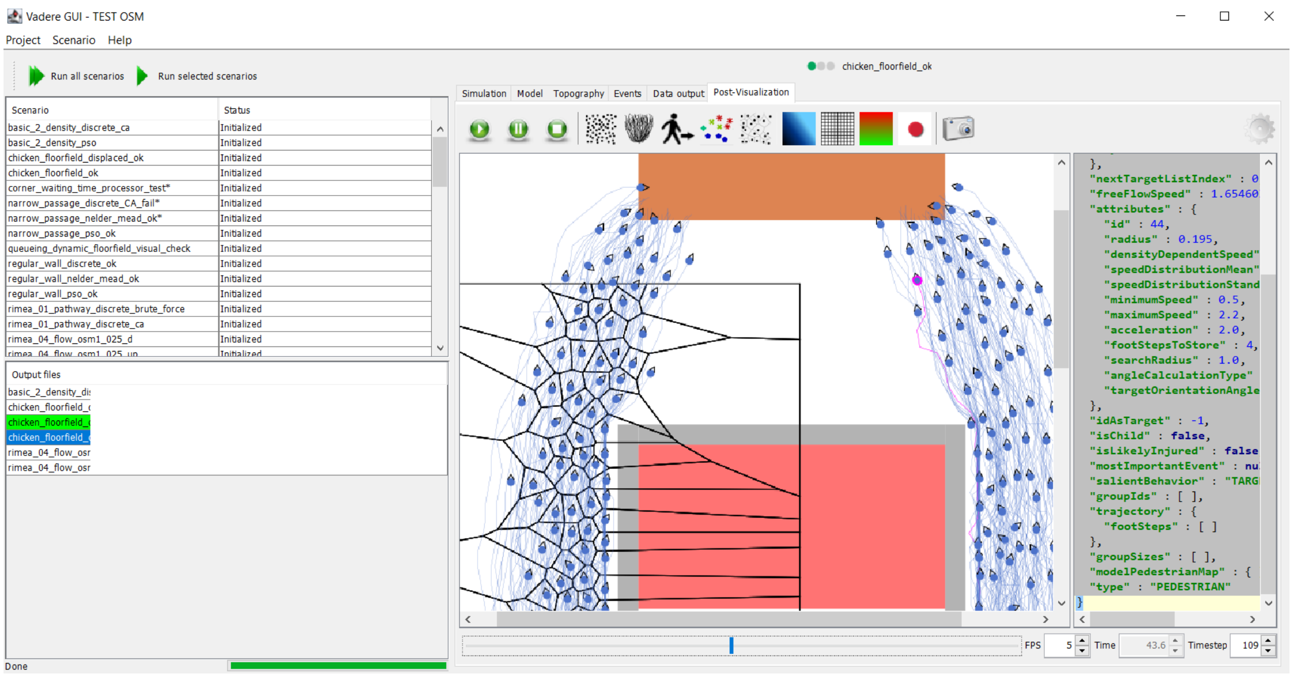
Exercise sheet 2
Using and analyzing simulation software

Due date:	2019–11–14 (2 weeks)
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 implement your own tools to compare output from different models. 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/`. I recommend to use the "stable branch" version. 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.



Scenario	Status
basic_2_density_discrete_ca	Initialized
basic_2_density_pso	Initialized
chicken_floorfield_displaced_ok	Initialized
chicken_floorfield_ok	Initialized
corner_waiting_time_processor_test*	Initialized
narrow_passage_discrete_CA_fail*	Initialized
narrow_passage_nelder_mead_ok*	Initialized
narrow_passage_pso_ok	Initialized
queueing_dynamic_floorfield_visual_check	Initialized
regular_wall_discrete_ok	Initialized
regular_wall_nelder_mead_ok	Initialized
regular_wall_pso_ok	Initialized
rimea_01_pathway_discrete_brute_force	Initialized
rimea_01_pathway_discrete_ca	Initialized
rimea_04_flow_osm1_025_d	Initialized
rimea_04_flow_osm1_025_up	Initialized

Output files
basic_2_density_di
chicken_floorfield_
chicken_floorfield_
chicken_floorfield_
rimea_04_flow_osr
rimea_04_flow_osr

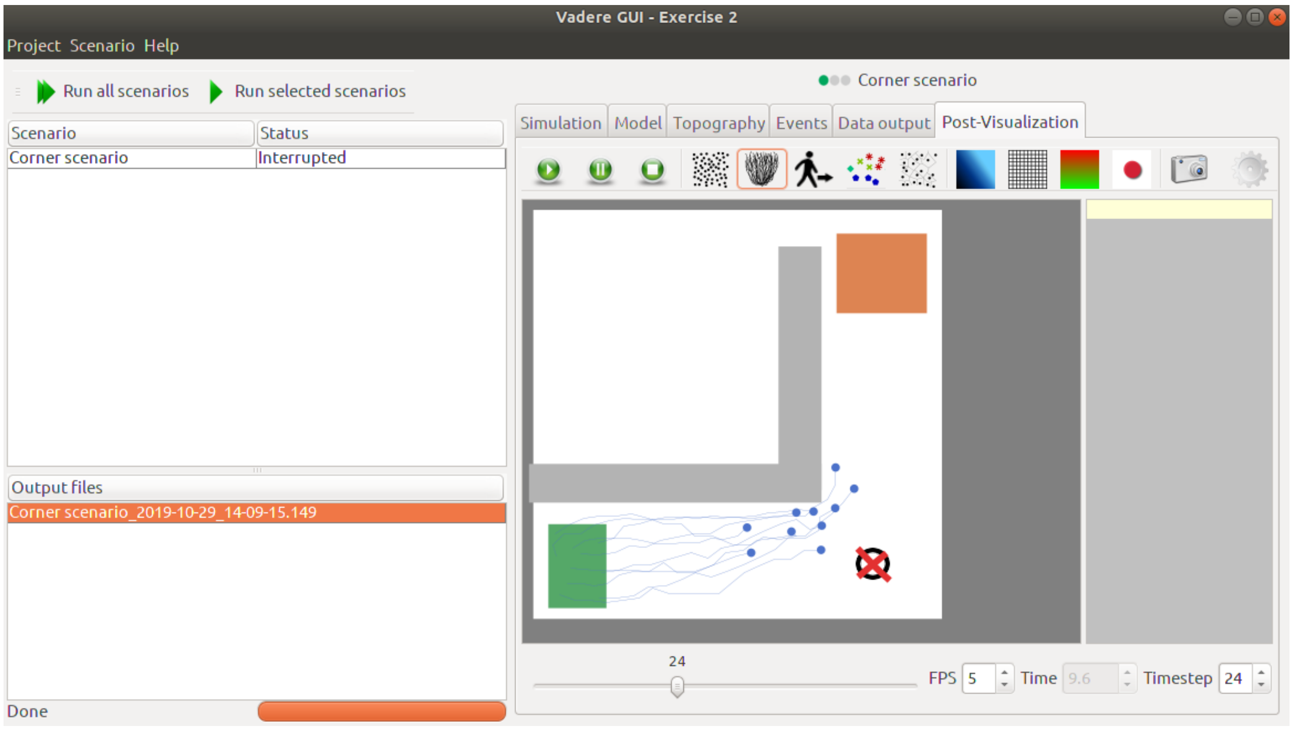
```
  },
  "nextTargetListIndex" : 0
  "freeFlowSpeed" : 1.65460
  "attributes" : {
    "id" : 44,
    "radius" : 0.195,
    "densityDependentSpeed"
    "speedDistributionMean"
    "speedDistributionStand
    "minimumSpeed" : 0.5,
    "maximumSpeed" : 2.2,
    "acceleration" : 2.0,
    "footStepsToStore" : 4,
    "searchRadius" : 1.0,
    "angleCalculationType"
    "targetOrientationAngle
  },
  "idAsTarget" : -1,
  "isChild" : false,
  "isLikelyInjured" : false
  "mostImportantEvent" : nu
  "salientBehavior" : "TARG
  "groupIds" : [ ],
  "trajectory" : {
    "footSteps" : [ ]
  },
  "groupSizes" : [ ],
  "modelPedestrianMap" : {
  "type" : "PEDESTRIAN"
}
```

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 stable branch at `http://www.vadere.org/releases/`).
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, [6, 2, 7]) 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, ...?



Scenario	Status
Corner scenario	Interrupted

Output files: Corner scenario_2019-10-29_14-09-15.149

FPS 5 Time 9.6 Timestep 24

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 [5, 4]. 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]. You will use this model again in the last task.

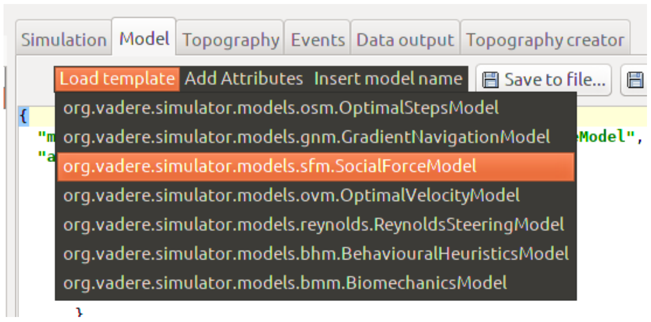

Simulation
Model
Topography
Events
Data output
Topography creator
Load template
Add Attributes
Insert model name
Save to file...
org.vadere.simulator.models.osm.OptimalStepsModel
org.vadere.simulator.models.gnm.GradientNavigationModel
org.vadere.simulator.models.sfm.SocialForceModel
org.vadere.simulator.models.ovm.OptimalVelocityModel
org.vadere.simulator.models.reynolds.ReynoldsSteeringModel
org.vadere.simulator.models.bhm.BehaviouralHeuristicsModel
org.vadere.simulator.models.bmm.BiomechanicsModel

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

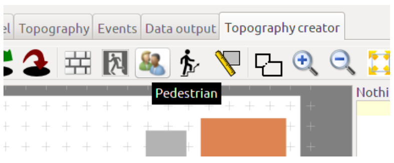
```
java -jar vadere-console.jar -h
```

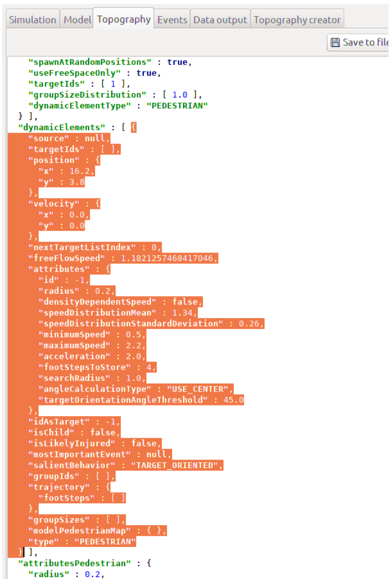
in the command line. Use the scenario file for the corner scenario you created in the first task, and run it by calling Vadere from the command line. Compare the output files you get here to the output you obtained by running the scenario in the graphical user interface (task 1). Are the results the 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.trajectories`, containing all positions, IDs, and targets for each pedestrian in the simulation, over all time steps. This file is what you need to use in this task, and also the last task of this exercise.

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 the last task, where you will need to use this workflow to analyze models 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/`





```
    "spawnAtRandomPositions" : true,
    "useFreeSpaceOnly" : true,
    "targetIds" : [ 1 ],
    "groupSizeDistribution" : [ 1.0 ],
    "dynamicElementType" : "PEDESTRIAN"
  } ],
  "dynamicElements" : [ {
    "source" : null,
    "targetIds" : [ ],
    "position" : {
      "x" : 16.2,
      "y" : 3.8
    },
    "velocity" : {
      "x" : 0.0,
      "y" : 0.0
    },
    "nextTargetListIndex" : 0,
    "freeFlowSpeed" : 1.1821257468417046,
    "attributes" : {
      "id" : -1,
      "radius" : 0.2,
      "densityDependentSpeed" : false,
      "speedDistributionMean" : 1.34,
      "speedDistributionStandardDeviation" : 0.26,
      "minimumSpeed" : 0.5,
      "maximumSpeed" : 2.2,
      "acceleration" : 2.0,
      "footStepsToStore" : 4,
      "searchRadius" : 1.0,
      "angleCalculationType" : "USE_CENTER",
      "targetOrientationAngleThreshold" : 45.0
    },
    "idAsTarget" : -1,
    "isChild" : false,
    "isLikelyInjured" : false,
    "mostImportantEvent" : null,
    "salientBehavior" : "TARGET_ORIENTED",
    "groupIds" : [ ],
    "trajectory" : {
      "footSteps" : [ ]
    },
    "groupSizes" : [ ],
    "modelPedestrianMap" : { },
    "type" : "PEDESTRIAN"
  } ],
  "attributesPedestrian" : {
    "radius" : 0.2,
```

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: Analyzing output of models **Points: 15/100**

The paper of Guy et al. [3] defines a relatively simple method to compare trajectories of a given model to observations of a real system. Their main observation is that in observations of a crowd, the entire system (consisting of many agents) can be decomposed into the individual agents. The individual movement of agents over a short period of time can be compared to model output.

To get you started, look at the Python code example on Moodle. It uses a very simple, two-dimensional system as "the truth", where a point moves around on a curve in two dimensions (the method `f_true` maps a given state `x` further in time along the curve, by `dt` physical time). The model itself is completely irrelevant, it is just a way to demonstrate the method. The coordinates of the point are observed by the function

$$x \mapsto z = h(x) = -x/2 + \cos(x)/3,$$

and these observations together with a "model" of the system are used to estimate the entropy. The model in the example already captures the true dynamics well, but corrupts the state in each time step by a small, normally distributed random noise.

The goal of this task is to understand the method of Guy et al. [3]. In this simple example, the entropy of the model can be estimated very accurately and even compared to the true entropy. Run the example code in a

Python environment, or implement the whole example in your language of choice, and then report your findings. How does the entropy change if you increase or decrease the "model_error" parameter, i.e. the difference of the model and the truth? You will get points for this task if you concisely describe

1. The model and the "truth" used for the example,
2. a few of the simplifications used in the code, compared to the algorithm of the paper (e.g., the example only considers one agent), and
3. the resulting entropies that arise when you change the model_error parameter. Why does the entropy change in this way?

Task 5/5: Analyzing output of Vadere — Points: 50/100

In the previous task, you had to analyze a very simple, two-dimensional model against observations of a "real" system. In this task, we will compare the Optimal Steps Model and the Gradient Navigation Model (GNM, [1, 2]) against each other. The observation function for this example is irrelevant, because we have full access to the true state (the positions and velocities of the pedestrians). This means you can use the identity function as the "observation" function in the algorithm1_enks method. Be careful that the GNM requires the velocity of pedestrians - if you reset the velocity at every time step to zero, you will get very different entropy results!

Choose a simple scenario for your comparison: a bottleneck, where a crowd of 25 pedestrians tries to move through a door of with 1.2m (see figure 5). Compute the entropy of the Grandient Navigation Model against the Optimal Steps Model (using the OSM as "the truth"), always for the same scenario but varying the number of pedestrians in $\{15, 20, 25, 30\}$ - i.e., you should get four values of the entropy, one for each number of pedestrians. Why do these values differ, even though you are comparing models?

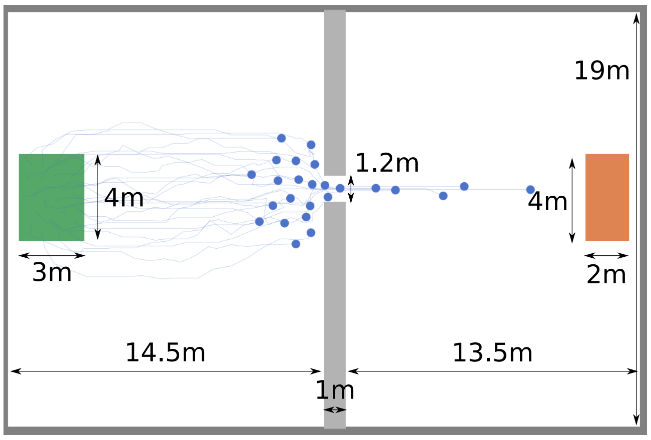


Figure 5: The bottleneck scenario, with 25 pedestrians modelled through the Optimal Steps Model moving from the starting area (left) through the bottleneck to the target area (right). It is crucial that you use the same dimensions for the scenario, because otherwise, the results of different groups cannot be compared.

Break down the task into the following steps. Even if you do not implement the model entropy estimation algorithm, you can still get points for the correct modifications of the scenario files.

1. Simulate the bottleneck scenario with the OSM, to generate the "real" data set consisting of the positions of the individual pedestrians over time (in task 4, this was the data stored in the zk variable).
2. The algorithm of Guy et al. [3] requires you to run Vadere with the positions of the pedestrians prescribed by the simulation run in the first step. To achieve this, you should use the scenario modification code you wrote in task 3. To know where to place the individual pedestrians, you have to use the postvis.trajectories file created in the OSM run. Describe how you use this output file to modify the scenario file appropriately. Remember to remove the source field in the new scenario file, because you are controlling the initial point for every pedestrian programmatically now.

3. Once you are able to run the GNM in a scenario with arbitrary initialized pedestrian positions, use the algorithm from Guy et al. to estimate the entropy of the GNM with respect to the OSM. If you implement the full entropy estimation algorithm, you will also have to add normally distributed noise to the OSM positions and then run Vadere.
4. Repeat the estimation for the varying number of pedestrians in the scenario, noted above: $\{15, 20, 25, 30\}$. Each time, you have to run the OSM again to create the "true states". The result are four values of the entropy of the GNM with respect to the OSM.
5. 5 bonus points: estimate the entropy of the OSM with respect to the GNM (i.e., reverse the experiment). Why do the entropy values differ?
6. 20 bonus points: estimate the entropy of your cellular automaton in exercise 1 with respect to one of the models from Vadere. This might pose several challenging problems, for example, you have to be able to interpret the positions from your simulation environment in terms of the positions of Vadere. Carefully describe what you needed to change and implement to complete this task.

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, 2014.

[3] Stephen J. Guy, Jur van den Berg, Wenxi Liu, Rynson Lau, Ming C. Lin, and Dinesh Manocha. A statistical similarity measure for aggregate crowd dynamics. *ACM Transactions on Graphics*, 31(6), 2012.

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

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

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

[7] 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.