Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project report:
crowd flow optimization

Flöry N., Omeradžić A., Zengerle T.

Zürich
May, 2012

Agreement for free-download

We hereby agree to make our source code for 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.

Flöry Nikolaus   Omeradžić Amir   Zengerle Thomas

Contents

1. Introduction and Motivations

Emergency evacuation is a very important topic in many fields of our daily lifes. We find examples from small scale evacuations of a building, up to to a panic scenario in a stadium, in demonstrations or parades. History has shown how dangerous such situations can get with many people getting injured or even killed by people pushing each other. So of course the main target of projects like this is to improve efficiency of evacuations and at the end to save human lifes.

There are many different models for simulating pedestrian dynamics. The most successful model is the so-called "social-force model" which is able to reproduce many effects seen in real life situations. In our project we have decided to use a model based on cellular automata. This type of model has its advantages in the rather easy implementation as it is discrete in space and time and provides good results even without using the social-force-model. However, these models are not able to reproduce all the collective effects observed empirically [1]. Anyway, our goal is not to reproduce all empirically observed effects but to see how the flow of pedestrians changes when placing obstacles into the route of evacuation.

We will especially consider dangerous situations taking place in a room with one exit. The main questions which we are going to clarify are the following: What exactly will happen to the moving crowd if there is an obstacle in their route? Will it have an influence onto the overall velocity of the evacuation? Will people get out of the room faster? Is there a special relation between the size of obstacles, the number and positioning of them relative to the exit?

1.1. Loveparade

Let us take a look at a specific example in real life that motivated our simulation:

The Love Parade was a popular and free access festival that was held annually in different cities of Germany. It is reported that between 200.000 and 1,4 million people were attending the festival throughout its history.

The Love Parade in July 2010 was held in Duisburg (North Rhine-Westphalia) and caused the death of 21 people and at least 510 injuries.

The main access to the area where the event was taking place led through a road tunnel (Figure 1), which is roughly 400 meters long. The flux of people was regulated at both ends of the tunnel and the entry was at the south end.

Figure 1: The tunnel at the "Karl-Lehr street" [2]

Very high people densities arised due to jamming effects and non efficient regulation of the flux and distribution of people. Those high densities led to severe physical injuries. [3]

Motivated by such events we try to simulate the movement of people through one exit while both the flux and the density are being

Figure 2: High people density at the entrance of the tunnel. [4]

measured.

1.2. Ski lift

Another personal experience that motivated our simulation is shown in the pictures below.

Figure 3: Ski-lift at Montafon, Vorarlberg, Austria.

We can see two queues on both sides and an obstacle between them. At this ski-lift there are six gates altogether. The obstacle is placed such that the flux at the gates is optimized. Another important aspect of the obstacle is that the density of people is reduced.

Figure 4: The obstacle at the lift.

The shape of the obstacle is similar to a drop and we know from physics that this kind of shape has a relatively low resistance to flow.

2. Description of the Model

For the realisation of our goals, we chose to use a cellular automata model, which means that we have a discretion of time, space and people moving through the simulated system. Characteristic for such an implementation is the exclusion principle ensuring that only one person is occupying one cell. We identified each cell with a surface area of 40 x 40 cm. It means that each person has a relatively small area to stand on which is important to simulate situations with high people densities. Another main principle of the concept is the motion only to neighboured cells.

The transition between cells has probabilistic dynamics. A person in the situation to decide gets probabilities for each possible transition. We have four main contributions for a transition

1. Desired direction of motion which is typically the direction towards the exit
2. The level of confusion simulating the disorientation of escaping people due to panic or optical disturbance; no confusion means that there is full knowledge about the possible paths
3. Reaction to geometrical obstacles, such as walls or obstacles of different shape
4. Reaction to people in proximate cells; in our model people are inevitable obstacles being equivalent to a part of a wall

The action is taking place on a static floor field. Statics in this case means that it is not directly affected by the density of people in the neighbourhood in terms of attraction towards the exit. A person affects the field only being a solid obstacle. The static field was chosen in order to measure the effects of the infrastructure (room shape or obstacles within the route) on the flux more precisely.

Another assumption of our simulation is that people are acting completely individually which means that there is no herding behaviour. Of course, in real life herding is a very important aspect of pedestrian dynamics.

3. Implementation

The implementation consists mainly of three parts which are the core of our simulation. We have the main simulation script in which all the exterior functions are called and where the actual iteration is taking place. The second part is the static floor field - a matrix ("the room") in which people are heading towards the exit. The third part is a function that does the transition for each person individually with a probabilistic approach.

3.1. Script and iteration

After the initialization of the static floor field ("potential"), people are being randomly placed within the room ensuring that there is no random overlapping. It is possible to choose an absolute number $pers$ or relative percentage (relative to the number of free cells) of individuals inside the room. Then, a matrix with dimensions $3 \times 4 \times pers$ containing all the information about the people is initialized. Each layer of the 3-dimensional matrix depicts one person and the information is stored as follows:

$$\begin{pmatrix} x & p_{11} & p_{12} & p_{13} \\ y & p_{21} & 0 & p_{23} \\ 0,1 & p_{31} & p_{32} & p_{33} \end{pmatrix}$$

The current coordinates are given by x and y, while the entry (3,1) of the matrix is "0", if the person is active and "1", if the person reached the exit during an iteration. The (3×3) - part (without the first column) contains the values of the potential around the person's position (x, y). These values are the starting points for the further computation of the movement and depict the person's desired direction of motion (attractive force of the exit).

If a person reaches the exit, it is removed from the $(3 \times 4 \times pers)$ - matrix and the total number of people is reduced by one for the sake of efficiency.

A problematic issue of calculating pedestrian dynamics lies in the type of iteration that is used. In reality people make their decisions simultaneously which makes it difficult to implement a completely accurate model of decision making. Especially if we consider that the computational simulation is being performed step by step. Thus, we try to minimize the possible "artefacts" of the iteration by letting the people choose randomly. This approach seems to be adequate, if we consider the time step of one iteration to be comparatively small.

We decided to implement two different types of random iterations and compare them:

1. Partially random: This method takes the number of active people $pers$ and generates an array with some random permutation of the integers. Then the iteration is performed over the array and this way it is ensured that each active person has the chance to move in every iteration step.

2. Totally random: This type is completely random and picks in each step an arbitrary active person to move. This means that some people may be more active than others which seems to be more natural in the way that some people tend to hesitate more than others.

The iteration is performed till the last person reaches the exit. Then the plots for evaluation are created with the data that has been gathered throughout the iteration.

3.2. The potential

The static floor field is realised by a physically motivated "potential" and it is being initialized at the very beginning of the script. The two functions creating the static floor field and placing obstacles are called potmat() and obst().

3.2.1. potmat()

The Matlab-function potmat() returns a matrix A with shape $m \times (n+2)$ which will be used as a basis for calculating the movement of the single human-beings in our model. The sense of this matrix is to let people know where the exit is and so in which direction they have to move. potmat() requires four input arguments m, n, o, q: The first two are the shape of the required matrix $(m \times n)$. The third one o is the shape of the "room" which is realized by setting all those matrix-elements to zero which should not be accessible to the persons in the model. The last input argument q is the required size of the exit.

The single matrix elements are filled up with information about it's distance to the exit. The calculation is performed by looking at every single element of the matrix and counting the specific Δm and Δn in regards to the position of the exit and applying following formula: $D(m,n) = \sqrt{(\Delta m)^2 + (\Delta n)^2}$. The non-accessible parts of the matrix are filled up with zeros.

3.2.2. potmat2()

This version of potmat works similarly like the the first one described above. There are two differences: The distance from the exit to the matrix element is additionally evaluated by the $exp()$ function. So the calculation is performed with $D(m,n) = \exp\left(\frac{-\sqrt{(\Delta m)^2+(\Delta n)^2}}{(n/d)}\right)$. Furthermore an extra parameter d is available which describes the speed

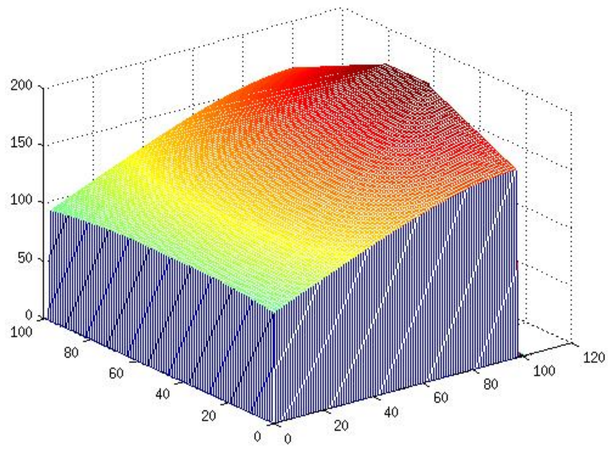

Figure 5: matrix returned by potmat(100,100,1,10). The height of the plot represents the value of the matrix element.

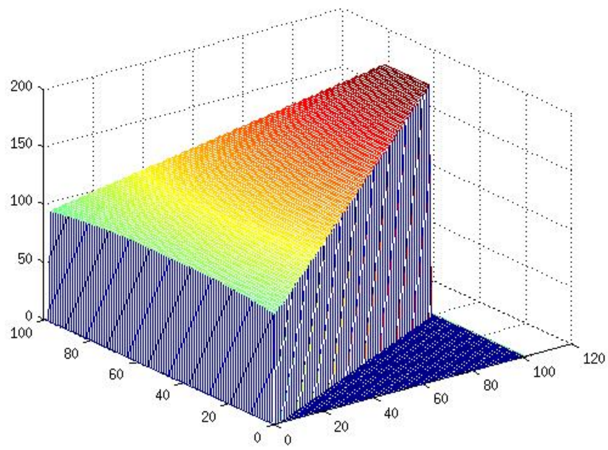

Figure 6: matrix returned by potmat(100,100,2,20). The hight of the plot represents the value of the matrix element.

of decay with distance from the exit. We decided to use potmat2() for our main simulations as it has been shown that a "potential"-field with exponential shape fits better to the fact, that people are heading more directly to the exit when getting closer to the exit (also see decision() and movev3()). As we know from physics, exponential functions very often exactly reproduce the physical processes in nature.

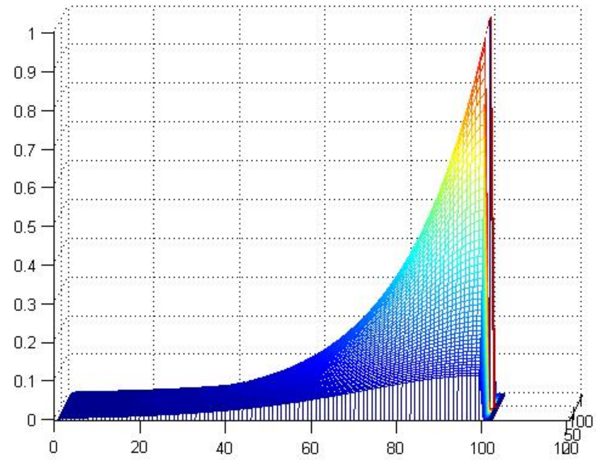

Figure 7: matrix returned by potmat2(100,100,1,10,5) - side view.

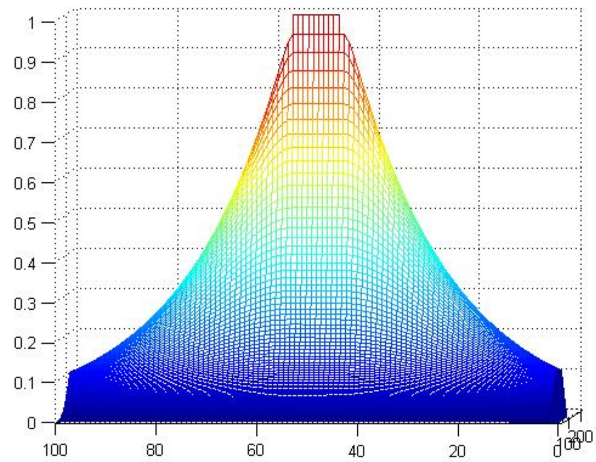

Figure 8: matrix returned by potmat2(100,100,1,10,5) - front view.

3.2.3. obst()

The function obst() was written in order to place obstacles into the matrices provided by potmat(). The input arguments for obst() are A, sm, sn, m, n. A is the matrix provided by potmat(), sm the width, sn the length of the obstacle and m, n determine the position where the obstacle should be placed to.

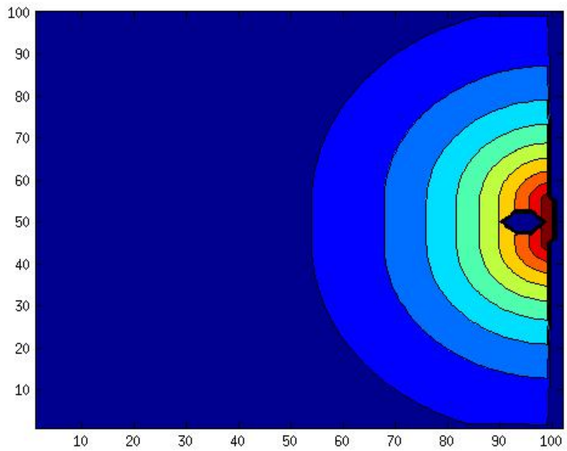

Figure 9: Contour-plot of an obstacle placed into a matrix A with function obst().

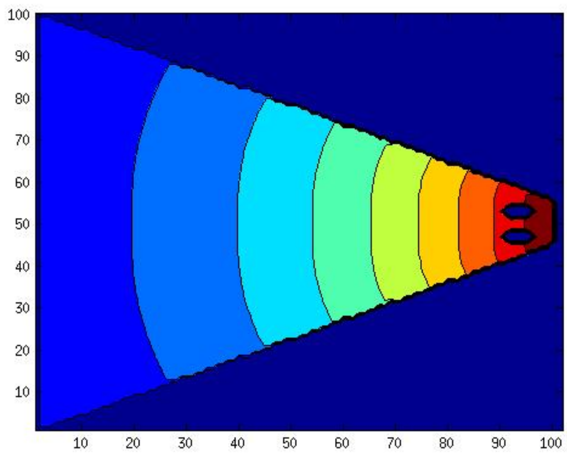

Figure 10: Contour-plot of two obstacles placed into a matrix A with function obst().

3.3. Transition - Movement

3.3.1. movev3()

As the name says, the function movev3() is the one, which actually performs the movement of a single individual, represented by a matrix. The function takes as an input a matrix A with the potential values as well as the coordinates, the length of the room n, and a *chaos* parameter which lies between 0 and 1. In our simulations, *chaos* represents the desired velocity, where 0 represents the highest and 1 represents the lowest possible desired velocity. After checking whether or not the person is still activated and has not reached the exit yet, it decides which coordinates the person gets next. This decision is based on finding the highest potential value in the places around the current position. If there is more than one highest value, we randomly select the next place. If there is a zero in an entry of the matrix, which means it is occupied by another person or wall/obstacle, it is not possible to move there. Another special case is that a person is surrounded just by walls and other people, which leads to no movement. The reason why we introduced a *chaos* parameter is that we do not want our individuals to act the same way each time we run our simulation, which would not be realistic. The influence of *chaos* to the decision is referred to in the function decision().

3.3.2. decision()

The function decision() manipulates the potential values of a person's Matrix. It therefore takes *chaos* as an input, which is a number between 0 and 1. Every single potential value in the matrix is multiplied by $(1 + sign \times x)$, where x is randomly selected in the range $[0, chaos]$ and $sign$ is +1 or -1, each with probability 0.5. Taking $chaos = 0.2$ for example means, that a value z lies somewhere between $[0.8 \times z; 1.2 \times z]$. After that the values are normed and given back to the function movev3(), which then can decide, which is the highest.

3.3.3. savepics()

This function is used to save an image of the current matrix including the activated people. It uses a $m \times n \times q$ matrix, q being the number of iterations, which is saved in the simulation script. These images are necessary to make a video out of our simulation and watch how people are moving across our room.

3.4. Evaluation of data

3.4.1. Measurement of the flux

The measurement of the flux is depending on the number of people that are passing through the exit per some number of iterations (sampling interval). This means that we check the number of people at the beginning of the interval and at the end. The difference is the flux per interval. The sampling interval can be chosen differently depending on the type of random iteration.

For the type "partially random" we use one iteration over the array of active people and then the flux is measured.

For the type "totally random" it is necessary to choose a static or dynamic (depending on the number of active people) sampling interval to measure the flux. This is due to the entirely random character of the iteration.

Let us consider one diagram:

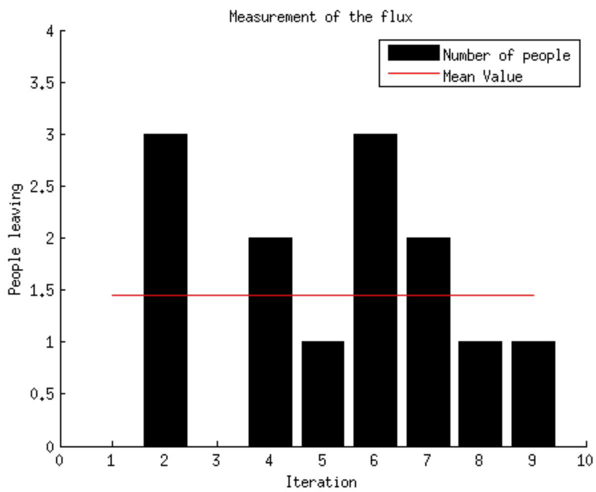

Figure 11: Flux through the exit for "partially random". Each bar tells how many people leave per iteration. The highest value of the y-axis (here "4") tells us the width of the exit.

For "partially random" the interval is one iteration step (all people move once). For "totally random" the interval is chosen separately. The mean value depicts the mean efficiency of the evacuation. The maximum value depends on the width of the exit and is the optimal value for "partially random“.

3.4.2. Measurement of the density

The Matlab-function dens() returns the density of humans in a specified area around the exit. This area has been chosen to be of quadratic shape with same width as the exit q. So overall, we have an area of $q \times q$. As input arguments dens() requires same information as potmat() and additionally the matrix A. So the values A, m, n, o, q are needed. dens() returns a relative number of persons within the specified area with a value between 0 and 1. In other word dens() returns the value: (number of persons currently within the specified area/total number of persons within the specific area when completely occupied).

4. Simulation Results and Discussion

All simulation results listed below are average values after 10 simulation runs. The deviation from the average value is given by the standard deviation $\sigma()$.

At the beginning of our simulation-runs, we decided to define standard parameters for better comparability. We used our script partiallyrandom.m with the following parameters:

- a (100×100) Matrix with an exit width of 10,
- based on a exponential potential matrix,
- a *chaos* parameter of 0.2,
- 0.2 as the starting density density of people, which approximately corresponds to 1 person per m^2 in reality.

4.1. Density

4.1.1. Variation of obstacle

When we started the first simulations one of the first things we could discover was the improvement concerning the density. Using our standard parameters the density in the area of the exit dropped when placing an obstacle in front of it. As defined in the explanation of the function dens(), we measure in the quadratic area with side length equal to the size of the exit, which is the area, where the density should be the highest.

Figure 12 shows the density during the simulation with respect to the iteration step. What we see is a big gradient at the beginning. This is simply because the people, who are randomly ordered, all start to walk towards the exit. At approximately step 25, we reach a level, where the room in front of the exit is full of people. There is a huge

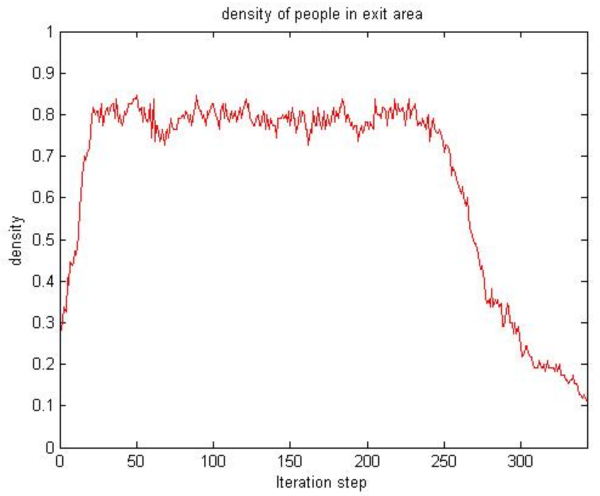

Figure 12: Without any obstacles placed near the exit.

pressure and force, resulting in a very high value. This value stays the same while individuals are leaving until at about step 250 the density goes down again because there are not that many people left to fill the space. As an average value for the density over the whole iteration we get 0.68. In the next situation we placed one single obstacle just in front of the exit.

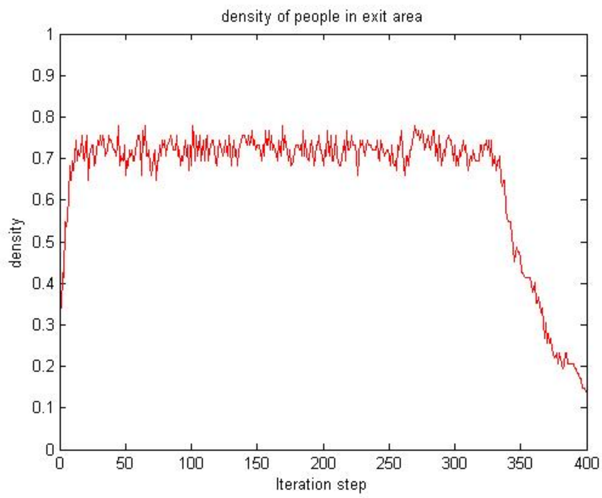

Figure 13: One obstacle was placed near the exit. Also see Figure 9 on page 16.

As we can see in Figure 13, the shape of the curve stays the same – again randomized people are getting to the exit at the beginning. But then, the values do not go up as high as before and we get an improved average value of 0.64. However, what we are also discovering is a slightly higher number of iterations needed to evacuate the same number of people as before. We will focus on this effect below. To reduce the density even more, we tried out to place another obstacle, so we now have two obstacles symmetrically placed in front of the exit.

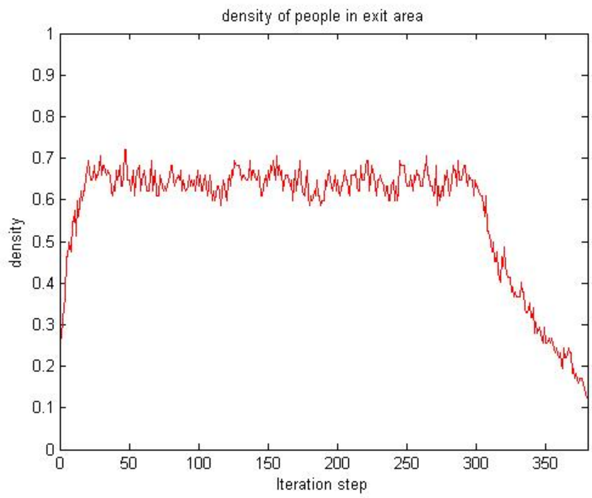

Figure 14: Two obstacles were placed near the exit. Also see Figure 10 on page 16.

Using this situation we got our best results, we can see that the level is much lower, with an average value of 0.56 over the whole iteration. What is more, even the number of iteration dropped compared to using one obstacle, and is just slightly higher than without any barrier. From the form of the curve we see that the average value over the whole iteration is not that significant, the mean value in the region between step 50 and 250 seems more interesting to us. Let us denote the density as ρ.

- No obstacle

$$\langle\rho\rangle_{tot} = 0.664 \qquad \sigma\left(\langle\rho\rangle_{tot}\right) = 0.012$$
$$\langle\rho\rangle_{50:250} = 0.789 \qquad \sigma\left(\langle\rho\rangle_{50:250}\right) = 0.004$$

- One central obstacle

$\langle\rho\rangle_{tot} = 0.653$ $\sigma\left(\langle\rho\rangle_{tot}\right) = 0.005$

$\langle\rho\rangle_{50:250} = 0.718$ $\sigma\left(\langle\rho\rangle_{50:250}\right) = 0.003$

- Two obstacles

$\langle\rho\rangle_{tot} = 0.577$ $\sigma\left(\langle\rho\rangle_{tot}\right) = 0.005$

$\langle\rho\rangle_{50:250} = 0.647$ $\sigma\left(\langle\rho\rangle_{50:250}\right) = 0.003$

4.1.2. Variation of the shape

In this section we will run our simulation using a conic shaped room. Apart from that we are using the same parameters so that we can compare with the above situation.

- Cone, no obstacle

$\langle\rho\rangle_{tot} = 0.732$ $\sigma\left(\langle\rho\rangle_{tot}\right) = 0.003$

$\langle\rho\rangle_{50:250} = 0.768$ $\sigma\left(\langle\rho\rangle_{50:250}\right) = 0.002$

- Cone, one central obstacle

$\langle\rho\rangle_{tot} = 0.660$ $\sigma\left(\langle\rho\rangle_{tot}\right) = 0.002$

$\langle\rho\rangle_{50:250} = 0.677$ $\sigma\left(\langle\rho\rangle_{50:250}\right) = 0.003$

- Cone, two obstacles

$\langle\rho\rangle_{tot} = 0.578$ $\sigma\left(\langle\rho\rangle_{tot}\right) = 0.003$

$\langle\rho\rangle_{50:250} = 0.594$ $\sigma\left(\langle\rho\rangle_{50:250}\right) = 0.004$

Again we see a much better density using obstacles, especially when using two of them. The drawback is the slightly higher number of iterations needed.

From these results we can conclude that for both rectangular and conic the use of two obstacles gives the most satisfying results. What does this mean in reality? A smaller density, especially in front of the exit, simply means, that there are less people in the same area. So the pressure and force among and between the individuals is not as high and therefore one could expect fewer injuries or accidents.

4.2. Flux

Let us examine the flux of people in our simulation. We shall take a look at the change of flux depending on the change of different parameters while keeping the others fixed. This approach makes it easier to find the influence of one specific parameter on the whole flux. For the results we did 10 iterations. For every iteration we measured the mean flux which tells us something about the continuous flow of people over the time of evacuation. Effects such as arching and clogging at exits [5] can easier be seen by plotting the flux and looking for irregularities. We took "Partially random" as our standard random iteration to ensure that every person moves once per iteration. Thus, the maximal and optimal flow is determined by the width of the exit. An exit width of 10 means that at most 10 people are able to pass the exit per iteration.

4.2.1. Variation of obstacles

Firstly, we kept all the standard parameters and variated the obstacles which means that we did simulations without obstacles, with one obstacle at the center and with two obstacles at the edges of the exit (pictures in the obst() section). Our repeatedly performed simulations led to following results (q is the number of iterations and ϕ the flux).

- No obstacle

$\langle q \rangle = 335.8$	Iterations	$\sigma(\langle q \rangle) = 4.4$
$\langle \phi \rangle = 5.72$	People/Iterations	$\sigma(\langle \phi \rangle) = 0.07$

- One central obstacle

$\langle q \rangle = 398.5$ Iterations $\sigma(\langle q \rangle) = 5.0$
$\langle \phi \rangle = 4.82$ People/Iterations $\sigma(\langle \phi \rangle) = 0.06$

- Two obstacles

$\langle q \rangle = 372.8$ Iterations $\sigma(\langle q \rangle) = 5.3$
$\langle \phi \rangle = 5.15$ People/Iterations $\sigma(\langle \phi \rangle) = 0.07$

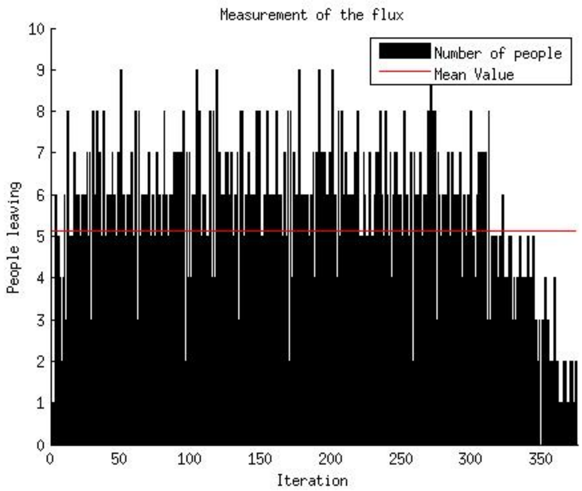

Figure 15: Here we have one exemplary flux for two obstacles plotted as bars. Due to the high density there is no continuous flow, but stop-and-go effects can be observed.

If we consider these results combined with the results of the previous section, then the last configuration with two obstacles seems to have made the most balanced results. We can see that the flux has the highest value for no obstacles, but it is clear that in this case we have very high people densities in front of the exit. As we expected, one central obstacle only reduces the density but flux and number of iterations (proportional to the time needed) are worse. For two obstacles the flux and the number of iterations is somewhere in between the other configurations and the density is considerably reduced.

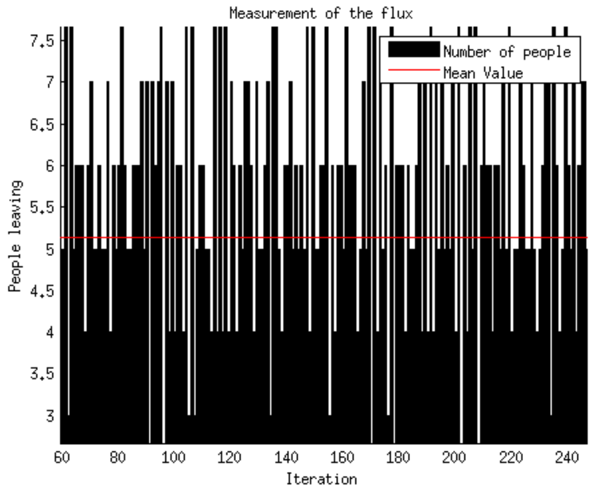

Figure 16: This is a smaller extract from the central part of the previous plot which shows more detail.

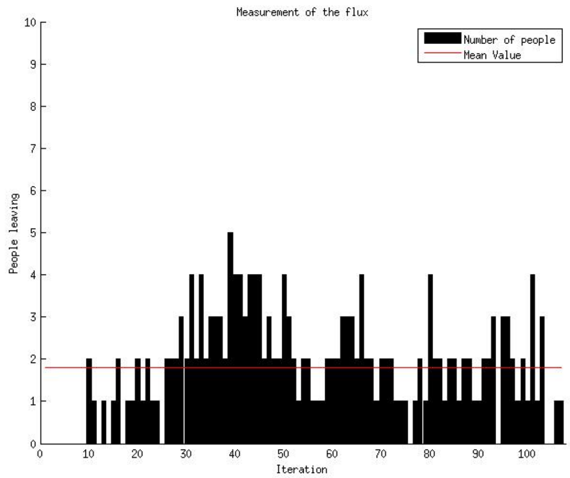

Figure 17: Comparing this to a starting density reduced by a factor of 10 and letting all parameters be the same, we get a more continuous flow. There is less arching and clogging.

4.2.2. Variation of the shape

Now we will have a look at the flux if the room of evacuation is cone-shaped and not rectangular. We use the same variation of obstacles as we did in the previous section. Thus, we can directly compare the flux in both shapes.

- Cone, no obstacle

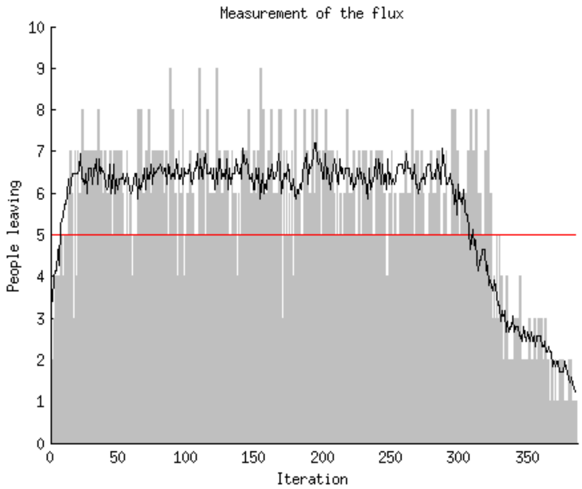

Figure 18: This plot shows the flux overlaid with the density function and the correlation between them.

$\langle q \rangle = 318.7$	Iterations	$\sigma(\langle q \rangle) = 1.2$
$\langle \phi \rangle = 6.03$	People/Iterations	$\sigma(\langle \phi \rangle) = 0.02$

- Cone, one central obstacle

$\langle q \rangle = 414.0$	Iterations	$\sigma(\langle q \rangle) = 2.5$
$\langle \phi \rangle = 4.63$	People/Iterations	$\sigma(\langle \phi \rangle) = 0.03$

- Cone, two obstacles

$\langle q \rangle = 387.0$	Iterations	$\sigma(\langle q \rangle) = 1.8$
$\langle \phi \rangle = 4.96$	People/Iterations	$\sigma(\langle \phi \rangle) = 0.02$

In the simulation we can see that “no obstacle” is clearly the best in terms of number of iterations and flux. But we have seen that the main disadvantage is the high density of people at the exit which is apparently due to the cone-shaped room.

The two simulations (rectangle and cone) with obstacles behave similarly relative to each other which means that two obstacles increase the flux and decrease the time needed for evacuation compared to one obstacle. But we can say that the difference between obstacle and no obstacle is generally wider for the cone. This observation fits the expectation (and daily observation) that a cone optimizes the flux very well.

4.3. Totally vs. partially random

We have also done simulations with a totally random approach but the results showed that the "partially random" approach was more adequate for our simulation. This is because the totally random iteration led to disproportionate behaviour. For example, people at the back were continuously moving while others at the exit were not leaving the room due to the randomness. Thus, "partially random" was the more appropriate choice because in panic situations we can assume that every person actually wants to leave the room.

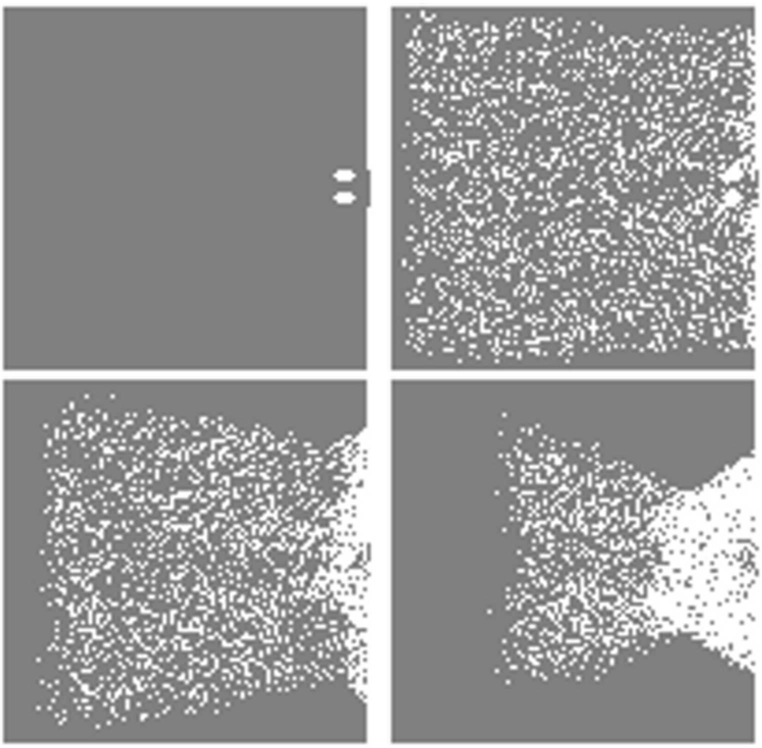

Figure 19: Simulation in action. Picture of an empty room with two obstacles and tree screenshots of a running simulation.

5. Summary and Outlook

Running several simulations, we showed that there certainly is a potential for making evacuations safer and more efficient by placing obstacles right in front of the exit. Especially the fact that it is possible to reduce the density of humans around the exit implies that less people will be injured. Furthermore we observed a not much longer duration of evacuation caused by the obstacle. Therefore we can conclude that a slight increase of the duration can result in a much safer evacuation.

If we compare the desired velocity (realised by the potential field and chaos parameter) with the actual velocity of the crowd, we can say that our simulation shows smaller evacuation times for higher desired velocities. This however is accompanied with higher densities and thus decreases the overall safety. Our simulations have also shown that the positioning of obstacles right at the edges of the exit give the most balanced results in terms of density and duration of evacuation.

Certainly there is a high potential using this principles, especially where time doesn't matter (i.e. at Ski Lifts) to reduce risk of injuries and to make the situation more comfortable for people. And if we consider events such as the loveparade, then we can conclude that it is very important to somehow regulate the flux of people in order to intelligently redistribute and reduce the people density, especially and key points such as entries or exits.

Obviously there are still investigations needed in regards to optimum size and shape of the obstacle as well as its optimal position. Furthermore, there is still more potential concerning the investigation of the correlation between the desired and realised velocities. Because of lack of time we decided to use some "standard"-parameters to do our simulations to get some results with reasonable effort. Finding the optimal shape, size and position of the obstacle for sure would go beyond the scope of this lecture.

All in all we can say that the simulation of pedestrian dynamics is a very important part of modern research and it will continue to do so because it is directly touching the everyday life of everyone of us.

References

[1] Schadschneider A., Kirchner A., Nishinari K.: CA Approach to Collective Phenomena in Pedestrian Dynamics

[2] Johann H. Addicks, addicks@gmx.net

[3] http://de.wikipedia.org/wiki/Unglück_bei_der_Loveparade_2010

[4] http://www.stern.de/panorama/jahrestag-loveparade-unglueck-gefangen-im-hexenkessel-1706033.html

[5] Dirk Helbing, Illés Farkas, Tamás Vicsek: Simulating dynamical features of escape panic

A. Matlab code

simulation_test_partially_random

```
%{
PARTIALLY RANDOM
skript where the simulation is taking place
%}

zaehler = 1;
it = 10;                    %number of simulations

iterationen = zeros(1,it);
fluss = zeros(1,it);
dichte = zeros(1,it);
dichte50 = zeros(1,it);

for i=1:it

% starting parameters
m = 100;          % size of matrix including walls (up and down)
n = 100;          % size of matrix including exit and walls left
endmove = n;     % for move-function
chaos = 0.2;     % for move-function, chaos parameter
ausgang = 10;    % width of the exit

% creating potential
P = potmat2(m,n,1,ausgang,10);
%P = obstneu(P,5,8,round(m/2),n-10);        % create one single obstacle
%P = obstneu(P,3,6,round(m/2-3),n-10);       % create two obstacles
%P = obstneu(P,3,6,round(m/2+3),n-10);       % at specific positions

% count occupied cells (obstacle), for density-function
besetzt = countobst(P,m,n,ausgang);

% determine resulting size
w = size(P);
m = w(1);
n = w(2);

% copy of potential, that will be used. people walk on P2.
P2 = P;

% making pictures
Bilder = zeros(m,n,[]);

%density or absolute value
pers=pdens(0.2,m-2,n-4);

%number of active people
new_pers=pers;
old_pers=pers;

%create 3x4xPers-Matrix
M2 = createMat(pers);
```

```

%initalize random coord. to matrix
M2 = init(M2,P,pers,m,n);

%person = obstacle
P2 = persObst(M2,P2,pers);

%assign potential to each (!) person by (x,y)
M2 = persPotAll(M2,P2,pers);


% initialize vectors to measure flux & density
pass = [];
density = [];
q = 1;          %iteration counter

while 1         %do, till last person reaches the exit

    old_act = new_pers;          % compute efficiency

    perm = randperm(new_pers);   % partially random iteration

    deleted = [];                % deleted people
    i = 1;                       % deleted counter

    for p = perm

        alt = M2(:,1,p);         % old coordinates

        M2 = persPot(M2,P2,p);   % assign potentials
        M2(:,:,p);

        v=movev3(M2(:,:,p),n-2,chaos,1);       % MOVE & new coordinates
        x=v(1);
        y=v(2);
        z=v(3);

        if z==0     %person moved & is still active

            %person=obstacle, restore original potential-value
            P2(alt(1),alt(2)) = P(alt(1),alt(2));
            P2(x,y) = 0;

            %update: new coordinates to matrix
            M2(1,1,p) = x;
            M2(2,1,p) = y;
            M2(3,1,p) = z;

        else        %person reached exit

            P2(alt(1),alt(2)) = P(alt(1),alt(2));
            deleted(i) = p;
            i = i+1;

        end

```

```

    end


    %delete old matrix elements
    M2(:,:,deleted) = [];                   % delete person
    new_pers = new_pers-numel(deleted);     % one person less
    Bilder(:,:,q) = P2;                     % make picture

    %density
    [a,b] = dens(P2,m,n,ausgang);
    densit = (b-besetzt)/(a-besetzt);
    density(q) = densit;

    %efficiency by computing difference of people
    new_act = new_pers;
    pass(q) = old_act-new_act;


    if new_pers==0  %breaking the while-loop, when all people at exit
        break
    else
        q = q+1;    %iteration counter
    end

end

step = linspace(1,q,q);          % number of iterations for plot
mittel = mean(pass);             % mean value of flux

%density = density*ausgang;      % scale density, only for plot

%{
% plotting
figure();
hold on
bar(step,pass, 'black')%,'EdgeColor',[0.75 0.75 0.75])
plot(step,ones(1,q)*mittel, '-r')
title('Measurement of the flux')
legend('Number of people','Mean value')
axis([0 q+1 0 ausgang])
xlabel('Iteration')
ylabel('People leaving')
hold off

%density = density*ausgang;      % scale density, only for plot
figure();
hold on
plot(step,density, '-b');
plot(step,ones(1,q)*mean(density), '-r')
axis([0 q 0 1])
title('Measurement of the density')
legend('Density of people','Mean value')
xlabel('Iteration')
ylabel('Density')
hold off
```

```
%}

% save data of current iteration
iterationen(zaehler)=q;
fluss(zaehler)=mean(pass);
dichte(zaehler)=mean(density);
dichte50(zaehler)=mean(density(50:250));

zaehler = zaehler + 1;

end

% show data after iteration
disp('Werte der Iterationen:')
iterationen
fluss
dichte
dichte50
disp('Mittelwert und Standardabweichung: Iterationen')
mean(iterationen)
std(iterationen)
disp('Mittelwert und Standardabweichung: Fluss')
mean(fluss)
std(fluss)
disp('Mittelwert und Standardabweichungen: Dichte')
mean(dichte)
std(dichte)
disp('Mittelwerte und Standardabweichungen: Dichte_50_250')
mean(dichte50)
std(dichte50)
```

movev3()

```
function v = movev3(A,n,chaos,velo)

B = A(:,2:4);    %just potential values


v = A(1:3)';     %coordinates

%desired velocity als wertung vor und r ck wertung
%B(1,1) = B(1,1)*

%kann nicht r ckw rts gehen
%if (v(2)< 2*n/3)
 %   B(1,1) = 0;
 %   B(2,1) = 0;
   % B(3,1) = 0;
%end

%desired velocity: 3 gr sste werte werden mit velo multipliziert, 3
%kleinste durch 3 dividiert
%bkoord = biggest(B);
%kkoord = smallest(B):
```

```

%vorher = v
% x(3) ist null(aktiviert) oder eins(deaktiviert)
if (v(3) == 1)
    v = v;

else
    if (v(2)==n) %steht in ffnung, wird eingemauert und deaktiviert
        v(3) = 1;
        v(1) = 2;
        v(2) = n;
    else

        %"entscheidungsmatrix" austellen
        B = decision(B,chaos,v(2),n);

        %werte normieren


        %desired speed




        %entscheiden
        x=0;
        y=0;
        xv = zeros(8);
        xv = xv(1:8)';
        yv = xv;
        ref =0.;
        count=0;   %wieviel mit gleichem wert
        %gr ssten wert suchen
        for i=1:3
            for j=1:3
                if (B(i,j)>= ref)
                    a = B(i,j);
                    if(a==ref)
                        count = count +1;
                        xv(count)=i;
                        yv(count)=j;
                    else
                        ref = B(i,j);
                        count = 1;
                        xv(1)=i;
                        yv(1)=j;
                        x = i;
                        y = j;
                    end
                end
            end
        end

        %x,y are coordinates in B, where the biggest value is
        % count>1 means that the biggest value is in more than one
            places
```

```
        if (count > 1)  %more than one places
            %wenn ref null ist an mehr als einem platz, stehen bleiben
            if (ref ==0)
                x = 2;
                y = 2;
            else
                a = rand(count);
                a = a(1:count)';
                b=max(a);                   %let chance choose
                for i=1:count
                    if (a(i)==b)
                        x = xv(i);
                        y = yv(i);
                    end
                end
            end
        end

        %absolute koordinaten ausgeben
        if (x == 1)
            v(1) = v(1)-1;
        end
        if (x == 3)
            v(1) = v(1)+1;
        end
        if (y == 1)
            v(2) = v(2) -1;
        end
        if (y== 3)
            v(2) = v(2) +1;
        end
    end

end

end
```

decision()

```
function A = decision(B,chaos,y,n)

%B normieren
A = normMat(B);

%v gibt grad an panik/desired velocity?? an
%v zwischen 0-1

r = rand(9);
r = r(:,1)*chaos*(1-(y/(n-2)));         %r(i) im intervall [0,v]

%optional zuf llig +/- rechnen?!
k=0;
for i=1:3
    for j=1:3
```

```
            k = k +1;
            sign = rand;
            if (sign < 0.5)
                sign = -1;
            end
            if (sign > 0.5)
                sign = 1;
            end
            if (sign == 0.5)
                sign = 0;
            end
            A(i,j) = A(i,j)*(1+sign*r(k));
        end
end

A = normMat(A);









end
```

potmat()

```
function [A] = potmat(m,n,o,q)

%OUTPUT:
% (mxn+2) Matrix
%INPUT:
% o = 1: rechteckig, o=2: konisch
% q = breite Ausgang


A = zeros(m,n);

if o == 1

            for r=1:q
                A(ceil(m/2-q/2)+r,n-5) = 2*n;

                for j=1:n
                    A(ceil(m/2-q/2)+r,j)= 2*n - (n-j);
                end
            end




            for i=1:ceil(m/2-q/2)
                for j=1:n
```

```
                A(i,j) = 2*n - ((i-ceil(m/2-q/2))^2+(j-n)^2)^(1/2.);
            end
        end
        for i=ceil(m/2+q/2):m
            for j=1:n
                A(i,j) = 2*n - ((i-(m/2+q/2))^2+(j-n)^2)^(1/2.);
            end
        end

    A(:,1) = 0;
    A(:,n) = 0;
    A(1,:) = 0;
    A(m,:) = 0;
    for r=1:q
        A(ceil(m/2-q/2)+r,n) = 2*n;
    end
end

if o == 2
        for r=1:q
            A(ceil(m/2-q/2)+r,n-5) = 2*n;

            for j=1:n
            A(ceil(m/2-q/2)+r,j)= 2*n - (n-j);
            end
        end




        for i=1:ceil(m/2-q/2)
            for j=1:n
                A(i,j) = 2*n - ((i-ceil(m/2-q/2))^2+(j-n)^2)^(1/2.);
            end
        end
        for i=ceil(m/2+q/2):m
            for j=1:n
                A(i,j) = 2*n - ((i-(m/2+q/2))^2+(j-n)^2)^(1/2.);
            end
        end

    A(:,1) = 0;
    A(:,n) = 0;
    A(1,:) = 0;
    A(m,:) = 0;

    for a=1:(n-1)
        for b=1:m
            if b<=a*(ceil(m/2-q/2)/n)
                A(b,a) = 0;
            end
        end
    end
    for a=1:(n-1)
        for b=1:m
            if b>=-a*(ceil(m/2-q/2)/n)+m+1
```

```
                A(b,a) = 0;
            end
        end
    end
    for r=1:q
        A(ceil(m/2-q/2)+r,n) = 2*n;
    end


end

A = [A,zeros(m,2)]
```

potmat2()

```
function [A] = potmat2(m,n,o,q,d)

%OUTPUT:
% (mxn+2) Matrix
%INPUT:
% o = 1: rechteckig , o=2: konisch
% q = breite Ausgang
% d = relative zahl f r die "schnelle" des abfalls

A = zeros(m,n);

if o == 1

        for r=1:q
            A(ceil(m/2-q/2)+r,n-5) = exp(-2*n);

            for j=1:n
                A(ceil(m/2-q/2)+r,j)= exp(-(n-j)/(n/d));
            end
        end

        for i=1:ceil(m/2-q/2)
            for j=1:n
                A(i,j) = exp(-((i-ceil(m/2-q/2))^2+(j-n)^2)^(1/2.)/(n/d
                    ));
            end
        end
        for i=ceil(m/2+q/2):m
            for j=1:n
                A(i,j) = exp(-((i-(m/2+q/2))^2+(j-n)^2)^(1/2.)/(n/d));
            end
        end

    A(:,1) = 0;
    A(:,n) = 0;
    A(1,:) = 0;
    A(m,:) = 0;
    for r=1:q
        A(ceil(m/2-q/2)+r,n) = 1;
```

```
    end
end

if o == 2
        for r=1:q
            A(ceil(m/2-q/2)+r,n-5) = exp(-2*n);

            for j=1:n
            A(ceil(m/2-q/2)+r,j)= exp(-(n-j)/(n/d));
            end
        end

        for i=1:ceil(m/2-q/2)
            for j=1:n
                A(i,j) = exp(-((i-ceil(m/2-q/2))^2+(j-n)^2)^(1/2.)/(n/d));
            end
        end

        for i=ceil(m/2+q/2):m
            for j=1:n
                A(i,j) = exp(-((i-(m/2+q/2))^2+(j-n)^2)^(1/2.)/(n/d));
            end
        end

    A(:,1) = 0;
    A(:,n) = 0;
    A(1,:) = 0;
    A(m,:) = 0;

    for a=1:(n-1)
        for b=1:m
            if b<=a*(ceil(m/2-q/2)/n)
                A(b,a) = 0;
            end
        end
    end

    for a=1:(n-1)
        for b=1:m
            if b>=-a*(ceil(m/2-q/2)/n)+m+1
                A(b,a) = 0;
            end
        end
    end

    for r=1:q
        A(ceil(m/2-q/2)+r,n) = 1;
    end


end

A = [A,zeros(m,2)]
```

obst()

```
function [B] = obst(A,sm,sn,m,n)

%OUTPUT:
% Matrix A including obstacle
%INPUT:
% A input matrix
% sm width of obstacle
% sn length of obstacle
% m m-position of obstacle
% n n-position of obstacle

B = A;

for j=0:floor(sm/2.)

for i=1+j:sn-j
    B(m+j,n+i) = 0;
    B(m-j,n+i) = 0;
end

end
```

countobst()

```
function [counto] = countobst(A,m,n,q)

%INPUT:
% A input matrix
% m m-position of obstacle
% n n-position of obstacle

%OUTPUT:
% counts occupied space in an area qx3q around the exit of matrix A

 counto = sum(A(ceil(m/2-q/2-q):ceil(m/2-q/2-q)+3*q,(n-q):(n-1)) == 0);
 counto = sum(counto);

end
```

dens()

```
function [ges, full] = dens(A,m,n,q)
%OUPUT:
% Ausgabe der dichte an Personen beim Ausgang

%INPUT:
% A Input matrix
% nxm Martix
% q Breite Ausgang
```

```
% Ausgabe: ges = Anzahl besetzter P l tze , full = P l tze im
% Bereich qx3p beim Ausgang.

 full = [];
 ges = [];


 full = sum(A(ceil(m/2-q/2):ceil(m/2-q/2)+q,(n-q):(n-1)) == 0);
 full = sum(full);

 siz = size(A(ceil(m/2-q/2):ceil(m/2-q/2)+q,(n-q):(n-1)));

 ges = siz(1)*siz(2);
```

createMat()

```
% INPUT
% number of people "pers"
% OUTPUT
% matrix M2 which is 3x4xpers-shaped, M2 contains the information about
    each person

function [ M2 ] = createMat(pers)

    for i=1:pers
        M(:,:,i) = zeros(3,3);
    end
    for i=1:pers
        M2(:,:,i) = [M(:,:,i),zeros(3,1)];
    end

end
```

getCoord()

```
function [c1,c2,act,inact] = getCoord(M2)
% get information out of matrix M2
%
% saves the number of active people into c1
% inactive people into c2
% coordinates of active people to act
% coordinates of inactive people to inact
% save values by "[c1,c2,v1,v2] = getCoord(M2);"

v = size(M2);
pers = v(3);

c1 = 0;
c2 = 0;

for i=1:pers
```

```
    if M2(3,1,i)==0
        c1 = c1 + 1;
    end
    if M2(3,1,i)==1
        c2 = c2 + 1;
    end
end

act = zeros(2,c1);
inact = zeros(2,c2);
count1 = c1;
count2 = c2;

    for i=1:pers
        if M2(3,1,i)==0
            act(1,c1-(count1-1)) = M2(1,1,i);
            act(2,c1-(count1-1)) = M2(2,1,i);
            count1 = count1 - 1;
        end

        if M2(3,1,i)==1
            inact(1,c2-(count2-1)) = M2(1,1,i);
            inact(2,c2-(count2-1)) = M2(2,1,i);
            count2 = count2 - 1;
        end

    end

end
```

pdens()

```
% compute number of people by percentage
% INPUT:
% size mxn and density x
% OUTPUT:
% number of people pers

function [ pers ] = pdens(x,m,n)
% people densitiy in %

    pers=round((m*n)*x);

end
```

persObst()

```
% write each person as obstacle into potential by coordinates
% person = obstacle = 0
% INPUT:
% matrix M2, potential P2, number pers
```

```
% OUTPUT:
% potential P2

function [P2] = persObst(M2,P2,pers)

for p=1:pers

    x = M2(1,1,p);
    y = M2(2,1,p);

    P2(x,y) = 0;

end


end
```

persPot()

```
% get the potential values for one person p by coordinates
% INPUT:
% matrix M2, potential P2, person p
% OUTPUT:
% matrix M2

function [M2] = persPot(M2,P2,p)

    x = M2(1,1,p);
    y = M2(2,1,p);

    for i=-1:1
        for j=-1:1
            M2(i+2,j+3,p)=P2(x+i,y+j);
        end
    end

end
```

persPotAll()

```
% get the potential values for every person by coordinates
% INPUT:
% matrix M2, potential P2, number pers
% OUTPUT:
% matrix M2

function [M2] = persPotAll(M2,P2,pers)

for p=1:pers

    x = M2(1,1,p);
```

```
    y = M2(2,1,p);

    for i=-1:1
        for j=-1:1
            M2(i+2,j+3,p)=P2(x+i,y+j);
        end
    end
end


end
```

normMat()

```
function A = normMat(B)

%B ist pot matrix um person
%A soll berarbeitete matrix ausgeben (velocity)

%B normieren
ab = 0;
for i=1:3
    for j=1:3
        ab = ab + B(i,j);
    end
end

for i=1:3
    for j=1:3
        B(i,j) = B(i,j)/ab;
    end
end
A=B;

end
```

savepics()

```
function save = savepics(A)
v = size(A);
m = v(1);
n = v(2);
q = v(3);
for i=1:q
    for k=1:m
        for l=1:n
            if A(k,l,i)== 0
                A(k,l,i) = 1;
            else
                A(k,l,i) = 0.5;
            end
```

```
        end
    end
    filename = sprintf('pic_%d.bmp',i);
    imwrite(A(:,:,i),filename,'bmp')
end
print 'done'

end
```

simulation_test_random

```
% TOTATLLY RANDOM
% only used to compare with PARTIALLY RANDOM
% in the end PARTIALLY RANDOM chosen

m = 100;          % Matrixgroesse aktiv mit wand oben und unten
n = 100;          % Matrixgroesse aktiv inkl. Ausgang und wand links
endmove = n;      % fuer move-funktion
ausgang = 10;
chaos = 0.2;

P = potmat2(m,n,1,ausgang,5);    % 1 normal, 2 konisch
P = obstneu(P,3,6,m/2,n-13);
%P = obstneu(P,3,6,m/2+8,n-13);

w = size(P);
m = w(1);
n = w(2);    % resultierendes n = n+2

P2 = P;      % copy of potential, that will be used. people walk on P2.

Bilder = zeros(m,n,[]);
mv = 0;
video = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% density or absolute value
pers=pdens(0.15,m-2,n-4);
%pers=4;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

new_pers=pers;  %number of active people
old_pers=pers;

%create 3x4xPers-Matrix
M2 = createMat(pers);

%initalize random coord. to matrix
M2 = init(M2,P,pers,m,n);

%person = obstacle
P2 = persObst(M2,P2,pers);
Bilder(:,:,video)=P2;

%assign potential to each (!) person by (x,y)
```

```
M2 = persPotAll(M2,P2,pers);

pass = [];            %number of people passing exit per interval
i = 1;                %pass-counter
numerator = pers;     %helping numerator
density =[];

q = 1;                %iteraton-counter

while 1               %do, till the last person reaches the exit

    p = randi(new_pers,1,1);    %random (!) active person is chosen

    alt = M2(:,1,p);            %old coordinates

    %assign potential to current person
    M2 = persPot(M2,P2,p);

    v=movev3(M2(:,:,p),n-2,chaos);       %MOVE !!!
    x=v(1);
    y=v(2);
    z=v(3);

    if z==0      %person moved & is still active

        %person=obstacle, restore original potential-value
        P2(alt(1),alt(2)) = P(alt(1),alt(2));
        P2(x,y) = 0;

        %update: new coordinates to matrix
        M2(1,1,p) = x;
        M2(2,1,p) = y;
        M2(3,1,p) = z;

    else         %person reached exit

        P2(alt(1),alt(2)) = P(alt(1),alt(2));

        M2(:,:,p) = [];           %delete person
        new_pers = new_pers-1;    %one person less


    end

    mv = mv+1;
    modulo = round(pers/2);

    if mod(mv,modulo)==0

        video = video +1;
        Bilder(:,:,video)=P2;

    end

    if mod(mv,modulo)==0

```

```
        %density
        density(video-1) = dens(P2,m,n,1,ausgang);

        %fluss
        pass(video-1)=old_pers-new_pers;  %get number of people that passed the exit in the interval
        %step(i)=q;
        %i = i+1;
        old_pers=new_pers;
        numerator=numerator+old_pers;

    end

    if new_pers==0  %breaking the while-loop, when all people at exit
        break
    else
        q = q+1;    %iteration counter
    end

end

Bilder(:,:,video+1)= P2;

%Efficiency plot
step = linspace(1,video-1,video-1); %"Abtastinterval"
mittel = mean(pass);                %mean value

hold on
bar(step,pass);
%plot(step,ones(1,video-1)*mittel, '-r')
%plot(step,ones(1,video-1)*ausgang, '-g')
plot(step,density, '-r');
title('number of people passing the exit per sampling-interval')
%legend('number of people','mean value', 'max = exit width')
hold off
```