ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Lecture with Computer Exercises: Modelling and Simulating Social Systems with MATLAB

Project Report

Evacuation Bottleneck: Simulation and analysis of an evacuation of a high-school building with MATLAB

Alexander Jöhl & Tomáš Nyitray

Zürich
December 2011

ETH

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of Originality

This sheet must be signed and enclosed with every piece of written work submitted at ETH.

I hereby declare that the written work I have submitted entitled

Evacuation Bottleneck:
Simulation and analysis of an evacuation of a high-school building with MATLAB

is original work which I alone have authored and which is written in my own words.*

Author(s)

Last name	First name
Jöhl	Alexander

Supervising lecturer

Last name	First name
Balietti	Stefano

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (http://www.ethz.ch/students/exams/plagiarism_s_en.pdf). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

8.12.2011 Zürich
Place and date

A. Jöhl
Signature

*Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Print form

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of Originality

This sheet must be signed and enclosed with every piece of written work submitted at ETH.

I hereby declare that the written work I have submitted entitled

Evacuation Bottleneck:
Simulation and analysis of an evacuation of a high-school building with MATLAB

is original work which I alone have authored and which is written in my own words.*

Author(s)

Last name	First name
Nyitray	Tomáš

Supervising lecturer

Last name	First name
Balietti	Stefano

With the signature I declare that I have been informed regarding normal academic citation rules and that I have read and understood the information on 'Citation etiquette' (http://www.ethz.ch/students/exams/plagiarism_s_en.pdf). The citation conventions usual to the discipline in question here have been respected.

The above written work may be tested electronically for plagiarism.

Place and date	Signature
8.12.2011 Zürich	Nyitray

*Co-authored work: The signatures of all authors are required. Each signature attests to the originality of the entire piece of written work in its final form.

Print form

Agreement for free-download

We hereby agree to make our source code of this project freely available for download from the web pages of the SOMS chair. Furthermore, we assure that all source code is written by ourselves and is not violating any copyright restrictions.

Alexander Jöhl

Tomáš Nyitray

Contents

1. Introduction and Motivation

For our project we have selected the topic **Evacuation Bottleneck**. This topic was very interesting for us and especially for Tomáš, because the high-school he was attending for 8 years went through massive reconstruction where the attic was rebuilt and new rooms installed. This on one hand increases the capacity of the building but on the other hand the new fire stairs were built to face the problem of increased evacuation time. In this project we compare the evacuation times for the old and for the renovated building.

The building used in this project is the building of Gymnázium Ľudovíta Jaroslava Šuleka in Komárno. It was built it 1910 and since then its appearance remained the same. You can see it in figure 1.

Fig. 1: **The main building of Gymnázium Ľudovíta Jaroslava Šuleka [1]**

Since the 70's the building had insufficient space, so some rooms had to be located in other buildings. This was very dangerous for students, because the school is located next to a main street and the additional rooms were situated on the other side of the street. Therefore in 1998 the huge renovation of the attic was started. In 2004 the renovation was finished and finally after 30 years the whole school is situated in one building. In 2005 the high-school had 521 students and 40 teachers. [1]

2. Description of the model

The Students and teachers (Let us call them agents) do not behave very clever in our MATLAB model, which means they are not using the principle of the minimal waiting time, but they are using the principle of the shortest distance.

One can say that this is not the reality but what we have to consider is that not each agent is autonomous. They are just young students that follow the basic evacuation rules. In Slovakia the basic rule is that the class must stay together and they have to evacuate through the nearest exit. In MATLAB simulation they only use the principle of the shortest distance and the rule that they must stay together will be neglected. During a fire alarm several teachers are just standing in the floors navigating the students and students from different classes will just mix.

In our simulation a force map for each floor is used. Agents are also interacting with each other which results in a decrease of their velocity in the queue. The force map was calculated using the Toolbox Fast Marching. The fast marching algorithm generates a force field in the space, which contains the distance of every point to the next exit. It is used mostly in fluid dynamics applications. Figure 2 shows the generation of the vector field with the shortest path to an exit. This vector field is later used for navigation of agents in space.

http://www.mathworks.com/matlabcentral/fileexchange/6110

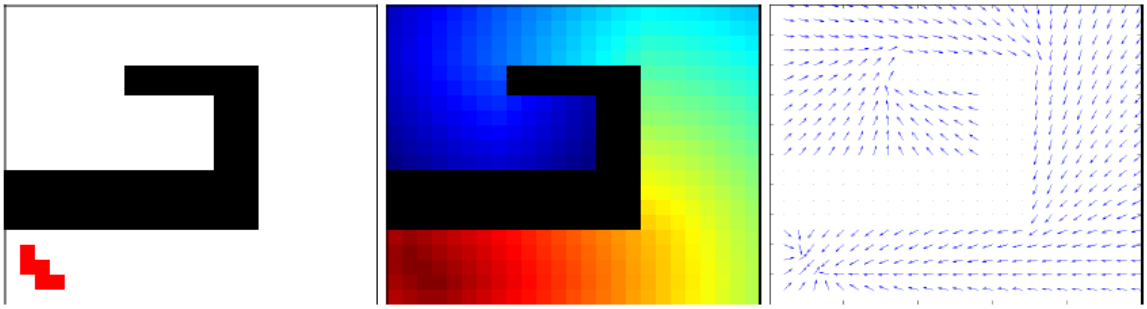


Fig. 2: **Development of the vector field containing the shortest path to the next emergency exit [2]**

Agents from upper floors are moving down the stairs to a lower floor. This enables us to approximate the time difference between the last agent exiting through the main entrance and the last exiting through the fire stairs.

The simulation time is very short (up to 30 seconds) and the visualization is done in more or less 400 iterations which are computed in a very short time. We were trying to provide an easy and robust simulation tool for evacuations of any type. The equations are chosen the way that by changing only two parameters the agent behavior can be easily adjusted.

3. The Building

Since the simulation and the pictures processing is done in MATLAB, we have to find a way to implement the drawings. We were inspired by the work of Philipp Heer and Lukas Bühler on Pedestrian Dynamics, Airplane Evacuation Simulation in last semester but we were facing some problems with the reading of our .bmp files. Therefore we have decided to use grey scale .bmp picture painted in GIMP. In figure 3 we can see how the reconstruction influences the shape of the floors that are shown in figures 4, 5, 6 and 7.

Fig. 3: **Some photos before and after the reconstruction [1]**

3.1. Building drawing

The imread() command in MATLAB reads in the grey scale images with values from 0 to 255. 0 corresponds to black color and 255 to white. For exits we have used the value of 110 and for agents 140. The number of agents is determined by the current number of students [3], but we have also added several other agents to the rooms that are used occasionally (labs, lecture rooms, etc.).
In the following figures we can see how the floor layouts were re-drawn to .bmp files that were used for the simulation of evacuation.

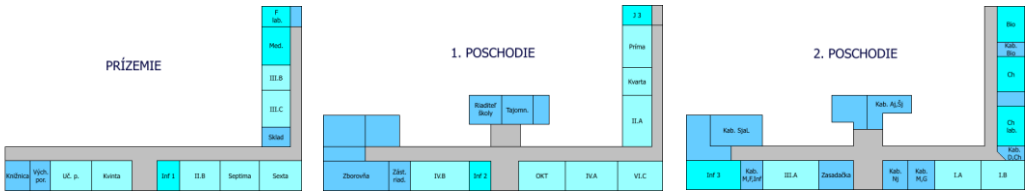


Fig. 4: **Ground, First and the Second floor [3]**

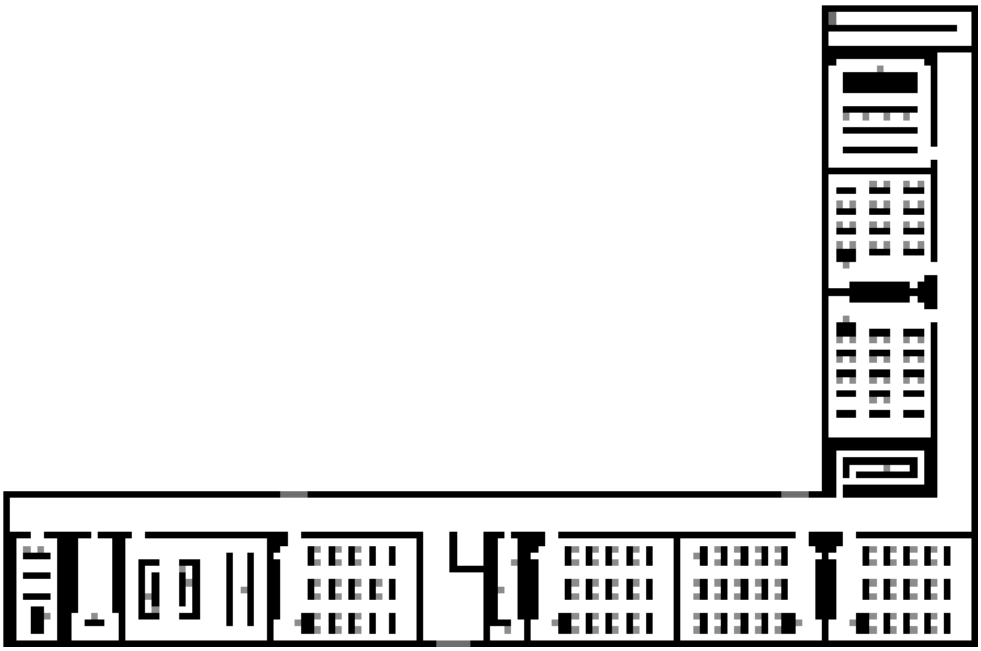


Fig. 5: **Ground floor layout used in MATLAB**

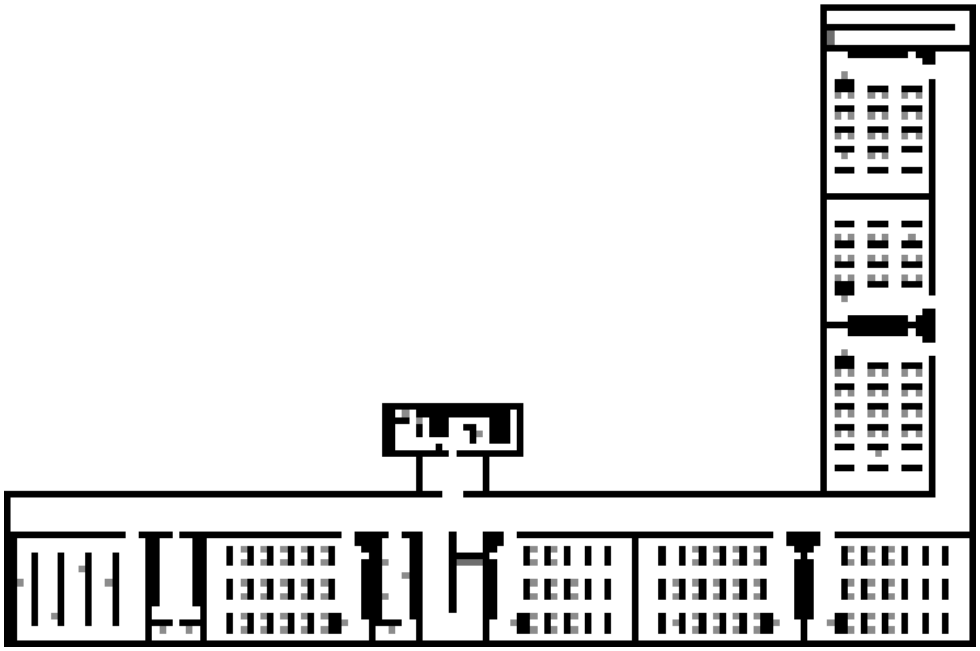


Fig. 6: **First floor layout used in MATLAB**

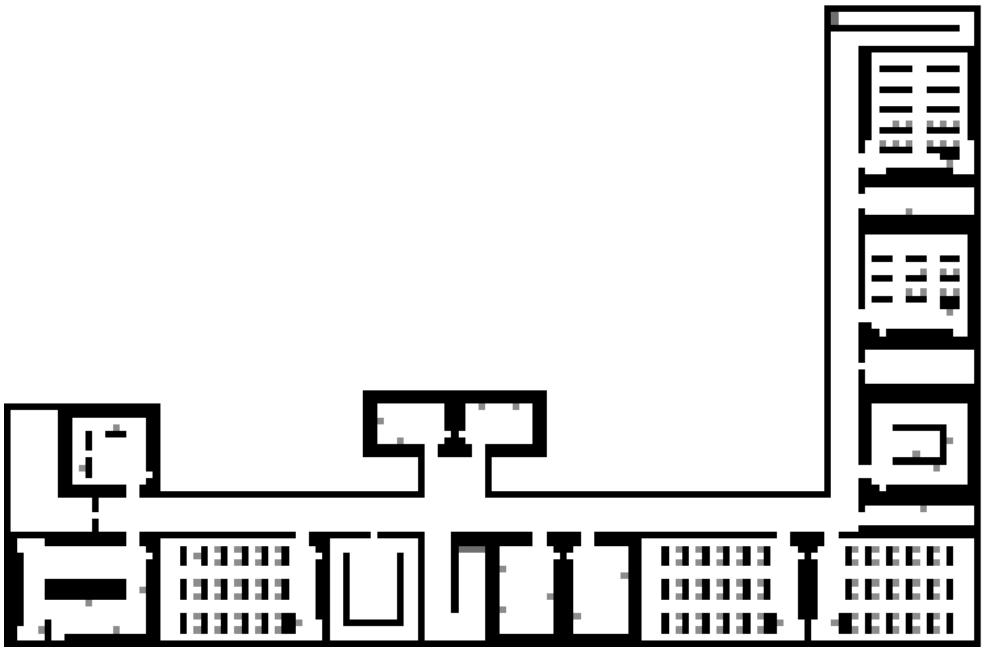


Fig. 7: **Second floor layout used in MATLAB**

3.2. Force Map

After reading in the pictures the Fast Marching toolbox was used. It generates force maps in X and Y direction. What these maps look like we can see in figure 8. Those maps will be later used for navigation of agents.

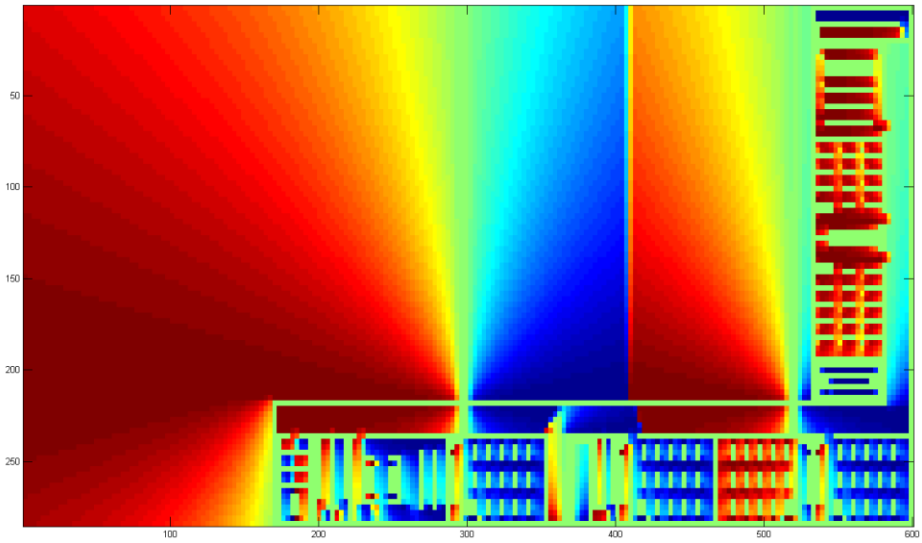


Fig. 8: **Force map in X direction for the ground floor**

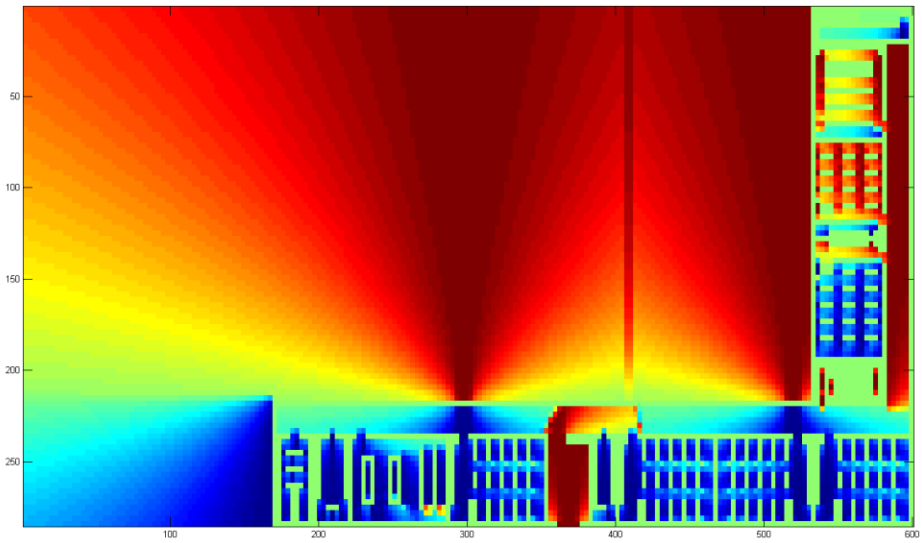


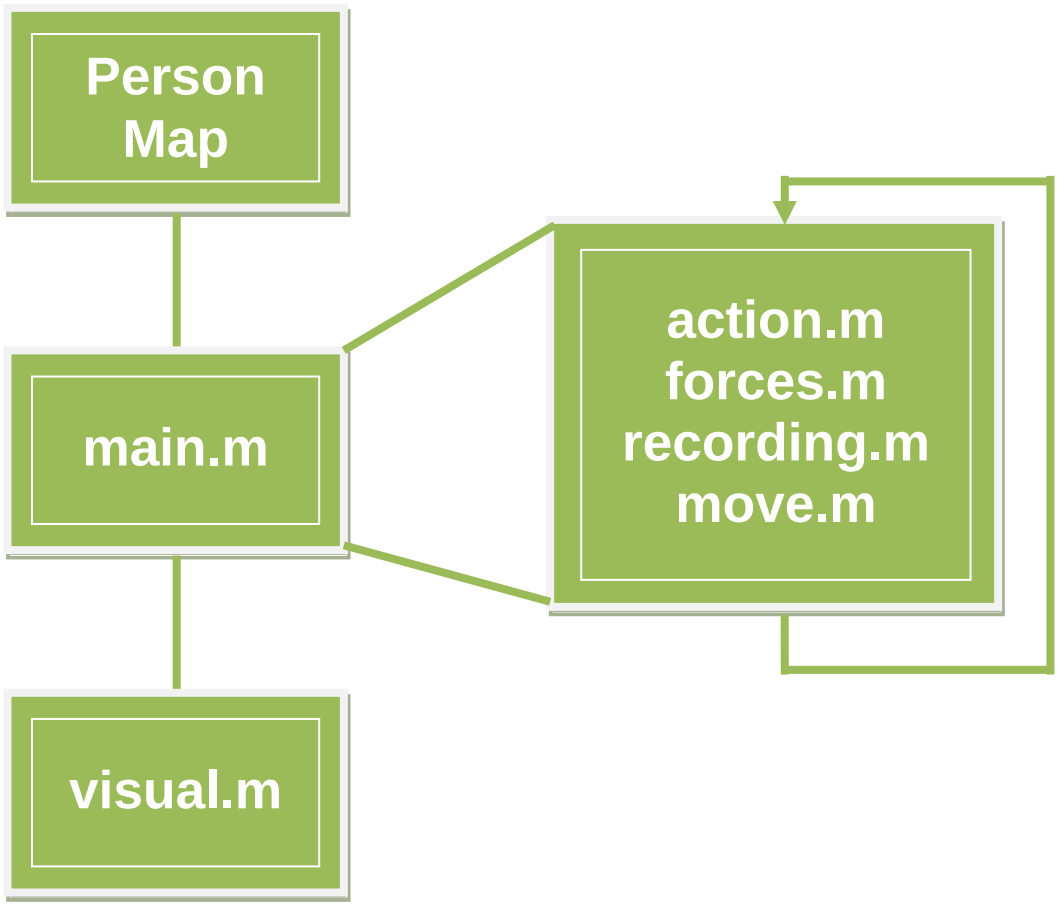
Fig. 9: **Force map in Y direction for the ground floor**

The total number of students is 380 and the number of teachers is around 35. We have to take also additional staff and at least several people in each room. If we sum it up we end up with number of agents of 469.

Class	**Number of Students**
Príma	21
Kvarta	17
Kvinta	20
Sexta	24
Septima	29
Oktáva	16
I.A	30
I.B	30
II.A	25
II.B	24
III.A	29
III.B	20
III.C	20
IV.A	27
IV.B	30
IV.C	18
Teachers	35
Additional	55
Total	**469**

Fig. 10: **The number of students attending each class** with other agents [3]

4. The Program



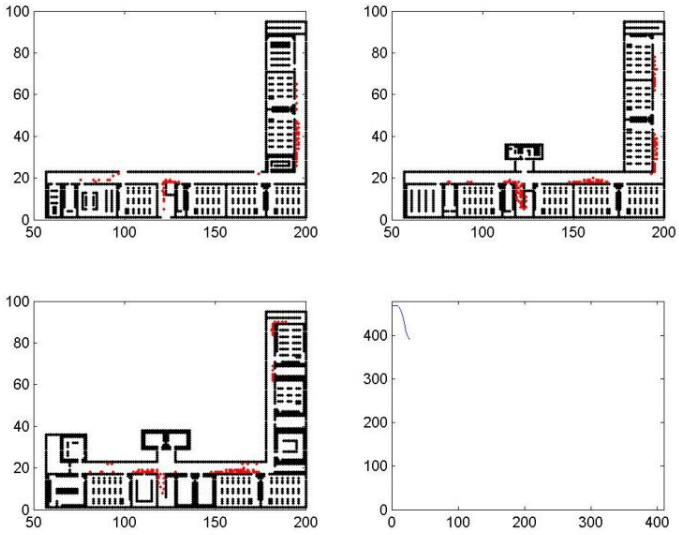


Fig. 11: **The flowchart of the MATLAB code**

As the previous flowchart shows, the inputs for our simulation are the map and the positions of the agents. They were both defined in previous steps. Those inputs are used by the main function that starts another 4 functions inside a loop:

- **action.m**
- **forces.m**
- **recording.m**
- **move.m**

Each function will be described in own section. After the simulation is done the visual.m function runs and plots the persons and map for each iteration. These frames are saved and using another program (Quicktime) we had created the movie with suitable frame frequency so that we can see the movement of agents.

4.1. Inputs

In this section we would like to explain how the inputs are used in the code. It is necessary because in the code structs are used. The idea behind this is that if the system is quite complex we can easily get lost in the names of our variables and the program gets hard to understand. The structs work the way that variables can be arranged together to another one higher in the hierarchy. It works on the same principle as the subsystems charts. In our code these are:

- **person.x** - the positions of agents in X axis
- **person.y** - the positions of agents in Y axis
- **person.level** - gives information on which floor the agents are
- **person.force_x** - the sum of the forces acting on agents in X direction
- **person.force_y** - the sum of the forces acting on agents in Y direction

- **map(floor).wall** - position of walls
- **map(floor).force_x** - force in X dir. acting on agents due to force map
- **map(floor).force_y** - force in Y dir. acting on agents due to force map
- **map(floor).action** - changing floor / taking out of the system / do nothing

4.2. action.m

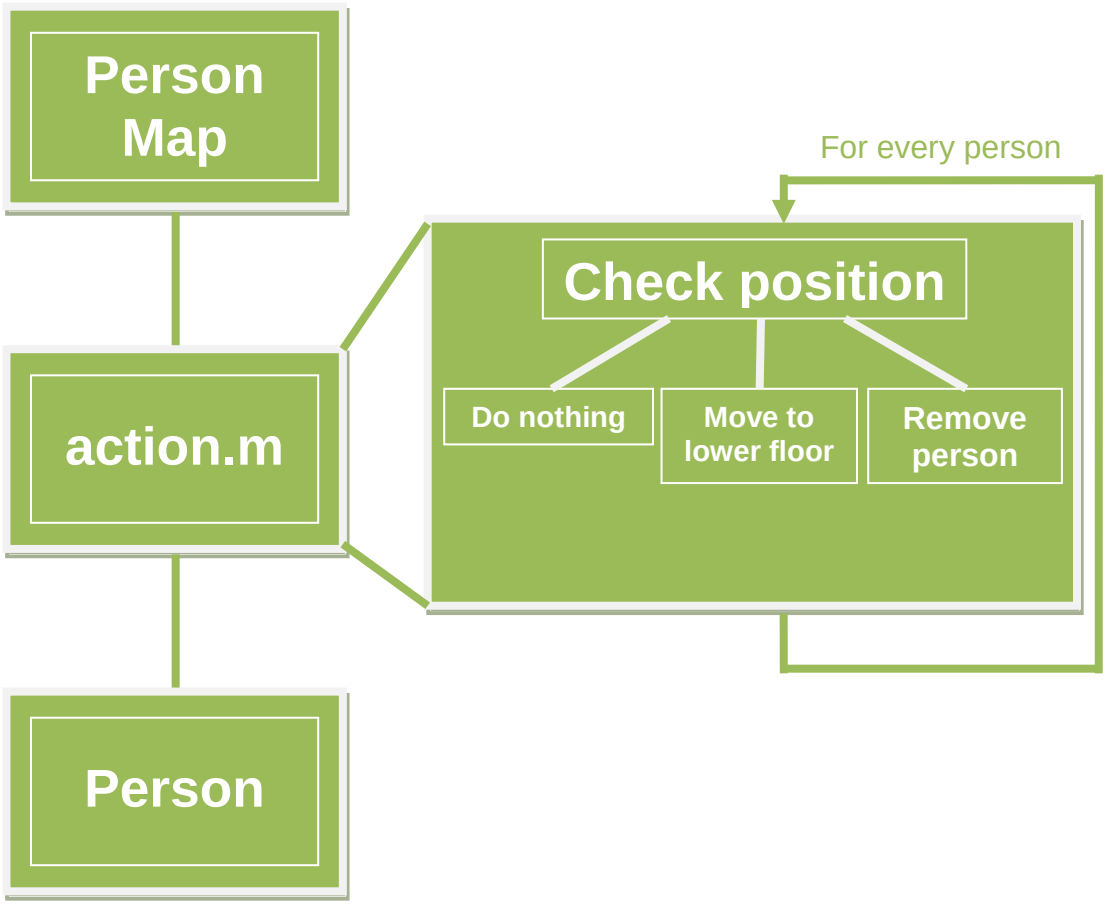


Fig. 12: **The flowchart of the action.m function**

The function action.m checks the position of all agents. If the agent is standing in the point where an exit is, it removes the agent from the system. If the agent is standing on the place where the stairs are, it moves agent to the lower floor. If the agent is on any other place it does nothing.

This function is very useful for moving the agents to other floors and also for removing of agents from the system. It is initialized before the move.m because when the agent is taken out in this step in the end of iteration another agent can get to the exit. Its output is position of all agents updated by the fact that the agents standing on the stairs are moved down to lower floor and agents standing on the exit are removed from the system.

4.3. Forces.m

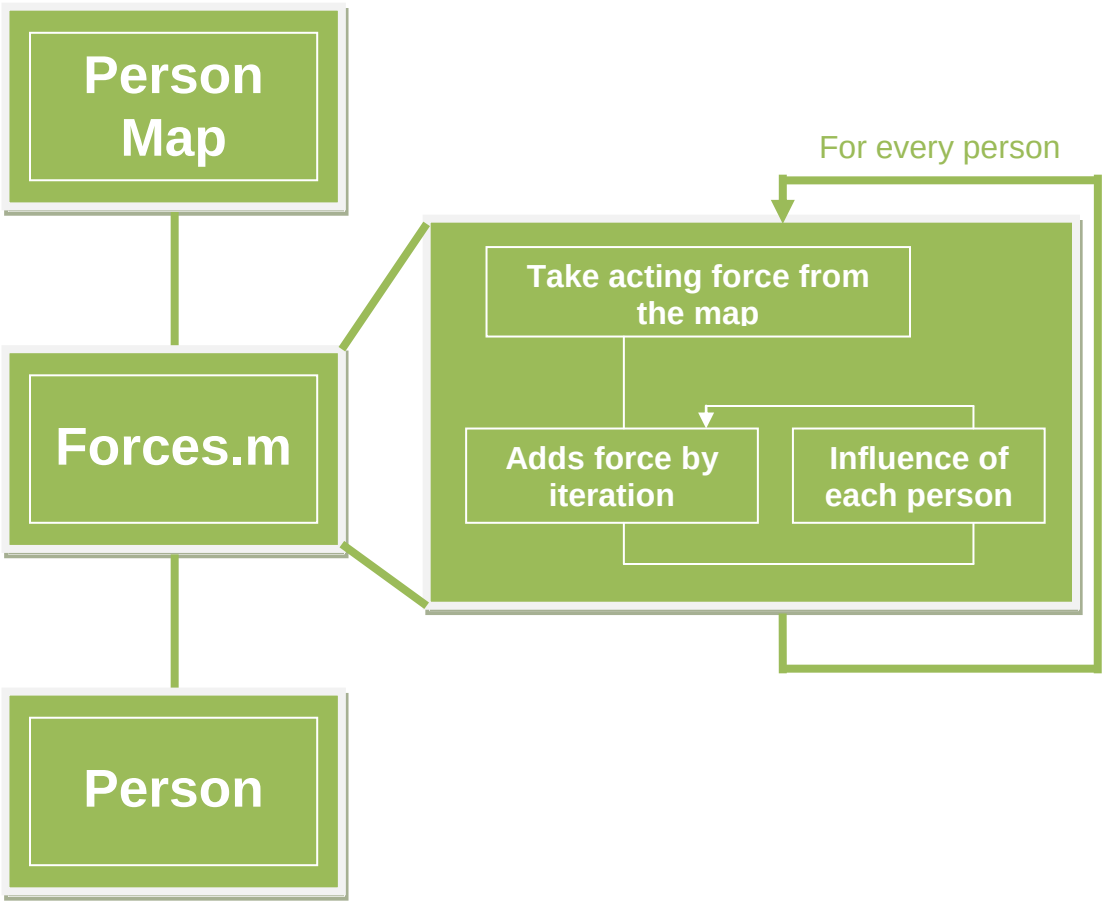


Fig. 13: **The flowchart of the Forces.m function**

In the Forces.m function the forces acting on the agents are calculated. First the force from the map is copied to the force acting on the person. Then the interaction with other agents is added. The agents are influencing each other in the way that as closer they are as higher the repelling force is.

This area around each agent which determines how he acts against other agent is in our simulation a circle. The more other agent gets to the center of the circle the bigger the acting force against this motion. In this function these forces are calculated the way, that for each agent is checked where are the other agents and how close they are. This procedure is then repeated for all agents so the outputs are new vectors of Person.force_x and Person.force_y.

4.4. recording.m

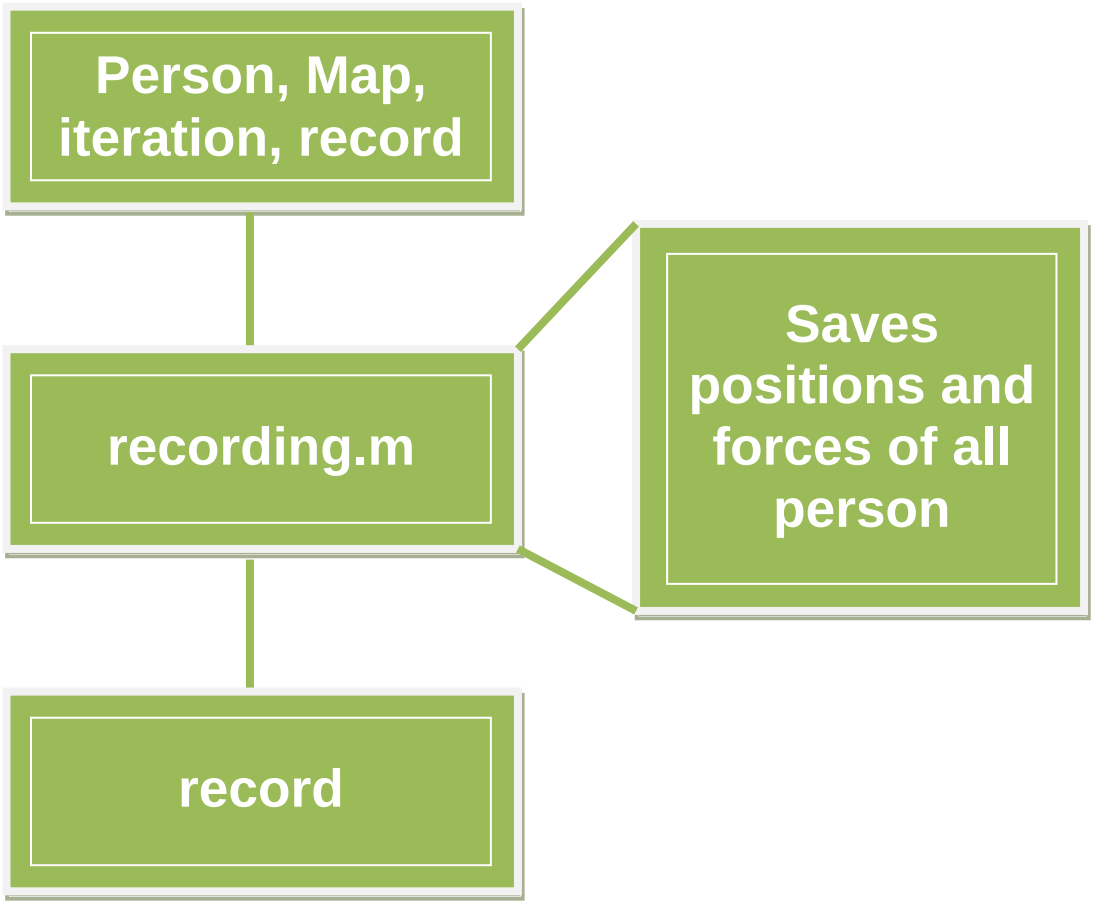


Fig. 14: **The flowchart of recording.m function**

What the recording.m function does is that it just saves positions and forces acting on each agent in each iteration and it enables us, in the end when the simulation is done, to see how the agents were moving, which means how they were changing their position iteration by iteration.

It also provides us data like how many agents are in a given iteration in the building, how many on which floor and by the estimation of simulated vs. real walking speed, it can provide us the time function of all these parameters.

In the end of this report when describing the results the number of agents remaining in the building over time will be much more important than the video showing how the agents were moving.

4.5. move.m

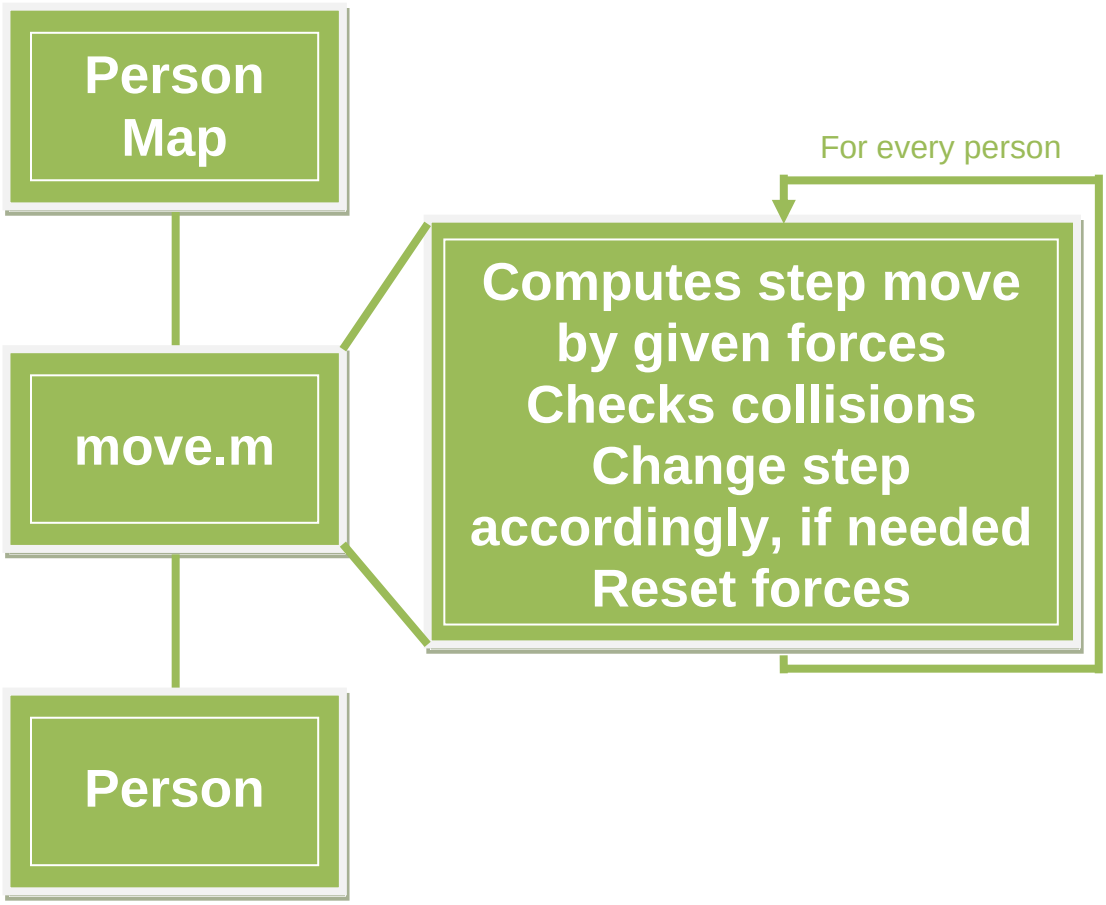


Fig. 15: **The flowchart of move.m function**

In the move.m function agents are finally moving towards the exit. The force given in Forces.m is now taken and recalculated into the distance the agent will move in x and y direction for iteration.

This function also checks for the collisions that may occur and changes the step accordingly. It resets the Person.force_x and Person.force_y and the new positions of agents are used as an input for new iteration.

It is the last function in the cycle and by the end of this iteration we have replaced the old positions of agents by new position (some have also changed the floor or are excluded from the system if standing in a exit). By iterating the agents are always moving towards the exits. Positions and forces for all agents are saved for all iterations.

4.6. visual.m

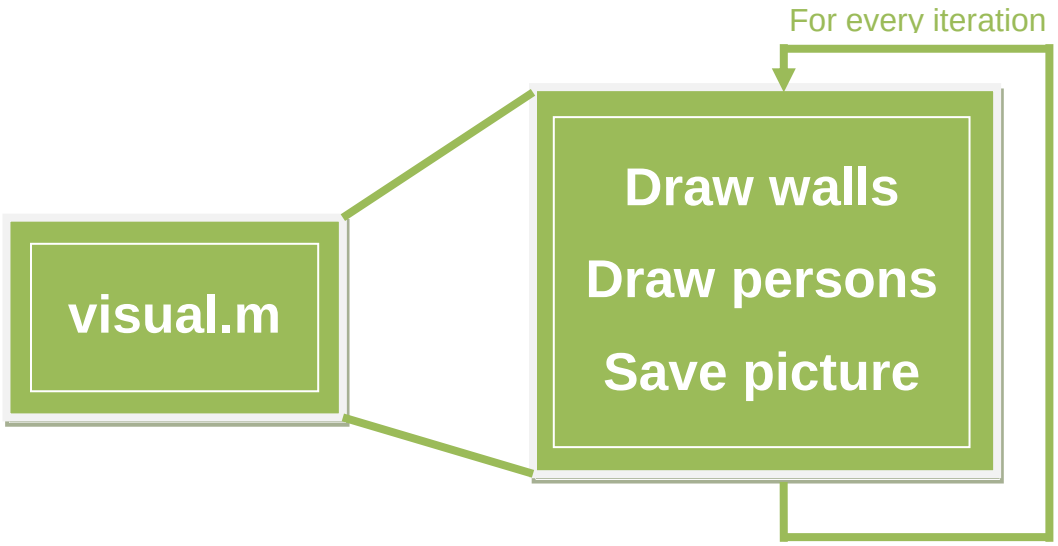


Fig. 16: **The flowchart of visual.m function**

The visual.m function draws the simulation. We were using command plot that draws the figure where black circles are walls and red circles are agents. One picture is drawn per iteration. Pictures were saved and the video was done. What such a picture looks like we can see in figure 17.

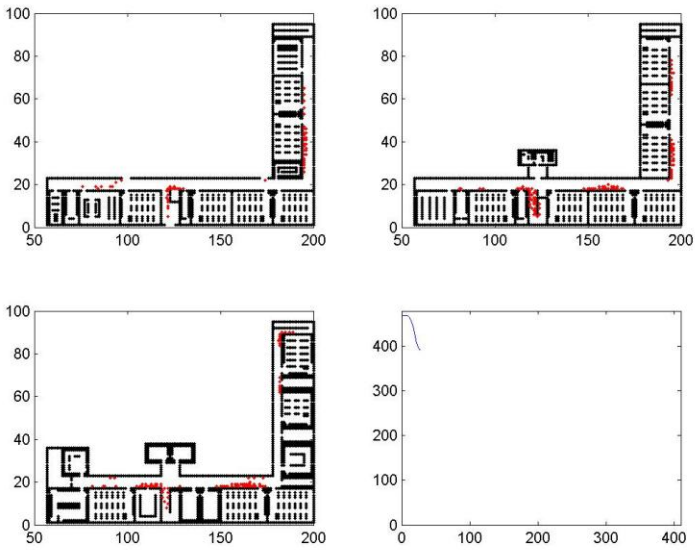


Fig. 17: **The picture of second floor for 27th iteration**

5. Results

In our simulation we were trying to get to a real evacuation bottleneck problem. It was at least Tomas feeling that in this building such a problem may occur.

Therefore results are quite surprising but in next sentences we will try to explain why the evacuation bottleneck problem did not occurred.

5.1. Time of evacuation

The simulation was done in iteration domain and therefore we have to first transform the number of iterations to seconds. We could use methods for transforming pixels to meters but our simulation was done more intuitive way. We were watching the video and from our experiences we were adjusting parameters for forces, steps, etc. Therefore we have found out that for a video that looks reasonable 3 iterations equals to 1 second.

5.2. Tactic for agents on the second floor

The persons on the last floor may get the feeling that evacuating through the fire stairs is a good idea even if the distance from the other side of the floor is more than three times longer. They may guess that surely all the exits will be stuck and if we account they are the only floor that is evacuating through the fire stairs this tactic may sound good.

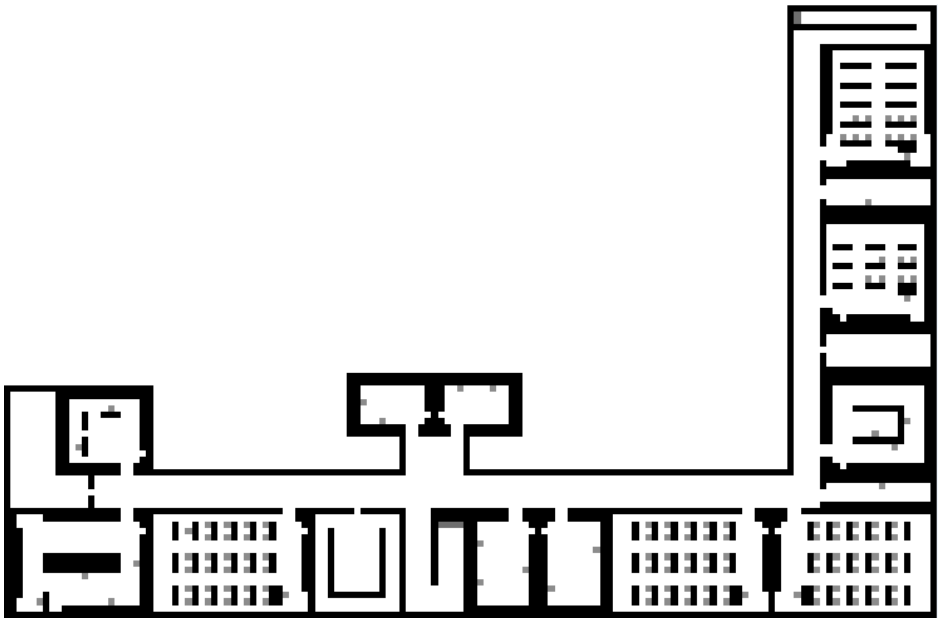


Fig. 18: **The layout of the second floor**

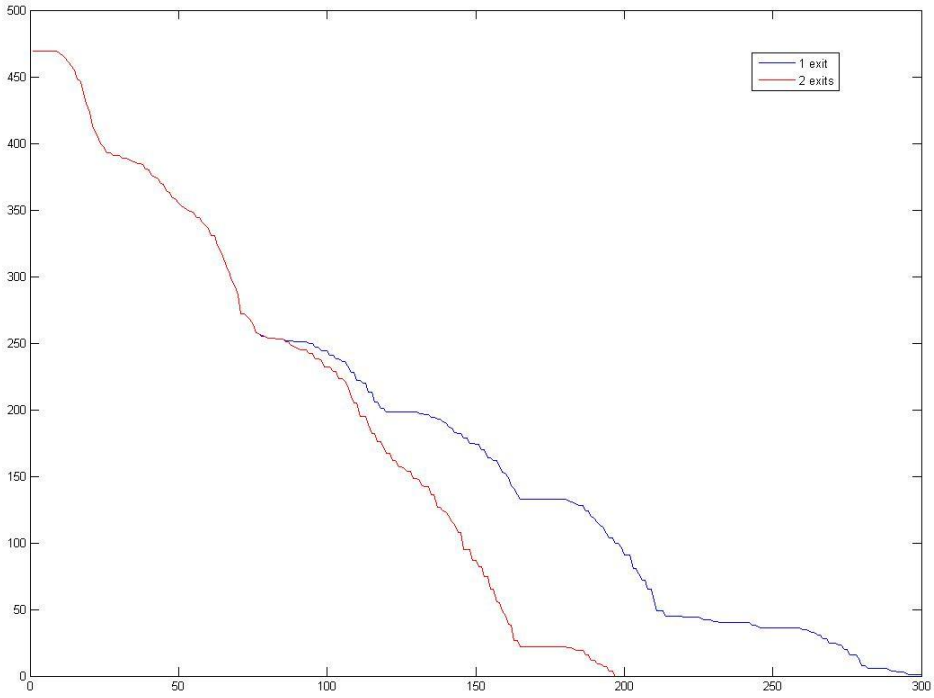


Fig. 19: **Comparison of evacuation of the second floor**

From figure 19 we can see that this tactic is not the right one. By evacuating through second floor the last person exits the building in 68 seconds. If the whole second floor will take only the fire stairs the evacuation time for the last person will be 100 seconds which makes the difference of more than half a minute.

5.3. Evacuation time

As was already stated the final evacuation time is 68 seconds. This value may sound a little bit confusing because is it possible to evacuate 470 persons in nearly one minute from the three floor building?

The answer is why not. The fact that is speaking for this value is that the exits are really large, because up to 4 persons can go together through the main exit. The corridors leading to these exits are wide enough for 5 persons. Therefore if we account that this is not a mock evacuation, where especially students are moving really slow but a critical situation we can see from the video that the bottleneck problem will not occur.

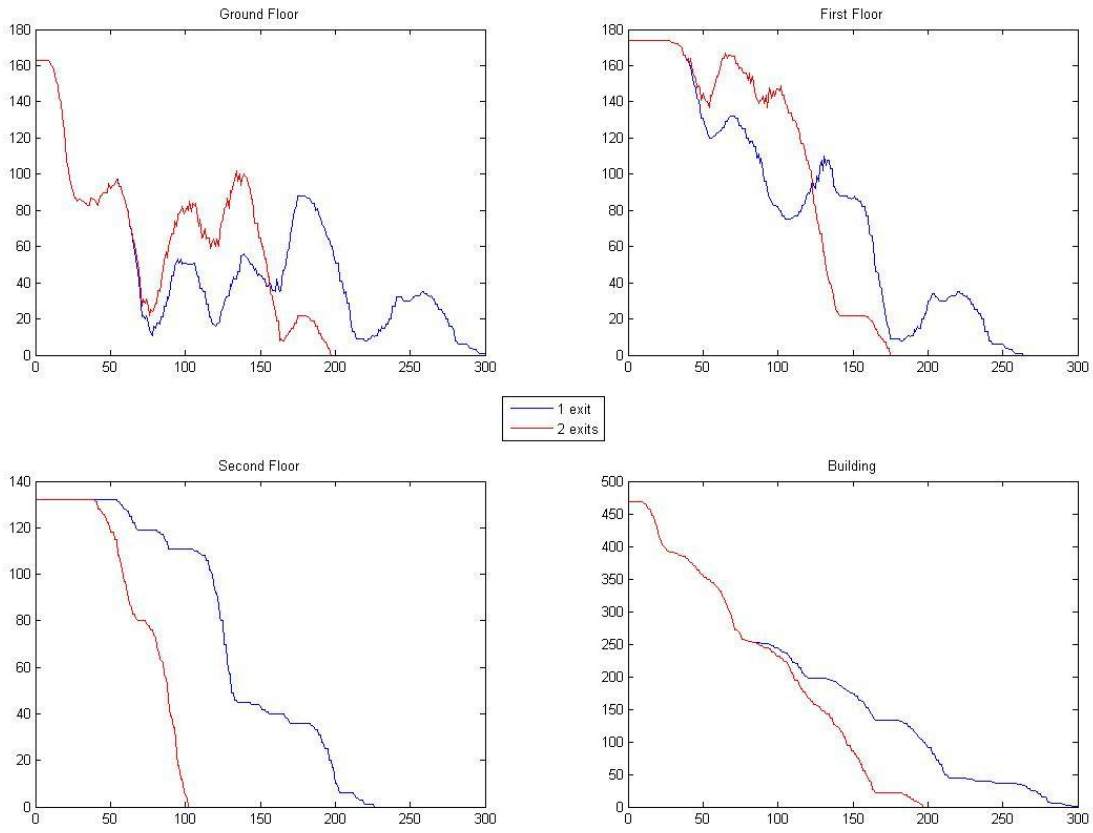


Fig. 20: **Number of persons remaining in the building and in each floor**

In figure 20 we can see the development of the number of persons remaining in the building and for each floor. The red curve is the tactic where also in the second floor agents are trying to get to the nearest exit.

We can see that from the second floor all persons are evacuated just in 35 seconds. This value sounds a little bit impressing but if we take that there is a real danger and we neglect little delays due to organization of the students by teacher this value starts to sound reasonable.

6. Conclusion

In this report we show that even with very simple mechanics an evacuation can be simulated and different situations can be compared. We were also able to make some statements on the basis of the simulation. Those statements partly did not fulfill our expectations but were rather surprising.

This simulation may not seem very reality-like with its discrete force-maps and movement, but the spacing is quite small, so it can approximate the continuous reality very well. The main advantage is the speed of the simulation and the visualization. This enables one the quickly check the results and change parameter. The only slow part is producing the necessary maps, but this cannot be automated and has always to be done manually.

For our case of the evacuation of a Slovak Highschool our assumptions still hold, because the students have to strictly follow the rules for the evacuation and thus they are not acting autonomous and the rules enforce the principle of minimal distance and not the minimum waiting time.

As we are not students in informatics or social science we tried to avoid complicated matters and programming that need a lot of background in these fields. Most of the code was written by ourselves and that is why we are very proud of our work that is easy to understand.

7. References

[1] http://www.gljs.sk/gljs60/priloha.html

[2] http://www.gljs.sk/wp3/?page_id=281

[3] Heer P., Bühler L.: Pedestrian Dynamics, Airplane Evacuation Simulation

8. Appendix

```
function [record] = main(map,person,max_iter)
% %main
% load Personputter/personsLARGE
% load Maps/Building(1exit)LARGE

%max_iter = 800;
record = recordinit(max_iter,person);


for iter=1:max_iter
    person = action(person,map);
    person = Forces(person,map);
    record = recording(person,iter,record);
    person = move(person,map);
end
end
```

```
function [record] = recordinit(max_iter,person)

l = length(person.x);
record.time_x = zeros(max_iter,l);
record.time_y = zeros(max_iter,l);
record.time_floor = zeros(max_iter,l);
record.time_force_x = zeros(max_iter,l);
record.time_force_y = zeros(max_iter,l);

end
```

```
function person = action(person,map)
%takes persons out of the map, if they reach the exit
%moves them to another floor
%actions are defined by map.action
%1 means exit
%2 means change floor (to lower floor)
new_person = person;

for i=1:length(person.x)
    x = person.x(i);
    y = person.y(i);
    if map(person.level(i)).action(y,x) == 2   %change floor (put those cells generously around the
stairs)
        ind1 = find(person.level == person.level(i)-1); %take all persons on the lower floor
        indx = find(person.x(ind1) == person.x(i)); %take all indices with the same x coords of ind1
        indy = find(person.y(ind1) == person.y(i)); %take all indices with the same y coords of ind1
        ind = intersect(indx,indy); %ind gives you the index of the person on the lower floor
        %with the same coords as the current person
        if isempty(ind)
            new_person.level(i) = person.level(i)-1; %change  floor if there is no person blocking
        end
    end
end
```

```
function [person] = Forces(person,map)

a_map = 0.3;  %for LARGE 6, for normal 1
a_pers = .1; %force parameter , for LARGE 1.5, for normal 0.75

for i=1:length(person.x)
    floor = person.level(i);

    %force by precomputed forcefield
    person.force_x(i)=map(floor).force_x(person.y(i),person.x(i));
    person.force_y(i)=map(floor).force_y(person.y(i),person.x(i));



    %force by other persons
    for k=1:length(person.x)
        if i~=k && person.level(i)==person.level(k)  %not itself, and only persons on the same floor
            deltax = person.x(i)-person.x(k);
            deltay = person.y(i)-person.y(k);
            dist = (deltax^2+deltay^2);
            if dist == 0
               dist = 1;
            end
            force = a_pers/dist;
            person.force_x(i) = person.force_x(i) + force*deltax/dist;
            person.force_y(i) = person.force_y(i) + force*deltay/dist;
        end
    end

end
```

```
function [record] = recording(person,iter,record)
l = length(person.x);

record.time_x(iter,1:l) = person.x;
record.time_y(iter,1:l) = person.y;
record.time_floor(iter,1:l) = person.level;
record.time_force_x(iter,1:l) = person.force_x;
record.time_force_y(iter,1:l) = person.force_y;
end
```

```
function [person]=move(person,map)
[M N] = size(map(1).wall); %every map has the same size


for i=1:length(person.x)
x_step = int32(person.force_x(i));
y_step = int32(person.force_y(i));

x_new = person.x(i) + x_step;
y_new = person.y(i) + y_step;
if x_new < N && x_new > 0 && y_new < M && y_new > 0 %making sure it is inside the map
%making sure it isnt in the wall
x_new1 = x_new;
y_new1 = y_new;
    if map(person.level(i)).wall(y_new,x_new) > 0
      if map(person.level(i)).wall(y_new,person.x(i)) == 0
          x_new1 = person.x(i);
          y_new1 = y_new;
      elseif map(person.level(i)).wall(person.y(i),x_new) == 0
          x_new1 = x_new;
          y_new1 = person.y(i);
      end
    end

    person.x(i) = x_new1;
    person.y(i) = y_new1;
end
    %reset the forces
    person.force_x(i)=0;
    person.force_y(i)=0;

end

end
```

```
function visual(map,record,floor,nuller)
%visualizing stuff
figure(1)
set(1,'visible','off')
[M,N] = size(map(floor).wall);
x = [];
y = [];
for k=1:M
    for l=1:N
        if map(floor).wall(k,l) > 0
            x = [x,l];
            y = [y,k];
        end
    end
end
[numiter numpers] = size(record.time_x);
for n = 1:numiter
    hold on
    %scatter(x,y,10,'k')
    %scatter(x,y,10,'k','filled')
    plot(x,y,'k.')
    for m=1:numpers
        if record.time_floor(n,m) == floor
            %hold on
            %scatter(record.time_x(n,m),record.time_y(n,m),5,'r')
            plot(record.time_x(n,m),record.time_y(n,m),'r.')
        end
    end

%    xlim([0 N]);
%    ylim([0 M]);

    %pause(0.01);
    %waitforbuttonpress();
    disp(n);

    %for saving the pictures
    filename = 'C:\Users\joehla\';  %on alex' mac book pro
    nuller = '10000';  %five letters
    number = strcat(nuller(1:end-length(num2str(n))),num2str(n));
    filename = strcat(filename,number);
    saveas(gcf,filename,'jpg');

    clf(gcf);
end

hold off
end
```

```
clear all
clc

[FileName,PathName] = uigetfile('*.bmp', 'Select a Bitmap File');
I=imread(strcat(PathName,FileName));

[a b] = size(I);
for i=1:a
   for j=1:b
      if I(i,j)<50
         I(i,j)=5;
      end
      if I(i,j)>200
         I(i,j)=0;
      end
      if I(i,j)==140
         I(i,j)=2;
      end
      if I(i,j)==115
         I(i,j)=1;
      end

   end

end

% I(300,255)=2;
% I(200,150)=2;
% I(200,330)=2;

f = getFile_my(I);
[FX,FY]=computeGradientField1(f);



function [F] = getFile_my(I)
space=find(I==0);
exit=find(I==2);
passenger=find(I==1);
wall=find(I==5);
[n,m]=size(I);
F=zeros(n,m);
F(space)=1;
F(exit)=Inf;
F(passenger)=2;
F(wall)=0;
F=flipud(F);
end
```

```
%Personputter
numfloors = input('How many floors are there?');
if numfloors < 1
   error('incorrect input')
end
person.x = [];
person.y = [];
person.level = [];
person.force_x = [];
person.force_y = [];

for k=1:numfloors
selection = 'Please Select floor number: ';
disp(strcat(selection,num2str(k)));
[FileName,PathName] = uigetfile('*.bmp', 'Select the correct Bitmap File');
I=imread(strcat(PathName,FileName));

[y x] = find(I==140);    %May be changed to other value if necessary
person.x = [person.x,x'];
person.y = [person.y,y'];
person.level = [person.level,k*ones(1,length(x'))];
end
person.force_x = zeros(1,length(person.x));
person.force_y = zeros(1,length(person.x));

clear numfloors selection FileName PathName I x y k
```

MATLAB FS11 – Research Plan
Evacuation of a School

Document Version: 3
Group Name: Swiss-Slovak misunderstanding
Group participants' names: Alexander Jöhl, Tomáš Nyitray

General Introduction

We would like to compare the evacuation time for renovated building and old one. Each second matters because we want to save lives. There were cases of people dying in old buildings because they were not able to evacuate quickly enough. As a reaction to this problem, the government issued new safety regulations. The consequences are visible as new fire stairs appearing on older buildings.

The building considered for simulation is one of these cases. It is the building of Gymnázium Ľudovíta Jaroslava Šuleka Komárno, Slovakia. This building was renovated 5 years ago and also the new fire stairs were built to follow the new safety regulations.

Fundamental Questions

How does the renovation influence the evacuation time?
What is the better strategy to evacuate for students at the most upper floor? To take the fire stairs or the main stairs? How does the number of people in queue influence the question before?

The main interest lies in the time from the start of the evacuation until the last person is evacuated. We try to find a correlation between the number of people and the evacuation time and also between the fraction of people deciding for a certain strategy and the time.

Expected Results

The renovation does greatly decrease the evacuation time. The difference in time of the last person exiting through the main entrance and the person exiting the fire stairs is expected to be very small except for cases of extreme fractions of people deciding for one specific exit.

References

Helbing D., Johansson A.: Analytical Approach to Continuous and Intermittent Bottleneck Flows; DOI: 10.1103/PhysRevLett.97.168001

Helbing D., Johansson A.: Dynamics of crowd disasters: An empirical study; DOI: 10.1103/PhysRevE.75.046109

Helbing D., Farkas I., Vicsek T.: Simulating dynamical features of escape panic; NATURE |VOL 407 | 28 SEPTEMBER 2000

Heer P., Bühler L.: Pedestrian Dynamics: Airplane Evacuation Simulation

Biner D., Brun N.: Evacuation Bottleneck: Simulation and analysis of an evacuation of a lecture room with MATLAB

Research Methods

We are going to use an agent-based model with the agents moving in a continuous space, but with a discretized environment (Walls, Danger). The way to the exits will be modeled as a potential increasing to the exits. The agents will move to the higher potential, and they themselves have a negative potential to repulse each other. The potential can be time-dependent to model a moving danger, e.g. fire.

Other

http://www.gljs.sk/gljs60/priloha.html
(Photos of old and renovated building)

http://www.gljs.sk/wp3/?page_id=281
(Distribution of rooms and number of students in building)