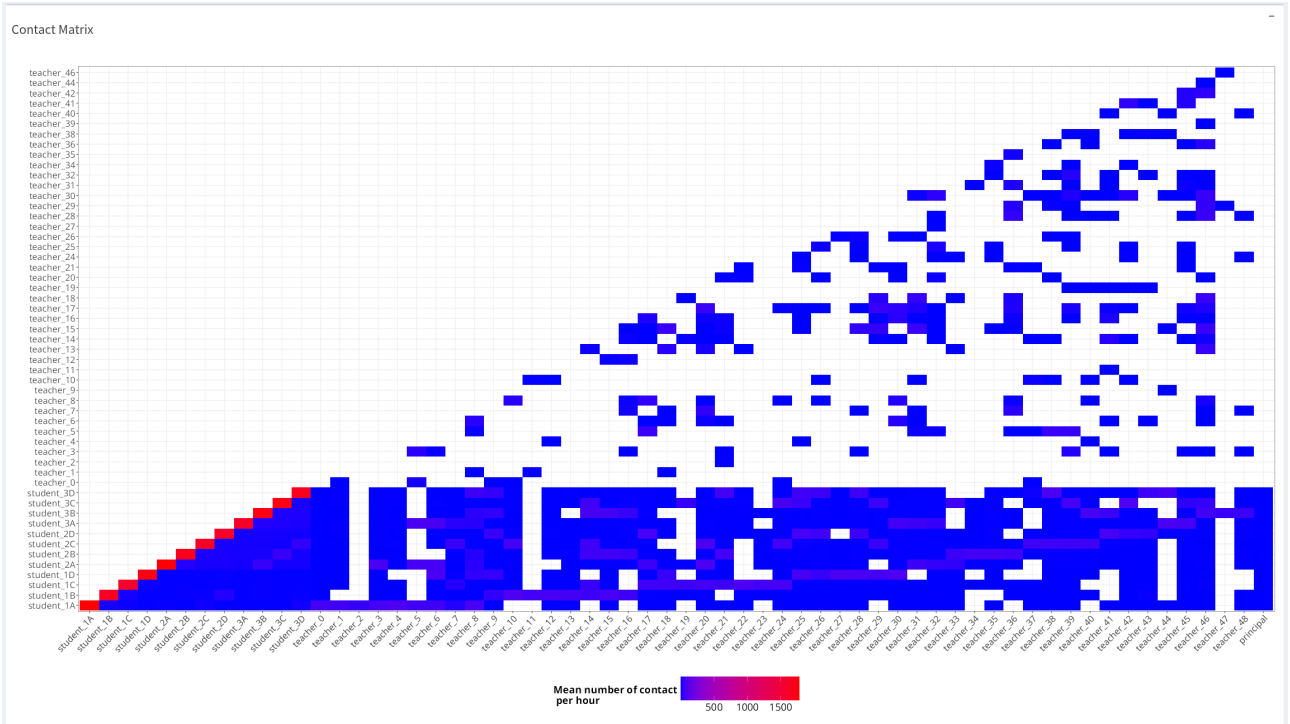

Figure 68: Contact matrix of the school model on the *No Countermeasures* scenario.

6 Link between F4F and FLAME GPU 2

The output of *F4F*—a JSON file including the whole model—is linked to FLAME GPU 2 through a Python script which automatically generates the necessary code to run the simulation.

In particular, the information required to simulate the model is stored in FLAME GPU 2 using environmental properties and macro-properties, agents' variables, and various C++ *defines*.

One of the key components of FLAME GPU 2 simulations is the movement graph, briefly introduced in Section 4. This graph is automatically generated using room centres, room doors, user-defined graph points, and stairs—the latter connects floors to each other. It is stored in FLAME GPU 2 environment and macro-environment variables to handle agent movements. Using the A* algorithm, the graph enables agents to determine the shortest path between a source and a destination whenever they need to navigate the modelled environment. For example, Figure 69 illustrates all possible edges of the graph for *floor0* in the model defined in Section 4.

When running the FLAME GPU 2 simulation with the visualization enabled—without Docker, see Section 3 for more details—, the environment created using *F4F*—as defined in Section 4.2—will be displayed, as shown in Figure 70. Moreover, in Figure 71 a 3D visualization of the school environment is shown.

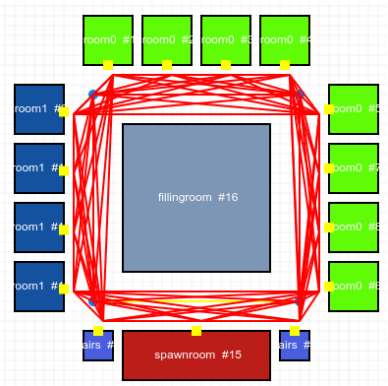

Figure 69: All possible edges of the graph for *floor0* in the model defined in Section 4

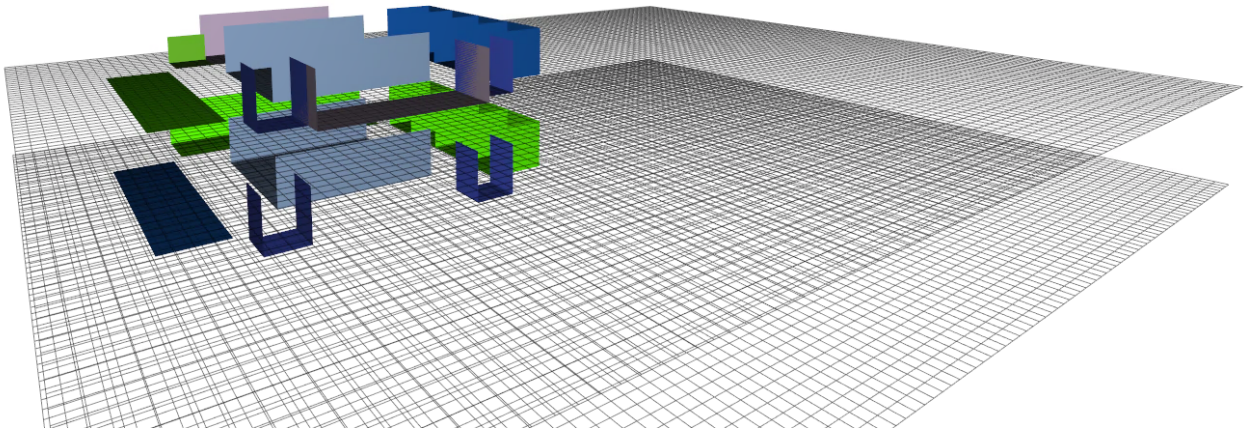

Figure 70: 3D visualization in FLAME GPU 2 of the example depicted in Section 4.

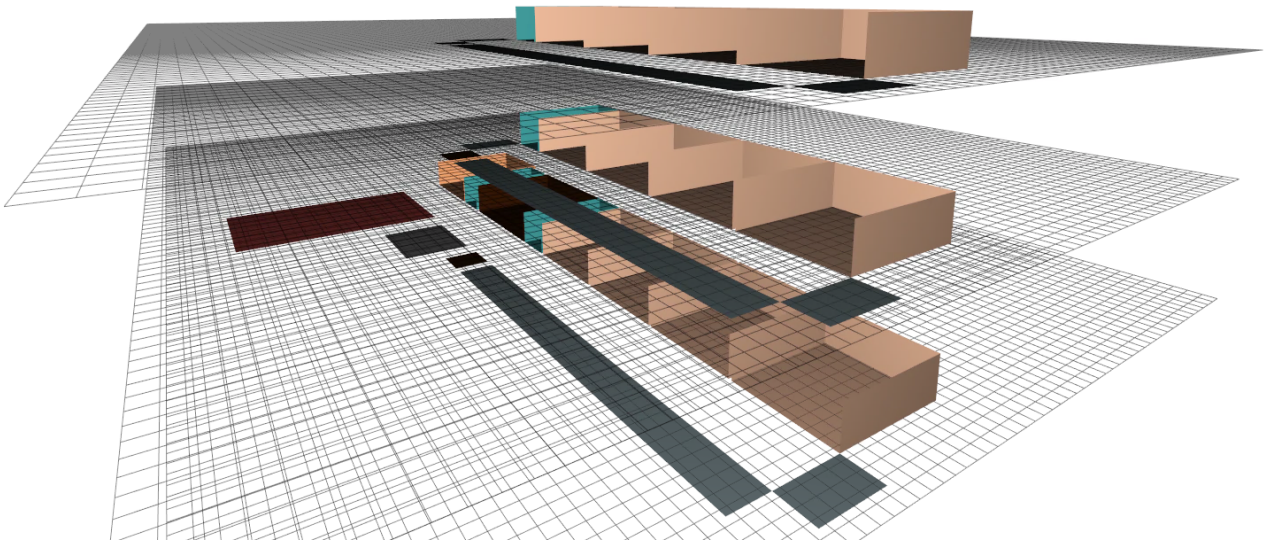

Figure 71: 3D visualization in FLAME GPU 2 of the school depicted in Section 5 and used in the main document.

In the main document and Section 3, we outline the ways users can run *F4F* and FLAME GPU 2 explaining how users can effectively run simulations. When FLAME GPU 2 completes the simulation runs, it generates CSV files containing various information required to post-analyse the simulated scenario (e.g., disease evolution, contact matrix, virus diffusion within the environment, and agents' movements). This information is stored in separate files—one for disease evolution, one for the contact matrix, and so on. These files can be passed to *F4F* for post-analysis and visualization of disease dynamics in the modelled system—see Section 4.11.

Specifically, each folder containing the model's results includes:

- `rooms_mapping.txt` stores the mapping between each room's ID in the graph and the corresponding position of its centre;
- `seed.txt` stores the simulation seed specified using *F4F*;
- various directories named `seed*`—there are as many directories as there are simulation runs—, each containing the following files:
  - `evolution.csv` stores the daily evolution of the disease; the file contains six columns: *Day*, *Susceptible*, *Exposed*, *Infected*, *Recovered*, and *Died*;
  - `counters.csv` stores the daily evolution of some interesting pre-defined counters—already introduced in Section 4.11; the file contains five columns: *Day*, *COUNTERS_CREATED_AGENTS_WITH_RATE*, *COUNTERS_KILLED_AGENTS_WITH_RATE*, *AGENTS_IN_QUARANTINE*, *SWABS*, and *NUM_INFECTED_OUTSIDE*;
  - `AEROSOL.csv` stores the hourly evolution of virus concentration in each room—entries are only logged when the virus concentration is greater than zero; the file contains three columns: the first represents the FLAME GPU 2 step—i.e., simulation time—, the second indicates the virus concentration, and the third corresponds to the room's ID in the graph;
  - `CONTACTS_MATRIX.csv` stores the hourly lower triangular contact matrix—entries are logged only if the number of contacts in the last hour is greater than zero; the

file contains four columns: the first represents the FLAME GPU 2 step, the second and third indicate the agent types, and the fourth records the number of contacts in the last hour between those agent types;

- `CONTACT.csv` stores interactions between susceptible and infected agents; it contains four columns: the first indicates the FLAME GPU 2 step, the second and third specify the types of the susceptible and infected agents, respectively, and the fourth provides the ID of the room where the contact occurred; if agents remain in the contact area from step t_1 to t_2—where $t_1 < t_2$—, the user will find t_2 - t_1 rows in the file, corresponding to the time steps t_1, $t_1 + 1$, ..., t_2;
- `AGENT_POSITION_AND_STATUS.csv` stores the position of each agent whenever it changes from the previous FLAME GPU 2 step; the file contains seven columns: the first indicates the FLAME GPU 2 step, the second represents the agent's ID, the third specifies the agent type, the fourth, fifth, and sixth provide the agent's position on the x, y, and z axes, respectively, and the seventh denotes the agent's disease state.
- `INFO.csv` contains information about the simulation. Its columns are formatted as follows: the first column indicates the FLAME GPU 2 step, and the second column is a string.
- `host_rng_state.txt` stores the state of the host's random number generator at the end of the simulation.

References

[1] Marco Aldinucci et al. "HPC4AI: an AI-on-demand federated platform endeavour". In: *Proceedings of the 15th ACM International Conference on Computing Frontiers*. 2018, pp. 279–286.

[2] D Attali. "Colourpicker: A colour picker tool for shiny and for selecting colours in plots". In: *R package version* 1 (2017). URL: `https://cran.r-project.org/web/packages/colourpicker/`.

[3] D Attali, T Edwards, and Z Wang. "shinyalert: Easily create pretty popup messages (modals) in 'shiny'". In: *R package version* 2.0 (2018). URL: `https://cran.r-project.org/web/packages/shinyalert/`.

[4] Dean Attali. "Shinyjs: Easily improve the user experience of your shiny apps in seconds". In: *R package version* 2.0 (2020). URL: `https://cran.r-project.org/web/packages/shinyjs/`.

[5] Daniele Baccega et al. "An agent-based model to support infection control strategies at school". In: *JASSS* 25.3 (2022), pp. 1–15. DOI: `https://doi.org/10.18564/jasss.4830`.

[6] Daniele Baccega et al. "Living along COVID-19: assessing contention policies through Agent-Based Models". In: (2024). The article is accepted in the proceedings of the CIBB 2023 conference Springer Lecture Notes in Computer Science (LNCS).

[7] Eric Bailey. "shinyBS: Twitter bootstrap components for shiny". In: *R package version 0.61* (2015). URL: `https://cran.r-project.org/web/packages/shinyBS/`.

[8] Giorgio Buonanno, Luca Stabile, and Lidia Morawska. "Estimation of airborne viral emission: Quanta emission rate of SARS-CoV-2 for infection risk assessment". In: *Environment International* 141 (2020), p. 105794. DOI: `https://doi.org/10.1016/j.envint.2020.105794`.

[9] Winston Chang et al. "Package 'shiny'". In: *See http://citeseerx. ist. psu. edu/viewdoc/-download* (2015). URL: https://cran.r-project.org/web/packages/shiny/.

[10] Winston Chang et al. "Package 'shinydashboard'". In: (2022). URL: https://cran.r-project.org/web/packages/shinydashboard/.

[11] Winston Chang et al. *shiny: Web Application Framework for R*. R package version 1.10.0, https://github.com/rstudio/shiny. 2024. URL: https://shiny.posit.co/.

[12] Winston Chang et al. "shinythemes: Themes for Shiny". In: *R package version* 1.2 (2018). URL: https://cran.r-project.org/web/packages/shinythemes/.

[13] Gábor Csárdi, Kuba Podgórski, and Rich Geldreich. *zip: Cross-Platform "zip" Compression*. 2020. URL: https://cran.r-project.org/web/packages/zip/.

[14] Gábor Csárdi et al. *remotes: R Package Installation from Remote Repositories, Including 'GitHub'*. R package version 2.5.0, https://github.com/r-lib/remotes#readme. 2024. URL: https://remotes.r-lib.org.

[15] Savvas Gkantonas et al. "airborne. cam: A risk calculator of SARS-CoV-2 aerosol transmission under well-mixed ventilation conditions". In: (2021). DOI: https://doi.org/10.17863/CAM.72192.

[16] Jim Hester and Jennifer Bryan. "Glue: Interpreted string literals". In: *R package version* 1.2 (2020). URL: https://cran.r-project.org/web/packages/glue/.

[17] Nicolas Hoertel et al. "A stochastic agent-based model of the SARS-CoV-2 epidemic in France". In: *Nature medicine* 26.9 (2020), pp. 1417–1421. DOI: https://doi.org/10.1038/s41591-020-1001-6.

[18] Fanny Meyer and Victor Perrier. *shinybusy: Busy Indicator for "Shiny"Applications*. 2020. URL: https://cran.r-project.org/web/packages/shinybusy/.

[19] Jeroen Ooms, Duncan T Lang, and Lloyd Hilaiel. "jsonlite: A robust, high performance JSON parser and generator for R". In: *R package version* 1 (2018). DOI: https://cran.r-project.org/web/packages/jsonlite/.

[20] David Ornelles. *EBImage Extra: Helper functions for EBImage*. Version 0.0.0.2. License: GPL-3. 2019. DOI: https://orcid.org/0000-0002-5151-8382. URL: https://github.com/ornelles/EBImageExtra.

[21] Grégoire Pau et al. "EBImage—an R package for image processing with applications to cellular phenotypes". In: *Bioinformatics* 26.7 (2010), pp. 979–981. DOI: https://doi.org/10.1093/bioinformatics/btq046.

[22] Thomas Lin Pedersen et al. "shinyFiles: a server-side file system viewer for Shiny". In: *R package version 0.6* 2 (2016).

[23] Zhe Peng et al. "Practical indicators for risk of airborne transmission in shared indoor environments and their application to COVID-19 outbreaks". In: *Environmental science & technology* 56.2 (2022), pp. 1125–1137. DOI: https://doi.org/10.1021/acs.est.1c06531.

[24] Victor Perrier, Fanny Meyer, David Granjon, et al. "shinyWidgets: Custom inputs widgets for Shiny". In: *R package version 0.4* 7 (2019). URL: https://cran.r-project.org/web/packages/shinyWidgets/.

[25] RStudio and Inc. "htmltools: Tools for HTML". In: *R package version* 3 (2017), p. 6.

[26] Kristin Tolksdorf et al. "Influenza-associated pneumonia as reference to assess seriousness of coronavirus disease (COVID-19)". In: *Eurosurveillance* 25.11 (2020), p. 2000258. DOI: https://doi.org/10.2807/1560-7917.es.2020.25.11.2000258.

[27] A de Vries, B Schloerke, and K Russell. *sortable: Drag-and-Drop in 'shiny'Apps with 'SortableJS'. R package version 0.5. 0*. 2024. URL: https://cran.r-project.org/web/packages/sortable/.

[28] Toru Watanabe et al. "Development of a dose-response model for SARS coronavirus". In: *Risk Analysis: An International Journal* 30.7 (2010), pp. 1129–1138. DOI: https://doi.org/10.1111/j.1539-6924.2010.01427.x.

[29] Hadley Wickham. *ggplot2: Elegant Graphics for Data Analysis*. Springer-Verlag New York, 2016. ISBN: 978-3-319-24277-4. URL: https://ggplot2.tidyverse.org.

[30] Hadley Wickham. "Package 'tidyr'". In: *Easily Tidy Data with'spread'and'gather ()'Functions* (2017). URL: https://cran.r-project.org/web/packages/tidyr/.

[31] Hadley Wickham. "stringr: modern, consistent string processing". In: *The R Journal* 2.2 (2010), pp. 38–40. DOI: 10.32614/RJ-2010-012. URL: https://doi.org/10.32614/RJ-2010-012.

[32] Hadley Wickham et al. *devtools: Tools to Make Developing R Packages Easier*. https://devtools.r-lib.org/, https://github.com/r-lib/devtools. 2022.

[33] Hadley Wickham et al. *dplyr: A Grammar of Data Manipulation*. R package version 1.1.4. 2023. URL: https://CRAN.R-project.org/package=dplyr.

[34] Hadley Wickham et al. "Package 'readr'". In: *Read Rectangular Text Data. Available online: https://cran. r-project. org/web/packages/readr/readr. pdf (accessed on 23 August 2023)* (2024). URL: https://cran.r-project.org/web/packages/readr/.

[35] Yihui Xie, Joe Cheng, Xianying Tan, et al. "DT: a wrapper of the JavaScript library 'DataTables'". In: *R package version 0.4* (2018). URL: https://cran.r-project.org/web/packages/DT/.