**Exercise sheet 2**
**Using and analyzing simulation software**

Due date: 2021-05-12
Tasks: 5

The Vadere software is a tool for the simulation and visualization of human crowds. In this exercise, you will learn how to use its graphical user interface, create, run, and modify simulation scenarios, and integrate a new SIR model into the software. This will be useful for working with Vadere, but also if you need to work with other simulation tools, or even implement your own. The software Vadere can be downloaded here: `http://www.vadere.org/releases/`. You should use the **"master branch"** (not the stable branch, the SIR code will not work with it). The source code can be downloaded from the LRZ gitlab project here: `https://gitlab.lrz.de/vadere/vadere`. To get you started, watch the video tutorials on the website and look at the slides from the lecture.

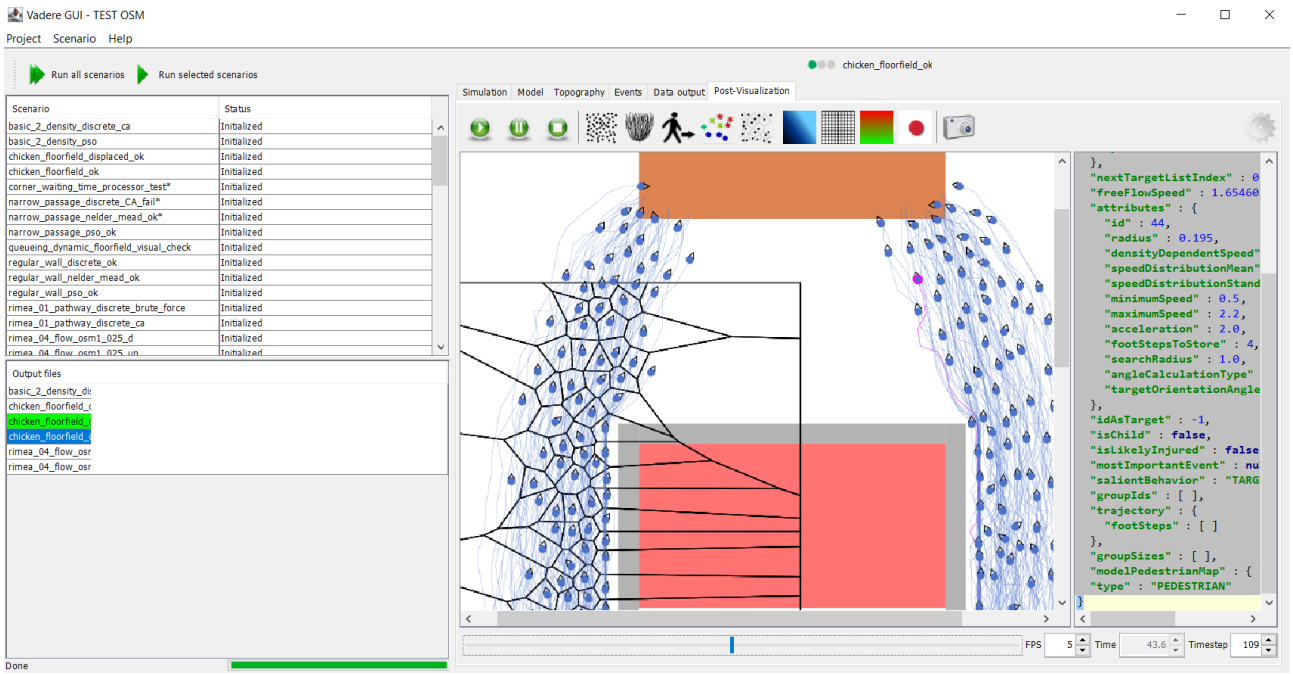


Figure 1: Graphical user interface of Vadere, with a simulation result of the "chicken test scenario" shown in the post-visualization tab.

Note: the number of points per exercise is a rough estimate of how much time you should spend on each task.

## Task 1/5: Setting up the Vadere environment — Points: 10/100

1. Download the Vadere software (not the source code version, just the compiled JAR files at `http://www.vadere.org/releases/`—you need the master branch version of the source code later, for tasks 4 and 5).
2. Start the graphical user interface of the software, and re-create the RiMEA scenarios 1 (straight line) and 6 (corner), as well as the "chicken test" you had to implement in the first exercise. Use the Optimal Steps Model (OSM, [5, 2, 6]) with its standard template. What do you observe? If you compare these scenarios with your own cellular automaton in the first exercise, what is different / similar in the model trajectories, the visualizaton, user interface, test results, ...?

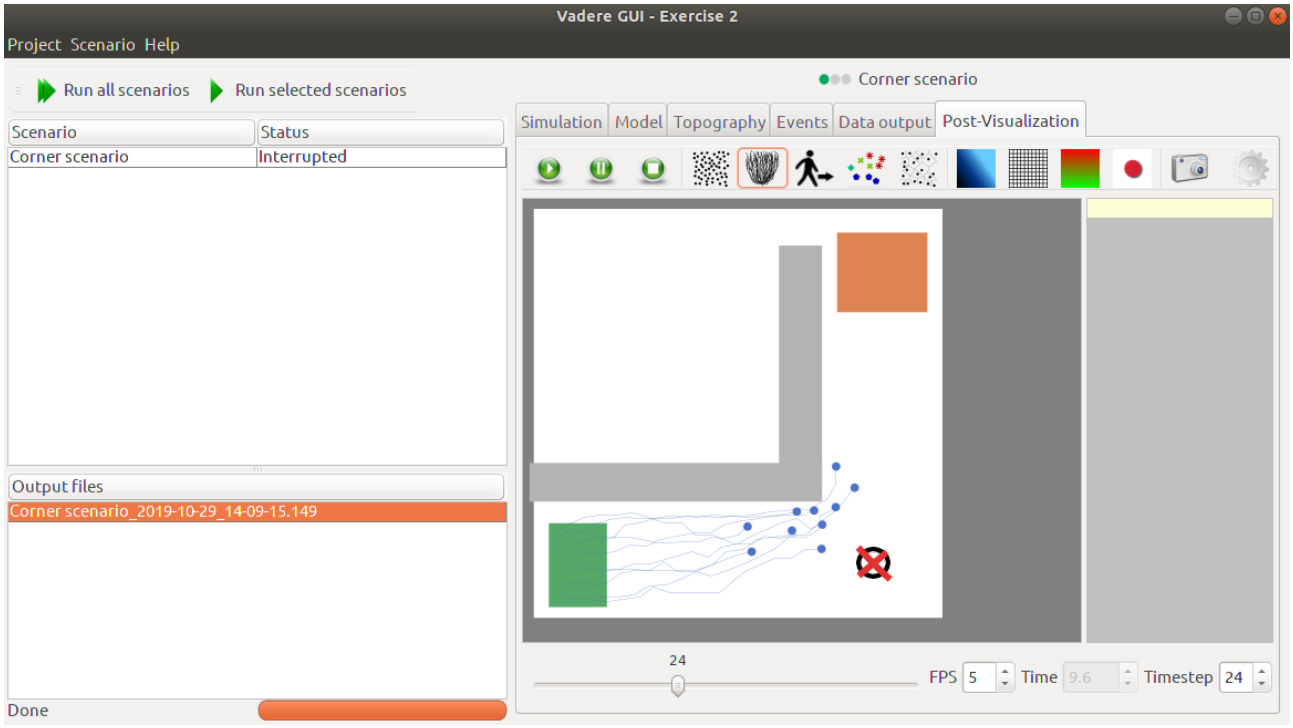


Figure 2: Ten virtual pedestrians move around a corner, modelled by the Optimal Steps Model in Vadere. The post-processing user interface shows the simulated results. The red cross in the corner is not part of the user interface, but will be important in task 3.

## Task 2/5: Simulation of the scenario with a different model — Points: 10/100

The Vadere software offers many different types of models for crowds. In this task, you have to change the model to the Social Force Model (SFM) from Helbing and co-authors [4, 3]. This model is already available as a template in the tab `Model` of a scenario. Re-run the three scenarios from the previous task, and report your findings. How do the results differ between the two models? What is similar? What do you think is the reason for the change in behavior? Do the same analysis with the Gradient Navigation Model [1, 2]. How do all three models differ, and in what respect are they similar?

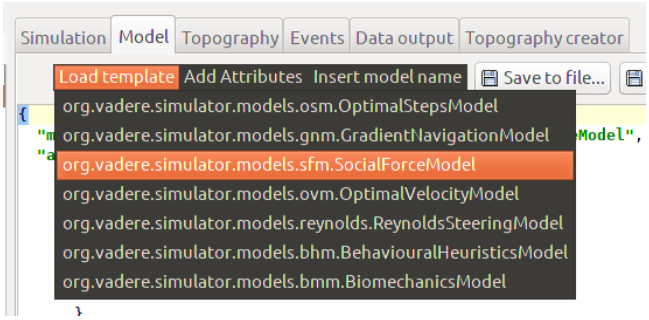


Figure 3: Selecting the Social Force Model template in the `Model` tab.

---

**Task 3/5: Using the console interface from Vadere** **Points: 15/100**

In this task, you need to access the Vadere simulation software through its console interface. This enables you to use Vadere as a "black box" from another software environment (e.g., from Python, or in another Java code). The `vadere-console.jar` can be used to run a single scenario file in the following way (enter this on the command line, in the folder with the `vadere-console.jar` file):

```
java -jar vadere-console.jar scenario-run

--scenario-file "/path/to/the/file/scenariofilename.scenario"

--output-dir="/path/to/output/folders"
```

You can access more information about the commands by typing

```
java -jar vadere-console.jar -h
```

in the command line. Use the scenario file for the corner scenario you created in the first task, and run it by calling Vadere from the command line. Compare the output files you get here to the output you obtained by running the scenario in the graphical user interface (task 1). Are the results the same?

The intended way to obtain useful output that can be post-processed is by using "output processors" in a scenario. They can be added in the user interface through the tab `Data output` of a scenario. The standard settings already write the file `postvis.traj` file, containing all positions, IDs, and targets for each pedestrian in the simulation, over all time steps.

To complete the current task, you have to modify the corner scenario file in a programming language of your choice, and then run Vadere on the new scenario file. This is in preparation for some of the next exercises, where you will use this tool to modify scenarios automatically. To insert a single pedestrian without using a source field, you can use the "`dynamicElements`" list in the topography block of a scenario file. It is usually empty if you create a new scenario. To add a pedestrian through the graphical user interface, click on the "pedestrian" icon in the topography creator tab. Figure 4 illustrates this. Vadere scenario files are text files with the information encoded in the JSON format[1]. You can open the files with a text editor, or modify them through code (as required in this exercise).

[1] See `https://www.json.org/`

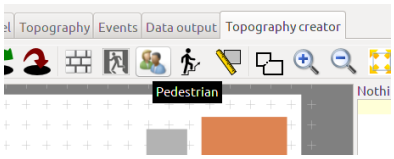



Figure 4: Left: the icon in the topography creator you need to use to add individual pedestrians. Right: once you added an individual pedestian, the highlighted information is added to the "`dynamicElements`" list in the topography section. You can also add this information programmatically, i.e. without the graphical user interface, by simply inserting the highlighted text in the scenario file.

Note that if you do not insert a number (the id of a target object in the scenario) in the `targetIds` list of the new pedestrian, the pedestrian will not move.

Use the following workflow to add pedestrians programmatically, and report your findings. Briefly describe how you change the scenario file and use the console version (i.e., what language did you use, how do you represent the scenario file in your code in order to change it, etc.).

1. Read the scenario file of the "corner scenario" (figure 2 and RiMEA test 3) you used in task 1.
2. Add a pedestrian in the corner (at the red cross shown in figure 2), away from any obstacles, to the list of individual pedestrians through code.
3. Save the scenario file with a different name.
4. Call `vadere-console.jar` with the newly modified scenario file.

How long does the inserted pedestrian take to reach the target, compared to the pedestrians starting in the source field?

## Task 4/5: Integrating a new model — Points: 30/100

Now that you have used the Vadere software with its existing features, you have to integrate a new feature. This means you have to download the source code, add the new model, and then re-build the software using Java. To re-build the software from its source code at the gitlab website[2], you need a proper IDE (I recommend IntellJ community edition[3], as eclipse may cause issues) as well as the latest java development kit (I recommend openJDK[4]). You will get points for this task if you have to concisely describe and implement the following sub-tasks and tests. If not stated otherwise, use the `OptimalStepsModel` as the main model for locomotion.

1. Integrate the SIR model in Vadere, as it is implemented in the files on Moodle (e.g. use UML diagrams, describe what the model does, etc.). Note that this implementation only contains "infective" and "susceptible" pedestrians, no "recovered/removed" yet!
2. To analyze the simulations, you have to get data about the "infective" pedestrians. You can use the modified output processor on Moodle to write the group information about the SIR model to a file.

[2]See `https://gitlab.lrz.de/vadere/vadere`
[3]IntelliJ `https://www.jetbrains.com/idea/download/`
[4]openJDK download at `https://openjdk.java.net/`, also see `http://www.vadere.org/getting-started/`

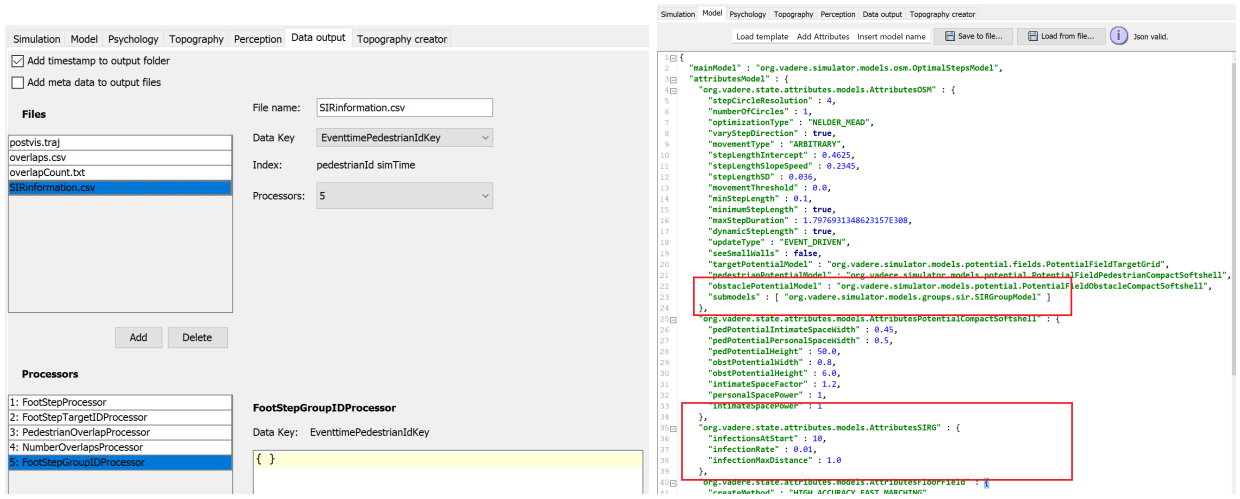


Figure 5: Left: The setup for the output processor in a vadere scenario. You have to link a new output file `SIRinformation.csv` to the output processor `FootStepGroupIDProcessor` in order to get the desired output. Right: the setup of the sub model `SIRGroupModel` with the main model `OptimalStepsModel`. Note that you have to add the sub model and its attributes.

3. The S, I, R groups are not visualized correctly in the current setup. Take a look at the method `getGroupColor` in the file `VadereGui/src/org/vadere/gui/components/model/SimulationModel.java`. You can define what color is assigned to what group by returning it depending on the current group of the given pedestrian (`ped.getGroupIds().getFirst()`). If you do that, be aware that the `FootStepProcessor` provided to you rounds the simulation time, while the group information needed for the PostVisualization does not work with rounded times. You have to decide if you (a) do not visualize the groups at all, (b) only visualize the groups during simulation, or (c) improve the processor so that the visualization works everywhere. For this exercise, (a-c) are all fine, but it may make it easier to report your results if you can visualize them.

4. You maybe realize that the distance computation for neighbors is not very efficient - all pedestrians always check all other pedestrians, with no efficient data structure to only check immediate neighbors. Use the `LinkedCellsGrid` in `org.vadere.util.geometry` to improve efficiency!

5. Run the following tests, and describe them together with their results:

   - Construct a scenario as shown in figure 6 (left), as described in the next task (but without the "recovered" state yet). Simulate the infection spreading through the static crowd, and visualize the results over time (for example, using the Dash/Plotly app you can download on Moodle). How long does it take for half the population to be infected?
   - Increase the `infectionRate` of the model using the GUI, and run the previous test again. Plot both results in one graph. How long does it take to infect half the population now?
   - Construct a corridor scenario of $40m \times 20m$, where one group of 100 people moves from left to right, another one (also 100) moves right to left. Use `useFreeSpaceOnly:true` in the source to create the pedestrians over time (not all at the same time). How many pedestrians get infected in this counter-flow?

6. In the original implementation, the `SIRGroupModel` infection rate depends on the step size of the simulation (which you can set in the "Simulation" tab of a scenario). This is not very nice, because it is difficult to interpret and also changes the infection behavior if the time discretization changes. Suggest and implement a way to decouple the infection rate and the time step as far as possible (of course, it may never be perfectly decoupled, because the simulation always depends to some degree on the time discretization).

7. Describe and motivate possible extensions (at least three, two or three sentences each) of the model beyond what will be required in task 5. You do not need to implement these extensions, just describe them and argue why they would be reasonable (e.g. more realistic, more settings, ...).

**Task 5/5: Analysis and visualization of results** **Points: 35/100**

In the previous task, you had to integrate a simple SIR model into Vadere and test its core features. Now, you have to add features to the model and test them thoroughly.

Implement the following features:

1. Add a "recovered" state that represents recovered persons[5]. Recovered persons cannot get re-infected, and cannot infect susceptible persons.

2. Add a probability for an "infective" person to become "recovered" at every time step. This is independent of where they are, or how many persons around them are "infected" or "recovered".

3. If you are not using the Dash/Plotly visualization, implement your own version and describe it. If you are using the pre-implemented version, modify it so that the "recovered" state is correctly visualized.

Test **at least** the following things of the new model. You get up to 5 bonus points for further tests (at least three for all 5 bonus points), if you describe and execute them thoroughly.

1. Construct a fairly large scenario with a source spawning 1000 pedestrians, and a target exactly on top of the source (not absorbing, see figure 6, left). To create exactly 1000 pedestrians randomly, you have to set `spawnAtRandomPositions:true` and `useFreeSpaceOnly:false`. Start with 10 infective and 990 susceptible. Visualize how the susceptible, infective, and recovered numbers change over time (see figure 6, right[6] for an example).

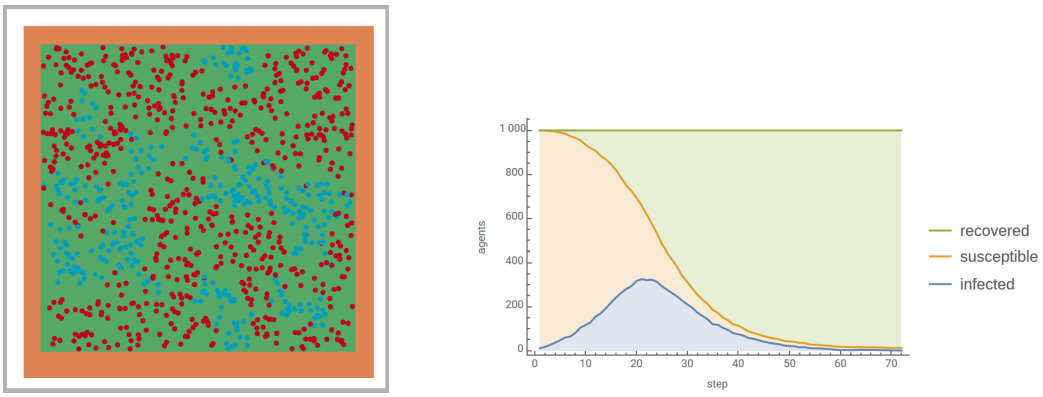


Figure 6: Left: Static scenario with 1000 pedestrians, randomly distributed in the scenario. Many of the pedestrians are already in the "infective" state, none "recovered". Right: possible (but not related) development of the SIR values over time.

2. Experiment with the infection rate $p$ and the recovery rate $r$ of your model (they are named `infectionRate` and whatever you call the recovery rate in the `AttributesSIRG` class). How do the SIR graphs change when these parameters are changed?

3. Now, decide on a fairly small infection and recovery rate. Then, construct an artificial "supermarket" scenario of at least $30 \times 30m$ in Vadere (see figure 7 for an example), where people enter at one particular location, and wander around inside the scenario before they leave again after some time. You can achieve this behavior using multiple targets, and listing their `id` in the `targetId` parameter of the source field. Try out several sources at once, to get pedestrians to take different paths. What happens if you increase the `pedPotentialPersonalSpaceWidth` in the Optimal Steps Model? Does it help to do this kind of "social distancing" in your supermarket, or is it too crowded anyway? How can you reduce the number of infections (e.g. how many people should be allowed in at any one time)?

[5]Note that in the actual SIR model, this would be called "removed"–and includes dead persons. Since the pedestrians in your simulation will still be running around, this would not make sense, so we will just make them recover and be immune.

[6]From: `https://community.wolfram.com/groups/-/m/t/1907703`

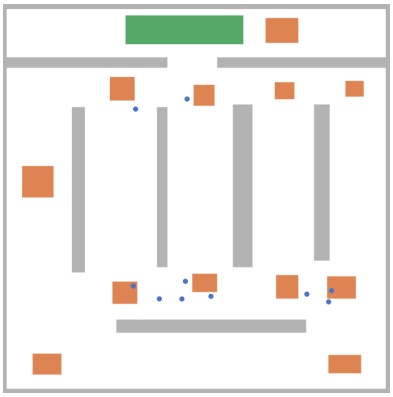

Figure 7: A possible (but very sloppy) layout of a supermarket, with several targets to specify the routing of pedestrians. They enter at the top, from the source, walk around in the supermarket and then leave through the same entrance again.

## References

[1] Felix Dietrich and Gerta Köster. Gradient navigation model for pedestrian dynamics. *Physical Review E*, 89(6):062801, 2014.

[2] Felix Dietrich, Gerta Köster, Michael Seitz, and Isabella von Sivers. Bridging the gap: From cellular automata to differential equation models for pedestrian dynamics. *Journal of Computational Science*, 5(5):841–846, 6 2014.

[3] Dirk Helbing, Illés Farkas, and Tamás Vicsek. Simulating dynamical features of escape panic. *Nature*, 407:487–490, 2000.

[4] Dirk Helbing and Péter Molnár. Social Force Model for pedestrian dynamics. *Physical Review E*, 51(5):4282–4286, 1995.

[5] Michael J. Seitz and Gerta Köster. Natural discretization of pedestrian movement in continuous space. *Physical Review E*, 86(4):046108, 2012.

[6] Isabella von Sivers and Gerta Köster. Dynamic stride length adaptation according to utility and personal space. *Transportation Research Part B: Methodological*, 74:104 – 117, 2015.