**Report for exercise 2 from group H**


| | |
|---|---|
| Tasks addressed: | 5 |
| Authors: | Taiba Basit (03734212) |
| | Zeenat Farheen (03734213) |
| | Fabian Nhan (03687620) |
| Last compiled: | 2021–05–12 |
| Source code: | https://github.com/Combo1/MLCMS/tree/main/Exercise2 |


The work on tasks was divided in the following way:

| | | |
|---|---|---|
| Taiba Basit (03734212) | Task 1 | 33% |
| | Task 2 | 33% |
| | Task 3 | 33% |
| | Task 4 | 33% |
| | Task 5 | 33% |
| Zeenat Farheen (03734213) | Task 1 | 33% |
| | Task 2 | 33% |
| | Task 3 | 33% |
| | Task 4 | 33% |
| | Task 5 | 33% |
| Fabian Nhan (03687620) | Task 1 | 33% |
| | Task 2 | 33% |
| | Task 3 | 33% |
| | Task 4 | 33% |
| | Task 5 | 33% |

**Report on task Task 1/5: Setting up the Vadere environment**

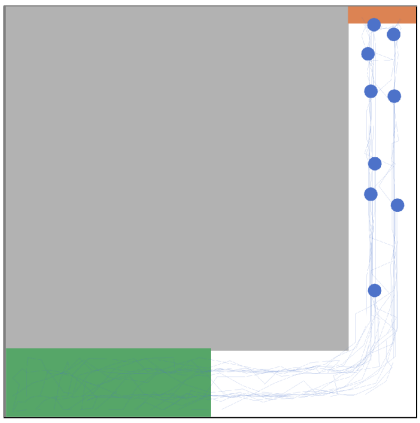
In the RiMEA scenario 1 running with the Optimal Steps Model, we observe that the pedestrians distributed themselves into small groups, used the whole width of the passage and always left some space around them. In our implementation in Exercise 1, the pedestrians moved straight towards the target, while groups formed according to their speed. The width of the passage was not a factor here, since they all moved straight in equal speed towards the target.

(a) Before

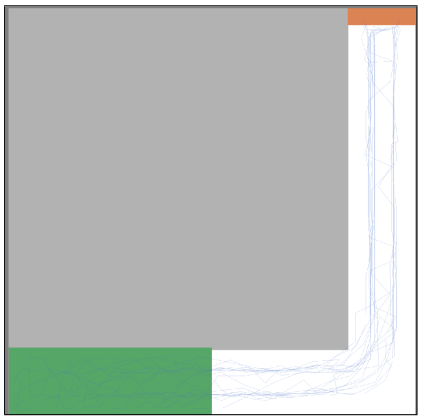
(b) After

Figure 1: Rimea Test 1 with OSM model

For scenario 6 some pedestrians were able to get ahead of the others and much more quickly reached the finish, while most of the others were obstructed due to the small passage. With the additional corner the pedestrians seemed to try to go around it as tight as possible. In our implementation it seemed like the pedestrian clouds were much more pronounced, but the pedestrians also seemed to take the corner really closely.



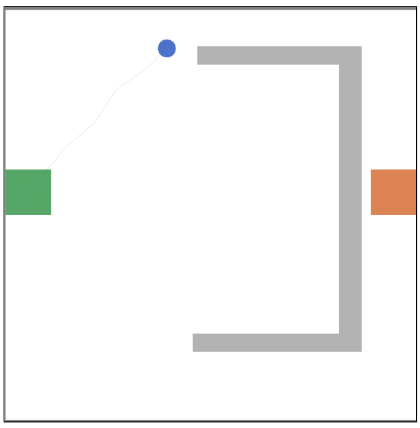
(a) Before



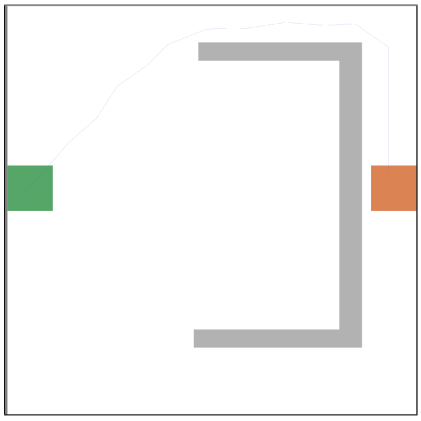
(b) After

Figure 2: Rimea Test 6 with OSM model

For the chicken test the result was not much different to our implementation the pedestrian moved quite efficient by moving around the boundaries of the cage.



(a) Before

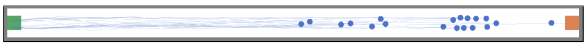


(b) After

Figure 3: Chicken Test with OSM model

**Report on task Task 2/5, Simulation of the scenario with a different model**

1. To analyze the difference between the OSM and SFM models for RiMEAscenario 1, we ran the simulation in post visualization. With the help of trajectory lines, we could see that the pedestrians take a more smoother and straight path in SFM model as compared to OSM model.Moreover,in OSM model, the pedestrians maintain a distance from neighbouring pedestrians throughout the simulation, while they move quite closely to each other in SFM.

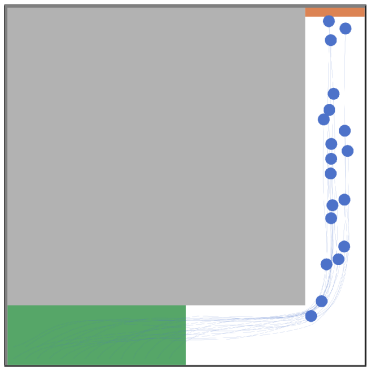


(a) Before

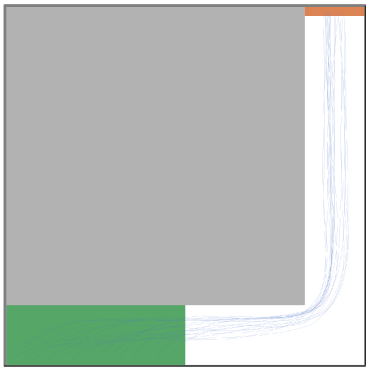
(b) After

Figure 4: Rimea Test 1 with SFM model

2. In RiMEA scenario 6, similar differences to the first scenario exist. It is evident in post visulaization of OSM model scenario 6 that to maintain the distance between pedestrians , a pedestrian even steps back for a moment to allow other pedestrian to move ahead. Although all pedestrians try to move as closely to the corner as possble, the need to maintain distance often pushes them away from the corner resulting in the observed zig-zag trajectory of few pedestrians. In SFM, the pedestrians successfully move close to the corner often overlapping each other in the process and not maintaining any social distance.



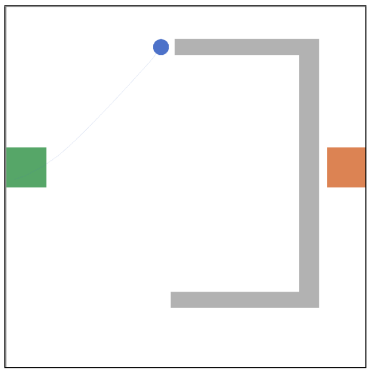
(a) Before



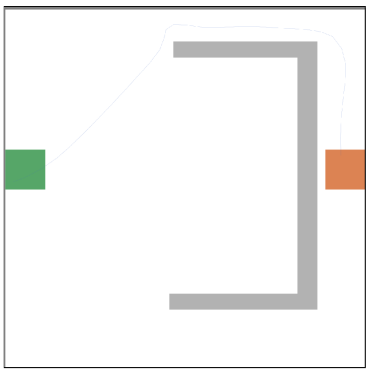
(b) After

Figure 5: Rimea Test 6 with SFM model

3. In the chicken test scenario, in OSM model, the pedestrian path of movement is much smoother than in the SFM model. In SFM model, the pedestrian takes the shortest path to the top corner and then re-adjusts it's path by making a sharp movement to avoid the obstacle. While in OSM, the model modifies the direction of motion from the beginning and the pedestrian smoothly moves over the corner to reach the target. In both the models, the pedestrians successfully avoid getting trapped inside the obstacle.



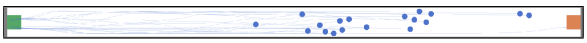
(a) Before



(b) After

Figure 6: Chicken Test with SFM model

4. To analyze the simulation of GNM model for RiMEAscenario 1, we ran the simulation in post visualization. We can clearly see that the simulation o GNM model ismuch more smoother than SFM model. This is because of the underlying implementation differences between the two models. The GNM only needs three equations,as opposed to four in SFM, to describe motion of one pedestrian. GNM also implements floor field to steer pedestrians which is computed by solving eikonal equations using efficient fast marching algorithm. The mollification techniques used in GNM, which are missing in SFM, are also responsible for the enhanced smoothness.

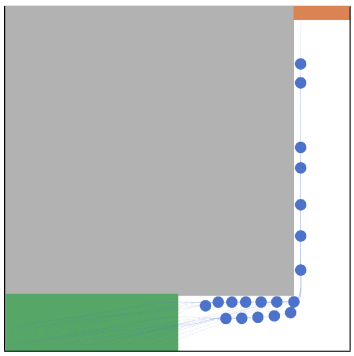


(a) Before

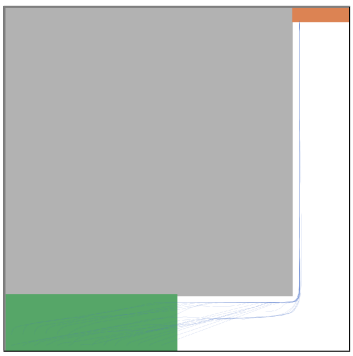
(b) After

Figure 7: Rimea Test 1 with GNM model

5. In RiMEA scenario 6, stark differences between the 2 models can be observed. In SFM model, the pedestrians try to move as closely as possible to the corner, while in GNM model we can see that each pedestrian waits to get closest to the corner. Due to this we see a single line of movement in GNM model near the corner which is not the case in SFM model. Moreover, in SFM model the pedestrians reach the target in a fairly distributed way along the width of the target, whereas in GNM model, all pedestrians travel in the same line one after the other to reach the target.



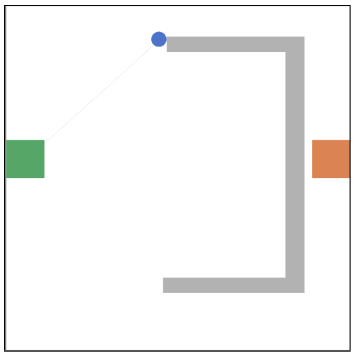
(a) Before



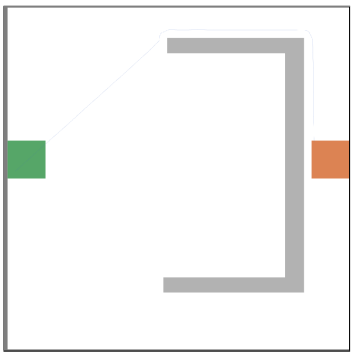
(b) After

Figure 8: Rimea Test 6 with GNM model

6. In the chicken test scenario, in GNM model we can notice that upon reaching the upper corner of the obstacle, the pedestrian makes a slight jump. This jump or upward motion is more pronounced in SFM model as compared to GNM model. This is due to the fact pedestrians are not subject to inertia in GNM model as they are in SFM model. Model induced oscillations are avoided completely since no actual forces are present in GNM model.



(a) Before



(b) After

Figure 9: Chicken Test with GNM model

### Report on task Task 3/5, Using the console interface from Vadere

The results obtained by running the corner scenario file in vadere gui is saved in directory "Task1.2_2021-05-08_gui". The output obtained by running the same file with console is saved in "Task1.2_2021-05-08_console". We observe that the output of both the files in postvis.traj files are exactly same. The only difference is that while we can control how long the simulation runs for in vadere gui, the simulation always runs for a fixed number of timesteps in vadere console which is 1330.

To edit the scenario file, we created a python script name "scenario_editor.py" . We extracted the json data for a single pedestrian by adding it through gui and saved it in a separate file named "dynamic_pedestrian.json". Our objective was to programmatically insert this json data for a pedestrian to the 'dynamicElements' field in the 'topography' of a scenario file. To do this, the script reads the scenario file for corner case(Rimea 3) and the dynamic_pedestrian.json file. It appends the pedestrian data in the appropriate field of scenario file i.e. ["scenario"]["topography"]["dynamicElements"]. We also change the name of the scenario to 'Task1.2_modified' and save it as a new scenario file.

```
# Here is code snippet - scenario_editor.py
def add_pedestrian(scenario, x, y, outfilename):
    pedestrian_obj = read_json_file('/home/isarsoft/MLCMS/Exercise2/task3/dynamic_pedestrian.json')
    pedestrian_obj["position"]["x"] = x
    pedestrian_obj["position"]["y"] = y
    target_list = scenario["scenario"]["topography"]["targets"]
    for target in target_list:
        pedestrian_obj["targetIds"].append(target["id"])
    scenario["scenario"]["topography"]["dynamicElements"].append(pedestrian_obj)
    scenario["name"] = "Task1.2_modified"
    with open(outfilename, 'w') as f:
        json.dump(scenario, f)
```

After this we run the scenario via vadere console as well as in vadere gui. After analyzing the output via post visualization, we observe that the pedestrian initally at position 11.4, 1.4 reaches the target before the pedestrians generated from the source as it is closer to target as shown in fig 10

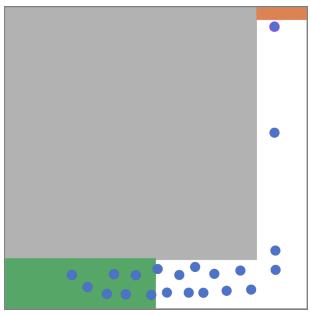


Figure 10: The pedestrian(purple outline) at (11.4,1.4) reaches before others

We also analyzed the output files generated after simulation. We can see that the pedestrian at 11.4,1.4 reaches the target the fastest.

### Report on task Task 4/5: Integrating a new model

1. We placed all the files in there respective location as it was mentioned in the models file. Here is the explanation of what the model does:

   - **SIRGroup:** Builds a group given a list of pedestrians. It assigns them a groupID and size and supports a variety of auxiliary functions.
   - **SIRGroupModel:** Implementation for different groups SIR (susceptible, infected, recovered). Contains a list with groups + groupID, a mapping from groupID to group, attributes of the groups (infectionsAtStart, infectionRate, infectionMaxDistance), their topography, their targets and the amount of infected people.

- **SIRType:** Enum with susceptible, infected and recovered.
- **AbstractGroupModel:** Abstract class template to overwrite. Each Group has to have registerNewMember, getNewGroup, assignToGroup.
- **AttributesSIRG:** Class containing the attributes for the SIR model like infectionsAtStart, infectionRate, infectionMaxDistance which are set by the user. It also provides getter methods to retrieve the values of these attributes.
- **FootStepGroupIDProcessor:** This processor processes foot step of each pedestrian from different groups based on their group id and logs this value as the key EventtimePedestrianIdKey.

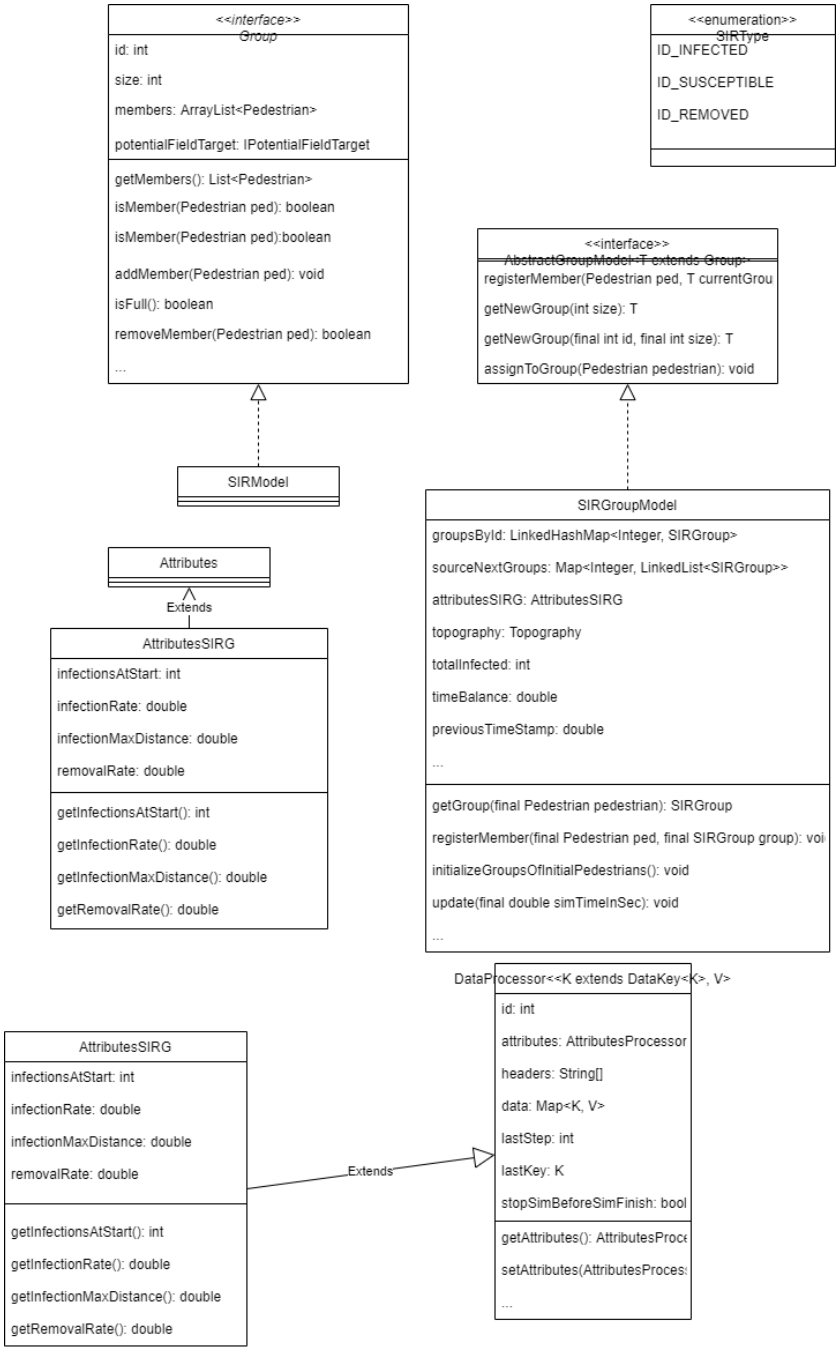
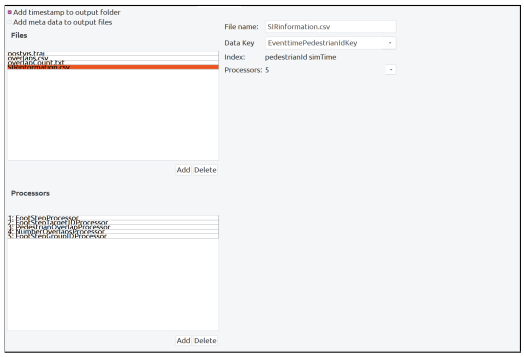



Figure 11: UML Class Diagram

2. For this task, we had to analyse the infective pedestrians. We linked the output file SIRinformation.csv to output processor FootStepGroupIDProcessor. We also added SIRGroupModel as sub model to our main model OptimalStepsModel. We also added the attributes of the AttributesSIRG to the attributes field.

```
{
  "mainModel" : "org.vadere.simulator.models.osm.OptimalStepsModel",
  "attributesModel" : {
    "org.vadere.state.attributes.models.AttributesOSM" : {
      "stepCircleResolution" : 4,
      "numberOfCircles" : 1,
      "optimizationType" : "NELDER_MEAD",
      "varyStepDirection" : true,
      "movementType" : "ARBITRARY",
      "stepLengthIntercept" : 0.4625,
      "stepLengthSlopeSpeed" : 0.2345,
      "stepLengthSD" : 0.036,
      "movementThreshold" : 0.0,
      "minStepLength" : 0.1,
      "minimumStepLength" : true,
      "maxStepDuration" : 1.7976931348623157E308,
      "dynamicStepLength" : true,
      "updateType" : "EVENT_DRIVEN",
      "seeSmallWalls" : false,
      "targetPotentialModel" : "org.vadere.simulator.models.potential.fields.PotentialFieldTargetGrid",
      "pedestrianPotentialModel" : "org.vadere.simulator.models.potential.PotentialFieldPedestrianCompactSoftshell",
      "obstaclePotentialModel" : "org.vadere.simulator.models.potential.PotentialFieldObstacleCompactSoftshell",
      "submodels" : [ "org.vadere.simulator.models.groups.sir.SIRGroupModel" ]
    },
    "org.vadere.state.attributes.models.AttributesPotentialCompactSoftshell" : {
      "pedPotentialIntimateSpaceWidth" : 0.45,
      "pedPotentialPersonalSpaceWidth" : 1.2,
      "pedPotentialHeight" : 50.0,
      "obstPotentialWidth" : 0.8,
      "obstPotentialHeight" : 6.0,
      "intimateSpaceFactor" : 1.2,
      "personalSpacePower" : 1,
      "intimateSpacePower" : 1
    },
    "org.vadere.state.attributes.models.AttributesFloorField" : {
      "createMethod" : "HIGH_ACCURACY_FAST_MARCHING",
      "potentialFieldResolution" : 0.1,
      "obstacleGridPenalty" : 0.1,
      "targetAttractionStrength" : 1.0,
      "cacheType" : "NO_CACHE",
      "cacheDir" : "",
      "timeCostAttributes" : {
        "standardDeviation" : 0.7,
        "type" : "UNIT",
        "obstacleDensityWeight" : 3.5,
        "pedestrianSameTargetDensityWeight" : 3.5,
        "pedestrianOtherTargetDensityWeight" : 3.5,
        "pedestrianWeight" : 3.5,
        "queueWidthLoading" : 1.0,
        "pedestrianDynamicWeight" : 6.0,
        "loadingType" : "CONSTANT",
        "width" : 0.2,
        "height" : 1.0
      }
    },
    "org.vadere.state.attributes.models.AttributesSIRG" : {
      "infectionsAtStart" : 10,
      "infectionRate" : 0.01,
      "infectionMaxDistance" : 1.0,
      "removalRate" : 0.01
    }
  }
}
```

(a) Added SIRGroupModel as submodel of OSM model




(b) Linked the SIRinformation.csv to FootStepIDProcessor

Figure 12: Required changes for running SIR model

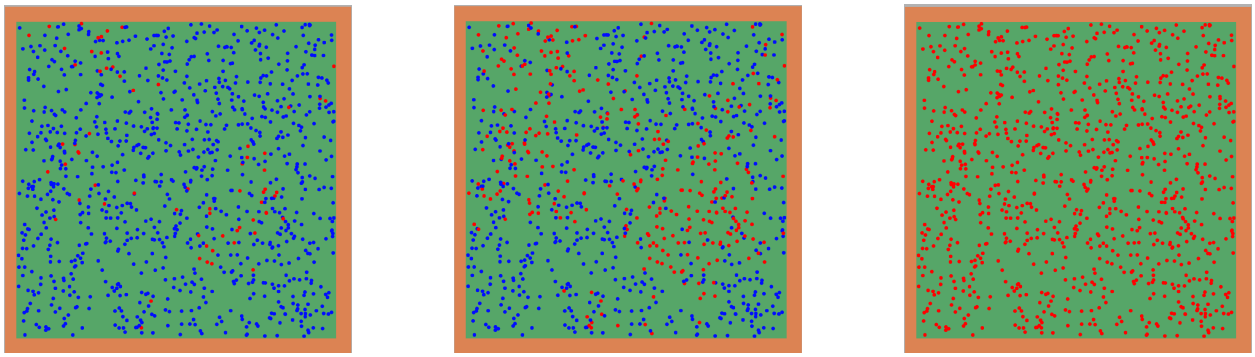
3. For this task, we edited the getGroupColor in the file VadereGui/src/org/vadere/gui/components/model/SimulationModel.java. We first obtained the information regarding groupID of a pedestrian using ped.getGroupIds().getFirst() method. Then based on the groupID we assigned different colors for different group. For our case we assigned blue to the susecptible, red for the infected group and green for the recovered group. Also, since just changing the code did not get reflected in vadere GUI we changed the coloring option present in vadere gui to colorbygroup . This helped in visualizing the different colors for different group. So our implementation does not visualize the group color in the PostVisualization, but during simulation.

```
//Here is code snippet for the coloring based on group ID
int groupId = ped.getGroupIds().getFirst();
Color c = colorMap.get(groupId);
//Replaced by specific color value for SIR Groups
        if (groupId == 0) {
                c = new Color(255,0,0);
                colorMap.put(groupId, c);
        } else if(groupId == 2) {
                c = new Color(0,255,0);
                colorMap.put(groupId, c);
        } else if(groupId == 1) {
                c = new Color(0,0,255);
                colorMap.put(groupId, c);
        } else {
                if (c == null) {
                    c = new Color(Color.HSBtoRGB(random.nextFloat(), 1f, 0.75f));
                        colorMap.put(groupId, c);
                }
        }
```

4. For this task we altered the update method inside of the SIRGroupModel file. We used the LinkedCellsGrid class to get a list of pedestrians, whose neighbours we could access quickly through the getObjects method. We obtained the LinkedCellsGrid through the topography class. For the second for loop, we then only iterated over the neighbours, which were a circle (radius = infection max distance) away from the current pedestrian. In the beginning there were some misunderstandings about this task, since we did not know what the distance metrics was used for and which file to edit, which would be nice to add in the task description for next year.

```
// code snippet Task 4.4  - SIRGroupModel.java - update():
LinkedCellsGrid<Pedestrian> neighbours;
List<Pedestrian> neighbours_of_neighbours;
/* update the second for loops such that only neighbours which are in the Infection radius
of the pedestrian are counted towards the infection check */
neighbours = topography.getSpatialMap(Pedestrian.class);
for(Pedestrian p : c.getElements()) {
    neighbours_of_neighbours = neighbours.getObjects(p.getPosition(),
        attributesSIRG.getInfectionMaxDistance());
        // loop over neighbors and set infected if we are close
        for(Pedestrian p_neighbor : neighbours_of_neighbours) {
            ...
        }
}
```

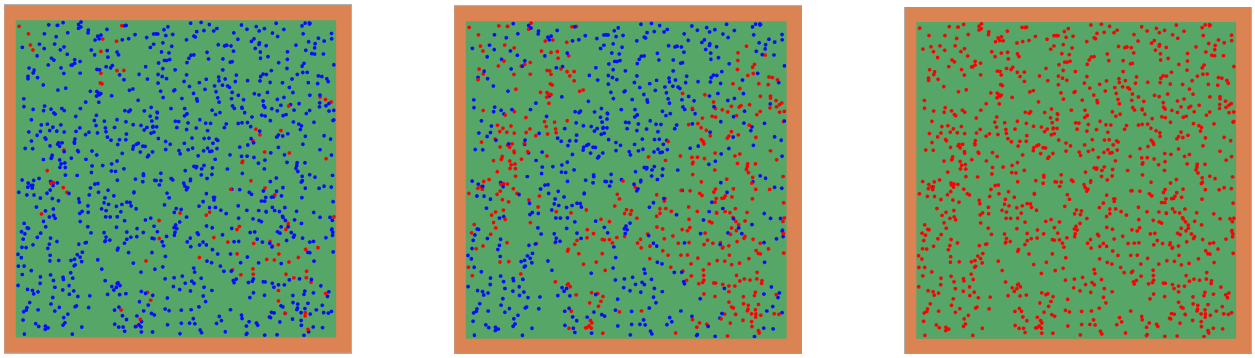
5. (a) For the first case, we built a scenario with source of size 10m X 10m and the placed the target above it and set the absorbing property of the target to be false. The number of pedestrians in this scenario are 1000 and are generated randomly. We set the parameters infectionsAtStart=10, infectionRate=0.001, and infectionMaxDistance= 1.0. Screenshots of simulation are shown in Figure 13.



(a) Begin (b) Mid (c) End

Figure 13: Simulation with infectionRate=0.001

(b) In the next scenario, we increased the infection rate to 0.01. The visualization obtained are shown in Figure 14. We can clearly observe in the graph for both the scenarios in Figure 15, that the population reaches a state of 50% infected an d 50% recovered much earlier in the case with high infection rate. This point is reached approximately at 7 sec of after the start of simulation, while it is reached at 50 sec for the case with lower infection rate.



(a) Begin (b) Mid (c) End

Figure 14: Simulation with infectionRate 0.01

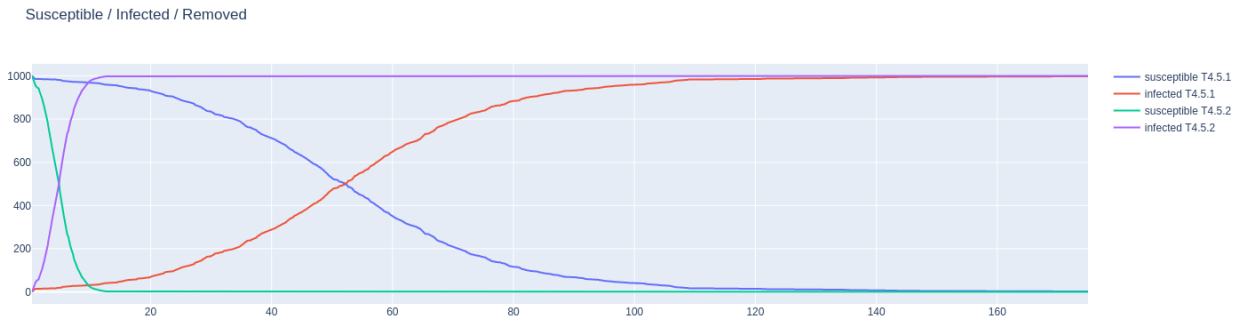
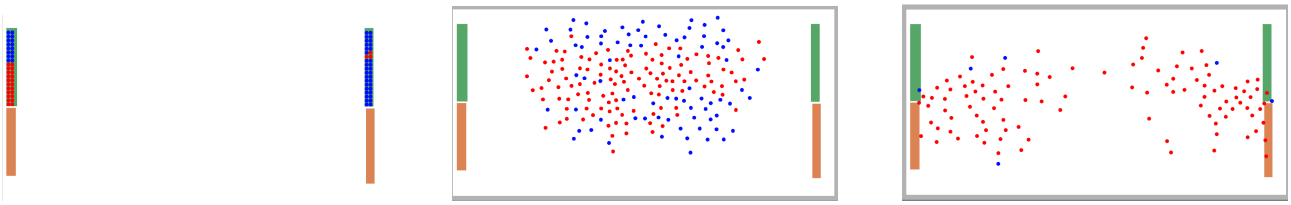



Figure 15: Combined plot for infectionRate 0.001 and 0.01

(c) For this case, we built a corridor of size 40m X 20m with two source and target at both ends. From both the source we generated 100 pedestrians and set the target to the one which is present at the other end. We also, set the useFreeSpaceOnly:true so that the pedestrians were created overtime. The targets are set with absorbing:true. For this case we set the infectionsAtStart=20, InfectionRate=0.05 and infectionMaxDistance=1.0. We see from the visualization in Figure 16a,that the initial configuration has more infected pedestrians in the left source. Over time, the two sets of pedestrians collide mid way and infect their neighbours. The graph in Figure 17 shows the total number of pedestrians that get infected in this counter flow is 181 out of 200.



(a) Begin (b) Mid (c) End

Figure 16: Simulation with infectionRate 0.05

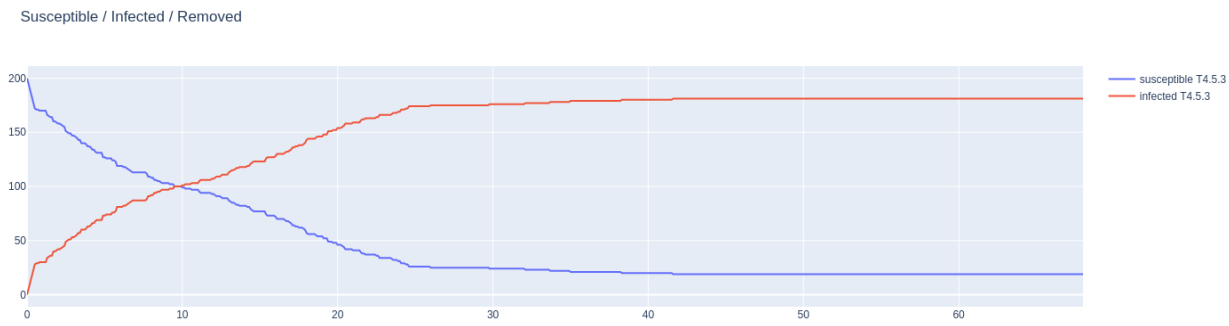



Figure 17: Plot with infectionRate 0.05

6. To decouple the infection rate from the step size of the simulation, we propose to interpret the infection rate as infections per second. Our solution adds two attributes: a timeBalance and a previousTimeStep. Both of them are doubles. To decouple the infection rate we update, at each update call, the timeBalance by the duration which passed since the last update call + the previous value of timeBalance. Then we update the previousTimeStamp to the value of the current simulation time. While the timeBalance is bigger or equal to 1, we infect the pedestrians and subtract the timeBalance by 1. Through this method the pedestrians get infected after each second, however the problem with this approach is that if we only simulate less than a second none of the pedestrians get infected, so for this issue to not influence the results. Our simulation needs to run for an extended amount of time such that this issue does not change the result.

```
// code snippet Task 4.6  - SIRGroupModel.java - update:
timeBalance = (simTimeInSec - previousTimeStamp) + timeBalance;
previousTimeStamp = simTimeInSec;
while(timeBalance >= 1) {
    ...
    timeBalance = timebalance - 1;
}
```

7. (a) For the simulation to have a practical application, it would be nice to have the option of adding several floors for the pedestrians to move to. Then simulating huge buildings such as airports with escalators and lifts would be much more realistic.

   (b) Another useful feature can be to calculate how many neighbouring pedestrians does each infected pedestrian infect during the simulation. So if we can control the speed of each pedestrian and the distance between pedestrians, this could help in seeing the relationship between the speed, distance and the number of infections that each pedestrian is responsible for.

   (c) Adding a feature of age for each pedestrian will help us in controlling the degree of a pedestrian being susceptible to the infection.

   (d) To make the system more realistic, we can add another state carrier which would represent the state of a pedestrian that can infect other pedestrians. This would be different from the state infected which could move to a state of recovery without infecting other pedestrians.

---

**Report on task Task 5/5, Analysis and visualization of results**

---

1. The state for recovered person was already present in the file SIRType named ID_removed. We decided to leave the name unchanged even though ID_REMOVED represents a state of pedestrians that have recovered and are moving around. To make sure that a recovered person does not get re-infected, we added the following lines of code in update() function in SIRGroupModel.

```java
// code snippet Task 5  - SIRGroupModel.java - update():
for(Pedestrian p_neighbor : c.getElements()) {
    if(p == p_neighbor || getGroup(p_neighbor).getID() != SIRType.ID_INFECTED.ordinal())
                continue;
        double dist = p.getPosition().distance(p_neighbor.getPosition());
        if (dist < attributesSIRG.getInfectionMaxDistance() &&
        this.random.nextDouble() < attributesSIRG.getInfectionRate()) {
                SIRGroup g = getGroup(p);
                if (g.getID() == SIRType.ID_SUSCEPTIBLE.ordinal()) {
                    elementRemoved(p);
                        assignToGroup(p, SIRType.ID_INFECTED.ordinal());
                        }
                }
        }
```

We check if the current pedestrian is in infected state, and it has a probability of getting recovered, we update it's state to ID_REMOVED group. This pedestrian is then removed from the infected group and added to the removed group.

2. We added the variable removalRate with a default value of 0.01 in AttributesSIRG.java which represents the probability for an infected person to become recovered at every time step.We generate a random number every time step to check if the infected pedestrian has a chance of recovery which makes it independent of how many person around them are infected or recovered.

```java
// code snippet Task 5  - AttributesSIRG.java:
private double removalRate=0.01;
public double getRemovalRate() {
                return removalRate;
}
```

3. To visualize the simulation output, we made the following changes in utils.py in SIRvisualization.In file_df_to_count_df(...), we added argument to this function for receiving group id for ID_REMOVED state. Previously, the code only kept a group count of infected and susceptible pedestrians. We added the code to check if a person is in infected state and change to recovered state in a particular simulation time step, we update the group count for both infected by -1 and recovered by +1, and break from loop.

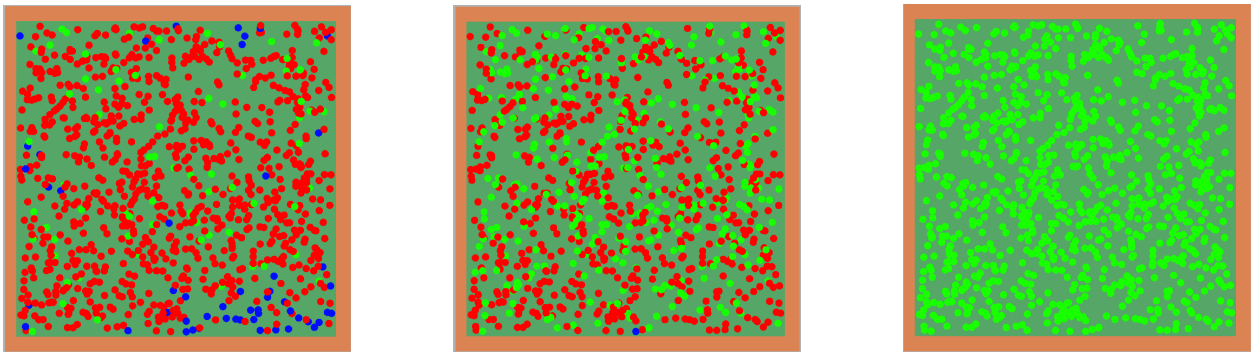
```python
# utils.py
for (st, g) in simtime_group:
    if g != current_state and g == ID_INFECTED and current_state == ID_SUSCEPTIBLE:
        current_state = g
        group_counts.loc[group_counts['simTime'] > st, 'group-s'] -= 1
        group_counts.loc[group_counts['simTime'] > st, 'group-i'] += 1

    elif g != current_state and g == ID_REMOVED and current_state == ID_INFECTED:
        current_state = g
        group_counts.loc[group_counts['simTime'] > st, 'group-i'] -= 1
        group_counts.loc[group_counts['simTime'] > st, 'group-r'] += 1
        break
```

We also edited create_folder_data_scatter(), to plot the graph for recovered pedestrians, we added the state ID_REMOVED and set it to group id 2 according to the output in SIRinformation.csv. We have also added the code to build scatter plot for recovered pedestrian.

```python
 scatter_r = go.Scatter(x=group_counts['simTime'],
                        y=group_counts['group-r'],
                        name='removed ' + os.path.basename(folder),
                        mode='lines')
    return [scatter_s, scatter_i,scatter_r], group_counts
```

## TESTS

1. For this test, we set the total numbers of pedestrians as 1000 by setting spawnNumber = 1000. These pedestrians are generated randomly by setting spawnAtRandomPositions = true and useFreeSpaceOnly = false. To start with 10 infective pedestrians, we set infectionsAtStart = 10. We set the infectionRate = 0.01 and recoveryRate = 0.01. During visualization of the scenario, we see that the state of susceptible pedestrians in blue change to infected(red) and finally to recovered(green). As the recovery rate and infection rate are same, we do not see a state of population that is fully infected which can also be inferred from the graph in Figure 19. We can also see from the graph the point of time where half of the population is infected and the other half is recovered.



(a) Begin (b) Mid (c) End

Figure 18: Simulation with infectionRate 0.01 and recoveryRate 0.01

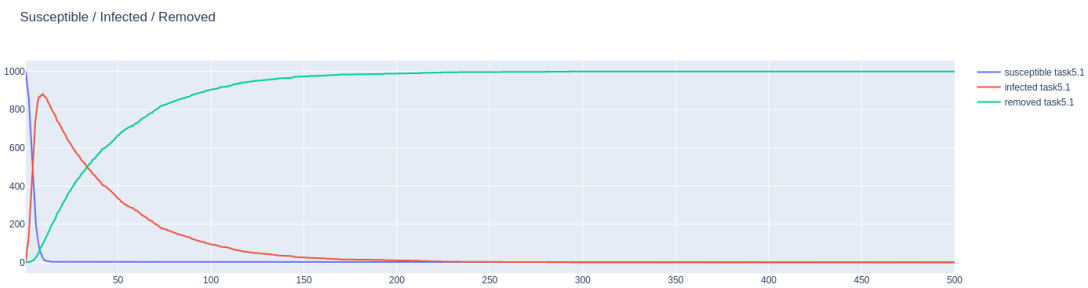
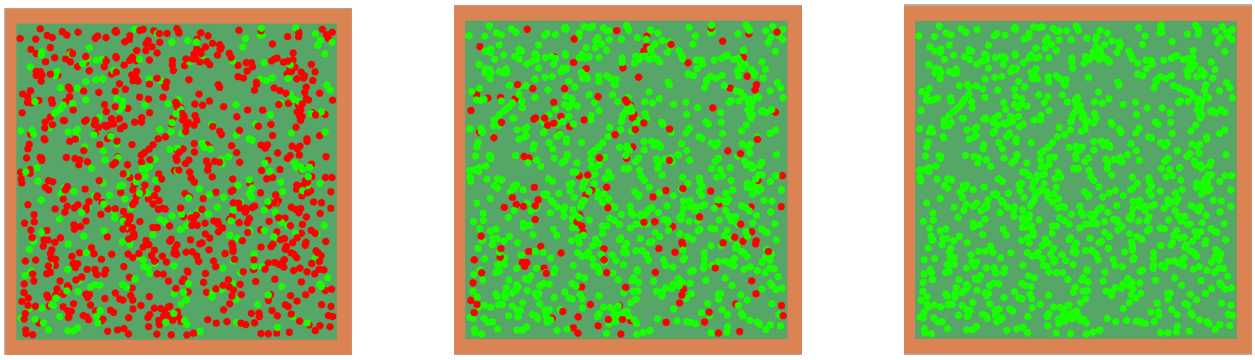



Figure 19: Plot for infectionRate 0.01 and recoveryRate 0.01

2. For this task, we used the same topography as the previous task. We chose infectionsAtStart to be 10 and total number of pedestrians in the simulation to be 1000. We simulated the model with different sets of values for parameters infectionRate and recoveryRate. In all following cases, we ran the simulation for the complete duration of 500 seconds :

   (a) **High infectionRate, High recoveryRate:**
   We set infectionRate = 0.1 and recoveryRate = 0.1. From Figure 21, we notice a very sharp spike in infection count towards the beginning of simulation. Moreover the crossover point (where 50% of population is infected and the rest 50% is recovered) is achieved quite early. We also observe that the infection scatter plot never reaches a peak of 1000 infected pedestrians. This is a reasonable result because of the very high recovery rate.



(a) Begin (b) Mid (c) End

Figure 20: Simulation with infectionRate 0.1 and recoveryRate 0.1

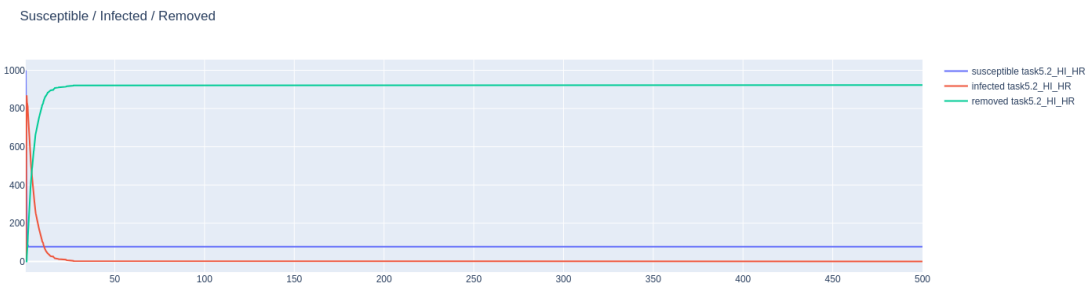
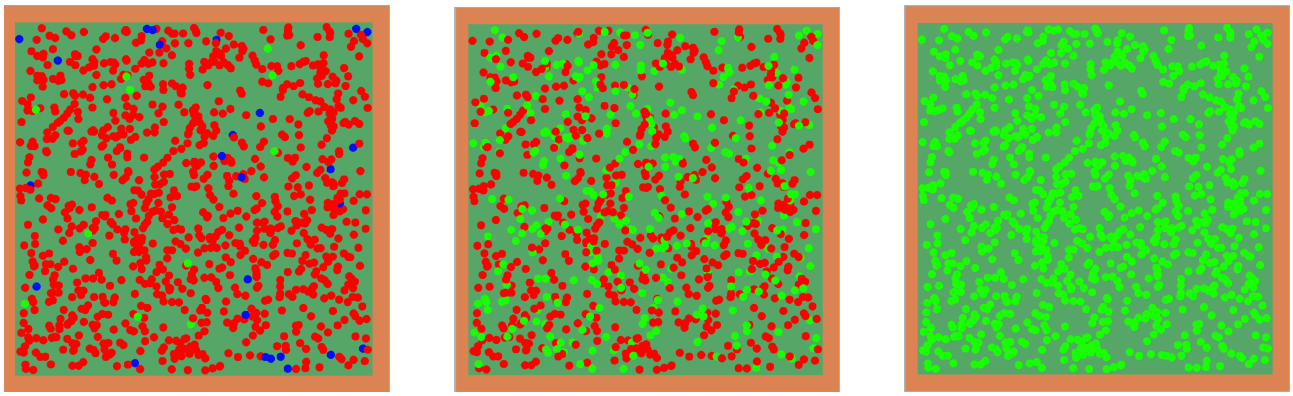



Figure 21: Plot with infectionRate 0.1 and recoveryRate 0.1

(b) **High infectionRate, Low recoveryRate:**
We set infectionRate = 0.1 and recoveryRate = 0.01. From the graph in figure 23 ,we see that the infection plot quickly reaches a peak of 1000 infected pedestrians in the beginning of the simulation. This state is achieved because of the very low recovery rate which results in the entire population being infected at a point of time. In addition to this, the crossover point is reached much later as compared to the previous case with high infection rate and recovery rate.



(a) Begin (b) Mid (c) End

Figure 22: Simulation with infectionRate 0.1 and recoveryRate 0.01

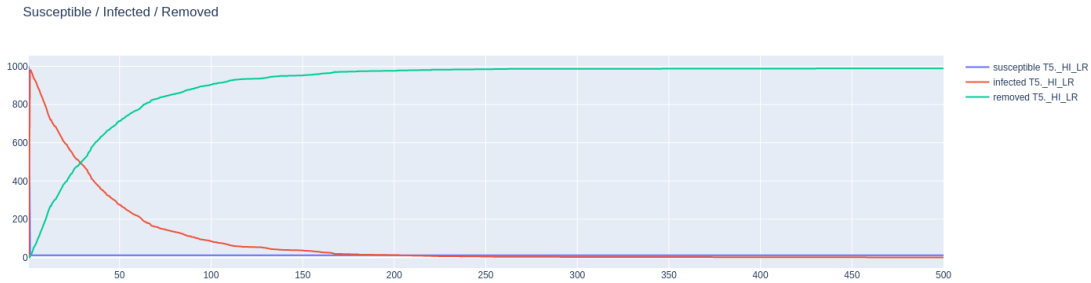
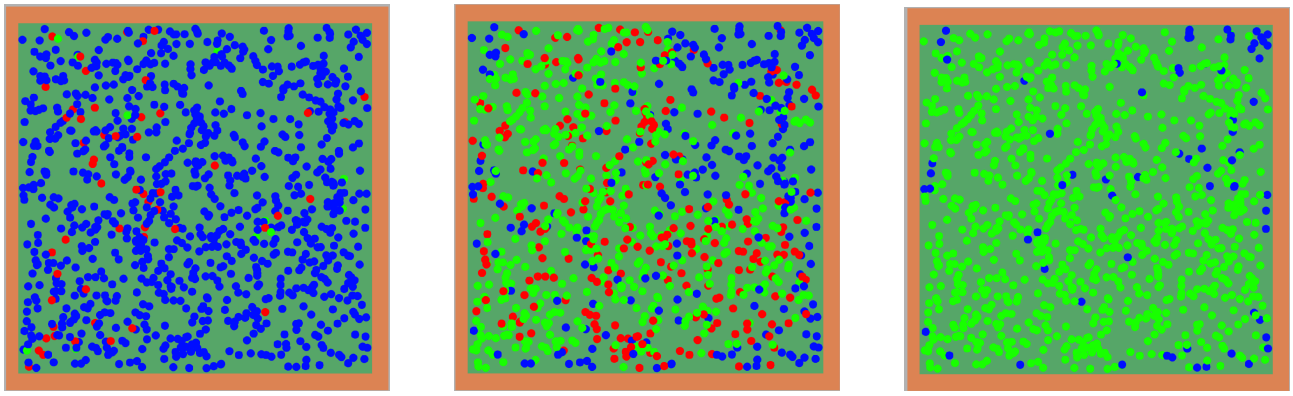



Figure 23: plot with infectionRate 0.1 and recoveryRate 0.01

(c) **Low infectionRate, High recoveryRate:**
We set infectionRate = 0.01 and recoveryRate = 0.1. From Figure 25, we observe that at the end of the simulation, there are pedestrians who are never got infected. This is also clearly visible in the visualization image Figure 24c. Another interesting point to note is the visibly smaller peak of infection plot, which is the result of very low infection rate and high recovery rate.



(a) Begin (b) Mid (c) End

Figure 24: Simulation with infectionRate 0.01 and recoveryRate 0.1

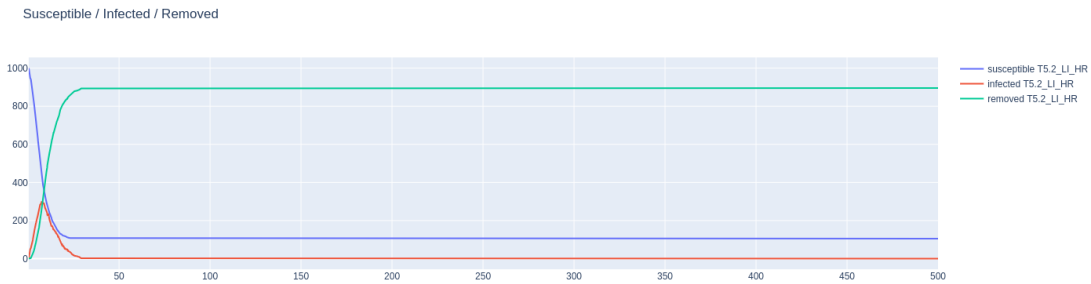
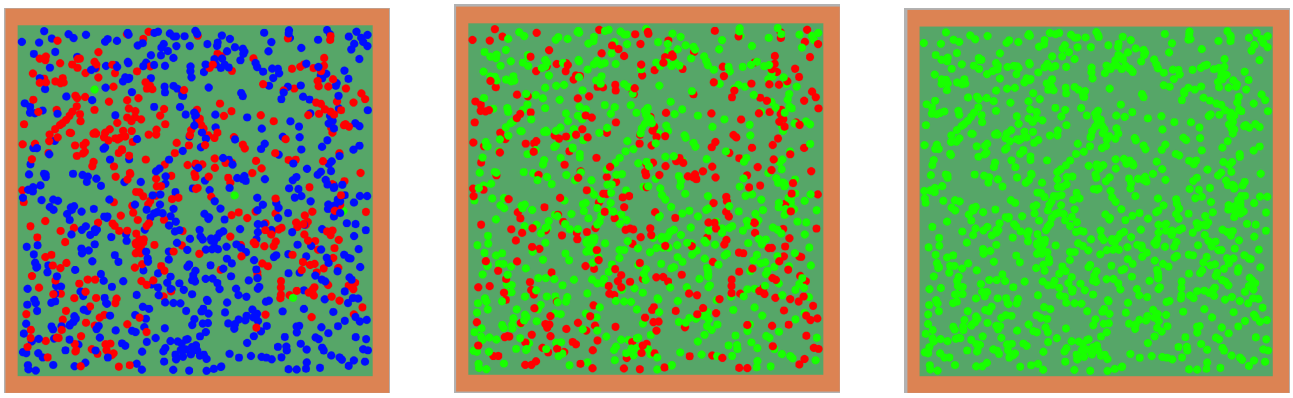



Figure 25: Plot with infectionRate 0.01 and recoveryRate 0.1

(d) **Low infectionRate, Low recoveryRate:**
We set infectionRate = 0.01 and recoveryRate = 0.01. The results obtained in this case are similar to the first one, except that the crossover point is reached much later in this case as shown in figure 27. The gradually decreasing number of susceptible pedestrians is visible in this case much more clearly than in case with high infection rate and high recovery rate.



(a) Begin (b) Mid (c) End

Figure 26: Simulation with infectionRate 0.01 and recoveryRate 0.01

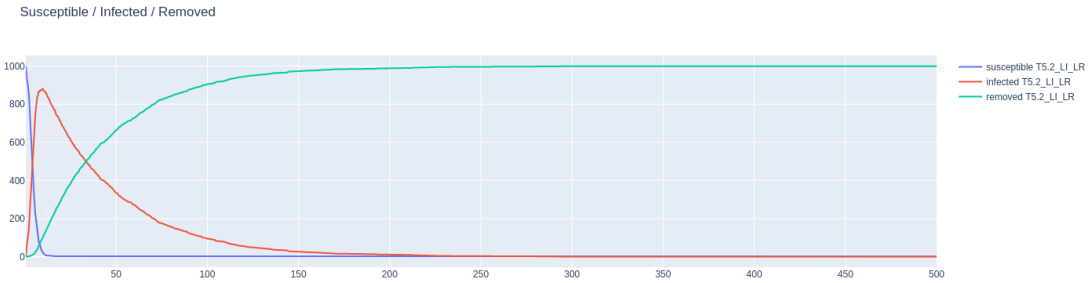
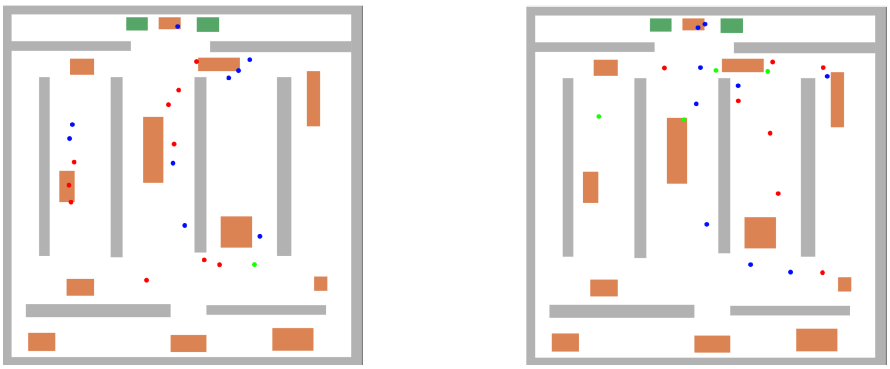



Figure 27: Plot with infectionRate 0.01 and recoveryRate 0.01

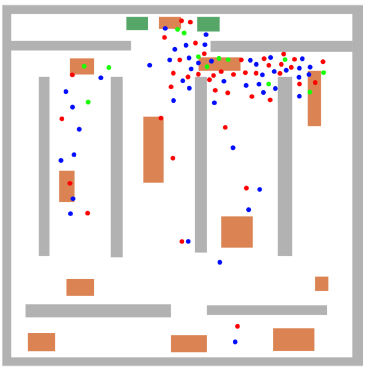
3. In this test case, we built a scenario representing a supermarket with one entrance and one exit. There are 2 sources near the entrance. The pedestrians spawned from both the sources travel 2 different trajectories of mulitple targets with higher chances of collision near the exit where we have placed a bottleneck. We test the scenario with different setting for total number of pedestrians and infected persons.

   Initially, we set spawnNumber = 20 and infectionAtStart = 10. This is represented in Figure 28a. In another simulation, we set spawnNumber = 100 and infectionAtStart = 50. This is represented in Figure 29a. Comparing these two cases, we notice that the pedestrians crowd near the exit bottleneck in the later case and the chances of infection are significantly higher in this case.

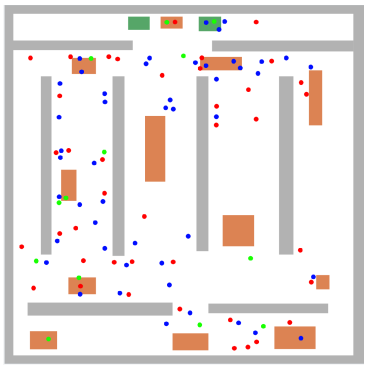


(a) Without Distance (b) With Distance

Figure 28: With 10 pedestrian



(a) Without Distance



(b) With Distance

Figure 29: With 50 pedestrian

In the next step, we altered the parameter pedPotentialPersonalSpaceWidth from default value of 1.2 to 6.0. Figure 28b and 29b show the visualization for both cases with this change. It is interesting to see that in case of a larger distance field between pedestrians the infection count is significantly lower than the previous scenario as shown in plot of Figure 30 and 31.

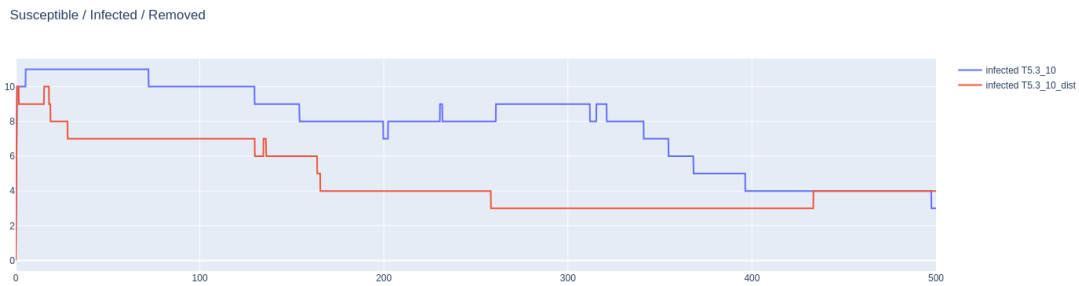



Figure 30: Plot of infected pedestrian with 10 pedestrians

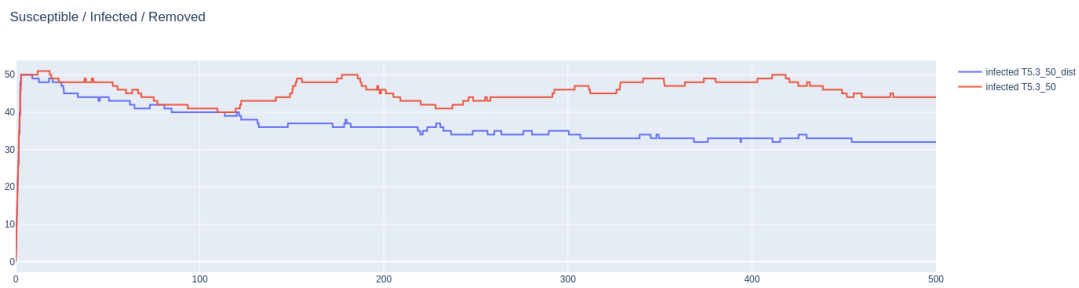



Figure 31: Plot of infected with 50 pedestrians

This can be understood as the result of less crowding near the exit which results in the significant drop. Therefore, we can conclude that by practising "social distancing", we can achieve much lower cases of infections in a supermarket scenario. Additionally, by controlling the number of pedestrians that are allowed inside the supermarket at a time, we can achieve a smoother in-flow and out-flow of pedestrians, which will result in a lower count of infections. We can also set different non-overlapping paths and multiple entry and exit points in a supermarket to further decrease the infections.