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

Project Report

Evacuation Bottleneck:
Simulation and analysis of an evacuation of a lecture room with MATLAB

Dario Biner & Noé Brun

Zurich
May 2011
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.

Dario Biner

Noé Brun
## Contents

1 Introduction and Motivations  
2 Description of the Model  
3 The Room  
   3.1 Room drawing  
   3.2 Room Design  
   3.3 Room plotting  
4 The Path  
   4.1 Path Design  
      4.1.1 P Matrix  
      4.1.2 PA Matrix  
      4.1.3 PE Matrix  
   4.2 Add Exit Algorithm  
   4.3 Path Optimization  
5 The Agent  
   5.1 Matrix A  
   5.2 Agent Algorithm  
      5.2.1 Agent and Density  
      5.2.2 Agent and Exits  
   5.3 Agent Initializing  
   5.4 Velocities  
6 Main Program  
   6.1 Initialization  
   6.2 Loop  
   6.3 Computations  
7 Simulation, Results and Discussion  
   7.1 Analysis of the impact of certain variables  
      7.1.1 MATLAB’s random function in PriorityUpdate and Movement  
      7.1.2 Different Values for MaxDensity and MaxOutflow  
      7.1.3 Velocity  
   7.2 Simulation of the ETA F5 lecture hall  
      7.2.1 Various crowd size  
      7.2.2 Different exit activity
1 Introduction and Motivations

At the start of this course, one is given a selection of topics on which MATLAB simulations can be done and investigated. Our topic is about Evacuation Bottleneck. We chose this topic because while reading literature, we asked ourselves "How save are our lecture halls really?" and "What happens, if a fire breaks out during a lecture with a big audience?", "Could we get out in time?".

So we decided to do our project about an evacuation event of a lecture room. The simulation should be able to determine how fast people could get out through the given exits. The goal is to determine the seats with the longest getting out time and to visualize the bottleneck regions.

Crucial parameters such as the amount and the location of the exits (potential bottlenecks), the amount of people, the size of the room and the distribution of the seats will be taken into account.

Figure 1: Scherrer lecture hall ETA F5 [2]
2 Description of the Model

For our main simulation we decided to investigate the Scherrer Hörsaal ETA F5. It is one of the largest lecture halls of the ETHZ with about 600 seats, 455.55m² of area and six exits. We decided to design our simulation environment in such a way that other rooms and settings could be evaluated as well which makes it more flexible and more useful.

In order to describe the “architecture” of the room in MATLAB we had to introduce key values which correspond to a certain object kind in real life, i.e. walls, seats, tables etc. With these key values we can parse our matrices and find ways for the agents to get out and determine were they can’t go.

It seems obvious that it would be very hard work and very time consuming to “draw” this matrices into MATLAB directly. We had to come up with a way to do this fast, easily and graphically.

For the MATLAB-implementation the main idea was to store all the information belonging to one task in one matrix. So we introduced three main matrices: one for the room, one for the paths and one for the agents. It’s clear that we have some other matrices as well because some data has to be saved differently or the size of the matrix doesn’t fit with the main one. Another reason for this is keeping order and hence not losing the overview.

With this concept we don’t have too much variables and we can declare them globally. Another advantage of this implementation is that we have all the variables dynamically available such that we can add, remove or change easily the data and MATLAB supports it very well with the intrinsic matrix functions.

The basic idea of the simulation is quite simple. In a way, we create a “game field” in matrix form with as many details as needed. The key objects we add are quite the most important and obvious ones we observe in the room: location of the exits, seats, paths etc.

Then we add an autonomous agent which should of course be as human as possible and we place it in the “room”. When pressing the “start button” the system begins to run and the agents autonomously find their way out and make their own decisions while doing so.

Finally one is presented the results and analyzing plots.
Figure 2: Simple schematic of the simulation
3 The Room

Since MATLAB will be our simulation platform it is clear that we will use the strengths of MATLAB, namely matrices and their manipulations, to realize our model. The room will be discretized into small areas which are stored in the matrix $R \in \mathbb{N}^{m \times n \times i}$ where the key values represent certain objects which will be explained later. It depends on the simulation complexity how fine-grained this discretization is.

The transition between continuous space (fig 4) and discrete matrices (fig 5) will make us lose some information. For example the space between two seats which in our case is omitted. The approximation of the real life causes further problems for example an exit consists of one door while widthwise it must be represented by two discretization units. The algorithm thus reads this as two exits close to each other which doesn’t make much sense. This problem can either be treated in the code or in the discretization step. Both approaches might eventually compromise the simulation results or the generic application of the system. Details follow in sec.4.3.

For modeling the room we introduced the following objects:

<table>
<thead>
<tr>
<th>Object</th>
<th>Keyvalue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Walls &amp; unsurmountable</td>
<td>-inf</td>
</tr>
<tr>
<td>objects</td>
<td></td>
</tr>
<tr>
<td>Fillups</td>
<td>-2</td>
</tr>
<tr>
<td>Tables</td>
<td>-1</td>
</tr>
<tr>
<td>Seats</td>
<td>0</td>
</tr>
<tr>
<td>Paths</td>
<td>1</td>
</tr>
<tr>
<td>Exits</td>
<td>inf</td>
</tr>
</tbody>
</table>

With these values it should be possible to represent the room under test in cartesian grids like matrices. In fig.3 one can see a small example of how the values are distributed. For design reasons the surrounding border (walls) was not drawn, but would be mandatory for a simulation!

The reason why we added the table keyword (we could also be using the walls keyword for example) is to create a more realistic representation of the room i.e. keep the all options open for the algorithm. One can imagine that certain people might need to jump over tables if the way was blocked for example.

3.1 Room drawing

For high flexibility of the whole system we tried to find a way to feed MATLAB easily with a new room architecture. The ability of MATLAB to import CSV files lead us to the idea to ”draw” the room in a spreadsheet application like in our case Numbers from apple.
With the export option it is possible to convert the spreadsheet into a CSV file which can be easily imported by MATLAB.

In fig.5 is our version of the lecture hall we wanted to inspect, realized in a spreadsheet structure.

As can be seen in fig.5 there is a key value “fill-up”. These fields should be “ignored” by the agent algorithm i.e. jumped over without time costs. Those fields only exist so that we can fit a not rectangular room into a (obviously rectangular) matrix. This isn’t the case for most of the rooms of course but with this feature almost every floor-plan can be realized.
Figure 5: Spreadsheet representation of the lecture hall ETA F5

3.2 Room Design

The room matrix R has a third dimension as seen in the introduction. With this structure we can store more information and reduce computation time during the simulation loop.

The first layer a) (see fig.6) is the architectural information as explained earlier. The second layer b) stores the accumulation of the crowd density at each coordinate. The third layer c) stores the accumulated waiting time at each coordinate. And lastly the fourth layer d) tracks the time needed for getting out for each starting point. This additional data
is finally used for the simulation results.

\[ \text{Figure 6: Structure of the room matrix } R(m,n,i) \]

### 3.3 Room plotting

For verification whether the room was correctly exported and imported we implemented a MATLAB function `PlotRoom` which draws the room matrix `R` as a plot (see fig. 7).

\[ \text{plot of the room matrix. Exit value = Inf} \]

\[ \text{Figure 7: MATLAB version of the ETAF5 lecture hall} \]
4 The Path

After we got the room structure right and well defined we had to come up with ideas on how to find a way out of the room. As explained earlier we don’t want to do all the heavy computation like the complex human behaviour and decision making in the simulation loop, so we tried to do as much as possible in the initialization (or even offline and save the variables from the MATLAB workspace to the disk). This allows to do possibly even more complex computations during the loop while maintaining reasonable simulation time.

4.1 Path Design

In order for the agents to find their way out it is necessary to implement a structure with which the agents can determine in what direction they should go and whether they can go at all. This is implemented as follows: in a matrix \( R \in \mathbb{N}^{m \times n} \) (the same size as \( R \)) an exit is assigned the maximum value \( \text{maxval} = m \cdot n \) at its coordinates ((1,3) in the example of fig.8). Every step one goes further away from the exit in this discrete grid results in a decrement of 1 and gets saved. Of course the “rules” on how to fill this matrix must be read from \( R \) (i.e. where the paths and the seats are). Our MATLAB-function \texttt{AddExit} does exactly this task. Once the whole grid is filled according to the constraints of \( R \) one can find out in what direction the exit is, namely in the direction of the next higher value. This path matrix is initialized with zeroes and hence where there is no path, seat or exit the value 0 remains.

![Figure 8: Simple example of the algorithm of the AddExit function](image)

4.1.1 P Matrix

To cope with various exits, we introduced a tensor \( P \in \mathbb{N}^{\text{max} \times j} \) where we store every exit’s value matrix mentioned above in the first two dimensions. The third dimension then stands for one of the \( k \) exits.
The second last matrix entry \(P(x,y,k+1)\) is not an exit layer. We use this layer to store the index with the highest value at the given coordinate. Mathematically this results in

\[
P(x,y,k+1) = a \mid \max_{a \leq k} P(x,y,a)
\]  

(1)

In the last matrix entry \(P(x,y,k+2)\) we store the actual maximum value over all the exit levels. Namely:

\[
P(x,y,k+2) = \max_{a \leq k} P(x,y,a)
\]  

(2)

These two additional matrices can be computed at the beginning and hence reduce the computation time during the simulation loop. In fig.9 an example for the different layer values of \(P\) is shown.

### 4.1.2 PA Matrix

The matrix \(PA\) (Path Arrows) basically carries the same information as \(P\) but in vector form. This means that the \(k\) exit layers contain the direction to the next exit instead of just the values. This helps speeding up the loop and can also be used for verification and analysis. It will also come in handy in the path finding algorithm explained later.

We used imaginary numbers to represent the directions. The real part \((\pm 1)\) is used for the rows and the imaginary part \((\pm i)\) is used for the columns.

### 4.1.3 PE Matrix

The PE (Path Exit) matrix was introduced for several reasons. Firstly the agents algorithm will make use of it for deciding which direction to take. Secondly it stores information
about the density evolution around the exits.

The structure of \( \text{PE} \) is rather particular. The values around an exit is cone shaped distributed (see fig.11). With this shape one can determine how many people are in the close neighborhood area of the exit. This zones are the most interesting ones for bottleneck analysis as well. Further, the decision algorithm can easily track very crowded exits and can thus influence the behaviour of the agents.

In the layer b) the exit is at the tip of the cone with the highest value which can be set by a parameter. Then the values decrease until the number 1 is reached which is also the indicator for the end of the cone.

In the first layer a) the value distribution is the same as in the second layer but the values are only boolean indicators whether there is a value (1) in the second layer or not (0). This allows to easily mask out the agents in the cone from the \( \text{A} \) matrix. The upper levels are for saving the density near the exits for every simulation loop. For example \( \text{PE}(\cdot,\cdot,3) \) stores the density after the first loop. Naturally the matrix \( \text{PE} \) grows quite fast but allows precise analysis of the bottleneck densities.
4.2 Add Exit Algorithm

The $[P] = \text{AddExit}(R, P, c, \text{value}, \text{depth})$ function is recursive and “pours” values into a matrix starting from the point $c$. It follows the accessible paths given in the matrix $R$ and counts the value down by 1 when going one step further. See fig.8 for reference. We had to introduce the depth parameter because MATLAB is very inefficient with recursion (especially with increasing data stack for big matrices). It counts the recursion depth and stops at a certain (hardcodeable) value. This value should be large enough in order to allow the AddExit function to cover all the paths.

The function $P = \text{PourPathValues}(R)$ retrieves the indices of all the exits from the matrix $R$ and starts the AddExit function with the appropriate arguments.

4.3 Path Optimization

Since we’re mainly looking at one particular lecture room we want to optimize our algorithms for this exact case. A particular problem that one wide exit in reality results in two distinct exits in our matrix system mentioned above, had to be solved. An agent usually aims for a higher value than the one “he’s standing on”. Since on two parallel running paths the perpendicular value is the same as the actual one the agent won’t change “the lane” although there might be space and better chance to get out. In fig.12 such a situation is shown. The right lane (to the very same exit) is free and in a primitive algorithm “go to the next higher value” this would stay so. One optimization we did is to define which two distinct exits are in fact the same in real life and take the maximum of the two values and add it to both of the exit layers in $P$.

![Figure 12: Example of a parallel path situation.](image)
5  The Agent

Like for the room and for the paths we’d like to have a similar system to keep track of
the agents. We thus have decided to introduce the matrix $A$. The choice on how to handle
an agent is very important. There are essentially two kind of approaches: the first uses
one controlling unit which supervises all the agents or agent groups. The second approach
is the autonomous agent. Every agent is represented by its own program instance and
acts on its own. It’s obvious that on one hand this choice is closer to reality and offers
many options but will consume a lot more resources on the other hand. We chose the
autonomous agent in the end.

5.1  Matrix $A$

The $A$ matrix holds all the data needed for representing the agent states. The exact descrip-
tion can be seen in fig.13.

![Figure 13: Agent matrix $A$](image.png)

- a) indicates whether an agent is at this coordiante (1) or not (0)
- b) duplicate of the $P(:,:,k+2)$
- c) stores the direction of the last move of the agent in complex coordinates (see PA)
- d) accumulates the waiting time of the agent
- e) saves the exit the agent is headed to
- f) starting position of the agent (for analysis)
- g) counts how many loops it takes until this agent has left the room
- h) set the velocity of the agent (for more realistic behavior)
5.2 Agent Algorithm

The agent system bases upon two main quantities: the *density* at the exits and the *proximity* of an exit. These quantities can either endorse each other (the closest exit has also the fewest people nearby) or diverge (the closest exit has the most people nearby).

Our system already “knows” where the closest exit is and will therefore retrieve the density values for comparison. When the decision is made for which exit an agent is headed, the system then looks for the next free unit.

5.2.1 Agent and Density

The agent fetches the information about the density of his current targeted exit in the PE matrix and decides then whether he continues to this exit or rather looks for another one with less people. This choice is based on a density threshold. If the density is higher than a certain value another exit is chosen. Because there are always two parallel exits in our room, we had to code a function which takes the density between adjoining paths and exits into account.

5.2.2 Agent and Exits

Once an agent has made his decision to what exit he’s heading to he’ll try to get there. The function *GetNextMove* looks for the closest free position by looking for a higher value in P. Important is that the *direction* is right, so that they don’t start to oscillate. This is where our prepared matrix PA comes in handy. If an agent can’t find a free spot with higher value, then the higher value condition gets ignored.

Should all fail, the agent will just wait. This waiting state is tracked and if he has to wait quite some time it will influence the decision for choosing another exit.

5.3 Agent Initializing

The initial conditions of the simulation are obviously quite important. That's why we implemented three versions to start with:

- fill all the seats
- fill the seats randomly with predefined amount of people
- fill all the seats and even the paths

We didn’t want to invest into a complex algorithm for placing the agents since it heavily depends on the kind and the amount of the participating people. The random functions of MATLAB gives us an easy and efficient way of creating semi-realistic situations.
5.4 Velocities

Until now the idea was to update every agent in one loop cycle which results in the same movement velocity of all the agents. Since in real life this is rarely the case we implemented a velocity term for each agent.

The idea is quite simple. Every agent gets a velocity value. The lower the value the faster he’s able to move. The value basically states how many loops he must “wait” until he can make the next move.

To distribute the velocities amongst the agents we decided to use the Gauss window function since its results are often quite realistic. There are other window functions which can be selected for example Chebischew, Turkey or Kaiser. One can then set up how many different velocities there should be. This highly influences the speed and the outcome of the whole simulation, so this value has to be chosen carefully. Also try to use an odd number since this results in a single main velocity. Of course the standard derivation \(\sigma^2\) can be chosen as well.

![Gauss Distribution for 9 Different Velocities](image)

Figure 14: Gauss distribution of 9 different velocities
6 Main Program

The main program SecuritySimulation starts and controls the whole simulation. It gathers all the set parameters, loads the given room from a file and sets up all the needed matrices before starting the loop and ultimately producing the output plots and statistics.

6.1 Initialization

With the LoadRoom function one can choose between two rooms. The first one is the lecture hall ETA F5 and the second one is a fictitious room. It is also possible to add a complete different room from a .csv file which will take longer though since all the path values have to be calculated.

The second part of the initialization places the agents and adjusts all the variables needed for the main loop.

6.2 Loop

The main loop primarily updates the agents. The function AgentUpdate(x,y) is called to move the agent at (x,y). There are many ways of selecting the update sequence of the agents as seen in the lecture. We chose to update the agents closest to the exits first in order to make room for the ones behind them. If multiple agents are “affected” and about to be updated we chose their sequence randomly with a Permutation function.

It is basically not correct to say “an agent is being updated” yet. The algorithm explained only defines what coordinate gets updated first and not the agent since there can be multiple agents at one coordinate (density!). But once the position to update is chosen AgentUpdate selects the agent lowest in the matrix and moves him, which then defines a clear order.

6.3 Computations

At the end we preprocess all the data needed for the analysis and plots like the density per exit, waiting time per exit etc. (see sec.7)
7 Simulation, Results and Discussion

We had to go through several steps in the development of the simulation in order to get correct results. We had to analyze the influence of every parameter and variable on the outcome of the simulation. For example the implementation of pseudo random functions or different movement velocities caused difficulties in the verification of their correct behaviour. In the following we’ll concentrate on the impact of several parameters on the simulation outcome.

7.1 Analysis of the impact of certain variables

In order to analyze an impact correctly we strongly related to our main lecture hall. Since the parameters we have to set up rely on interactions between agents we had to maximize these confrontations. This is obviously granted if we fill the room entirely (561 agents). With this setup we then “calibrate” our parameters to fit the room realistically. This step is also needed because of the discretization of the room.

7.1.1 MATLAB’s random function in PriorityUpdate and Movement

We use the MATLAB function randperm in Permutation and randi in GetNextMove. One can see in fig.15 that there’s quite a difference in the number of loops needed depending on whether there was some random function used or not.

![Impact of Permutation function on number of loops in multiple simulations (MaxPer =1, MaxOut =1, Velocity =1)](image)

Figure 15: Impact of permutation function on number of loops in multiple simulations (MaxPer =1, MaxOut =1, Velocity =1)

The impact of random functions seems quite striking and has to be considered. To make better analysis it is therefore recommended to do several runs and average the outcomes.
7.1.2 Different Values for MaxDensity and MaxOutflow

The parameters MaxDensity and MaxOutflow are also important to evaluate beforehand. We had to find the right ratio between the maximum amount of people in one unit and the maximum outflow of an exit. We figured that on one unit which is about 0.5 m$^2$ about five people could be at the same time.

The MaxOutflow parameter plays a crucial role on the outcome of the simulation. It is influenced by the agents in the following way: if there’s low density but high velocity of the agents the outflow is more important than in the case of high density and low velocity[1].

The velocity has small relevance in the case of low density situations because people can leave without confronting each other which would produce a bottleneck situation whereas in high density regimes it becomes quite relevant (see fig.16).

Since the “throughput” of an exit is limited and a crowd approaches with high velocity, a bottleneck situation will occur. This results in a shock wave effect near the exit. To simulate this effect in a simple way we introduced the limit of the outflow. For a short time a blockade will build up which then makes the agents have to wait a bit until they can force out again.

In fig.16 the simulation has been run with different MaxDensity and MaxOutflow values. Each parameter was swept from 1 to 10.

From now on we’ll use MaxPer = 4 and MaxOut = 2.

![Figure 16: Maximum density vs. maximum outflow](image-url)
7.1.3 Velocity

For the analysis of the different velocities we ran the simulation with randomly selected exits to update from and with a deterministic order of updating. We see a slight difference in the results (fig.17) and most notably we get “better” results with the randomized version.

The results are very comprehensible because the larger the derivations of the velocity the longer it takes to evacuate compared to a unified velocity (the diagonal in fig.17) which manifests itself in lower evacuation times.

In order to have interesting simulations in reasonable times we chose 7 different velocities and $\sigma=3$ for our lecture hall.

![Graph showing the velocity distribution with and without permutation.](image)

Figure 17: Velocity distribution with the gaussian window
7.2 Simulation of the ETA F5 lecture hall

We now concentrate more in our lecture hall the ETA F5. We’ll try different initial combinations of the active exits. This should show us the potentially dangerous seats and the most probable bottleneck locations.

7.2.1 Various crowd size

In this test we varied the number of persons in the room and ran the simulation with randomly distributed people and increased the amount steadily and noted the number of loops until every agent has left the room. The results can be seen in fig.18. Since the initial placement is still random we ran the test several times.

![Figure 18: Evolution of random placement with more people, MaxPer=4, MaxOut=2, #Velocities=7, σ=3](image)

One can see that the derivation of the several simulation results vary in a constant intervall of about 15 loops. This “error” is only due to the random initial placing.

7.2.2 Different exit activity

We now want to look at the influence of the individual exits. We ran the simulation for each exit once with the maximum amount of agents and always keep one exit closed (fig.19).
Figure 19: Simulation runs with full room, left away one exit at a time. Single exits (means one door closed) and entire exits

7.2.3 Discussion

Intuitively one would think that the amount of loops needed should increase with the amount of people respectively decrease with less people (fig.18).

Also from fig.19 one would expect fewer differences and more symmetry in the number of loops. What is the problem? To find an answer to this we had to do some more analysis. We plotted therefore how many agent needed a certain amount of steps (fig.20).

Figure 20: Time distribution over all agents, MaxPer=4, MaxOut=2, $\#$Velocities=7, $\sigma$=3

From this we can see that the problem comes from very few people who have a long
time to get out. This means that in fig.18 we basically plot the time of the slowest agent and his time is more or less always the same since he’s not affected by the bottleneck effect.

![Graph](image)

**Figure 21:** Differently sorted time distribution, MaxPer=4, MaxOut=2, #Velocities=7, σ=3

In fig.21 we see that there’s a linear evolution at the beginning which then turns into an exponential (at around 35 steps). That means that the slowest agents have a striking influence on the final evacuation time.
8 Summary

After all those simulations we conclude that the bottleneck effect has not always a big influence. Especially if the velocity differs a lot amongst the agents (i.e. there are some quite slow people). Fig.20 shows that the majority of the people is able to leave the room quite quickly. But in almost every plot we evaluate the time until every agent got out. This means that if we simulate with different velocities and analyze the overall time needed we basically only look at the slowest agent. This is not a bad result because if the goal is to save everybody we have to look at the worst case.

To have realistic results, we need to set the parameters right. In order to do this we need a lot of information about the crowd to simulate (maximum density, velocity distribution, etc.).

While working on this project we found out that it is very difficult to get good results. So we decided to spend a lot of time on programming in order to get a generic system which can be set up for many different scenarios.

We can, for example, simulate an evacuation of young, athletic people or older and slower people. One only needs to feed in the right data that corresponds to one’s situation.

Since in this thesis we concentrated on the lecture hall ETA F5, it is self-evident that the code is automatically optimized for this kind of room setting. For another room there must be some slight adaptation. One can imagine that keeping all the algorithms as generic as possible is a lot of work and would go beyond the scope of this thesis.

In the end every room and every simulation needs special attention to its very own characteristics anyway.
References


Appendix
%This is the print file to export the project as .ps
%
%script

%Main Program
%control the simulation
%Define the parameter
%Written by Dario Biner & Noe Brun
%ETH May 2011
%feel free to use, improve and enjoy it
%
%prepare the workspace
clear all;
clc;
clf;
close all;
tic
RandStream.setDefaultStream(RandStream('mt19937ar','seed',0))

%%
%Globals
global MyPause; %Activate pause in Simulation
global scrsz; %Your actual Screen Size
global PlotData; %Activate the plot functions [a,b]
global ActRand; %Activate the Random function in Permutation and GetNextMove
global PFS; %Plot full screen
global ActRoom; %Active Room Nr X
global MyCounter; %Main Loop Counter
global PECounter; %Counter for PE Matrix
global PPCounter; %Counter for PP Matrix
global LimitLoop; %Set a limit for the maximum numbers of loops
global MaxPersonPerUnit; %Define Maximum Person Density per Unit
global MaxOutFlow; %Define how much people can go out at one time
global DeleteExit1; %Define the exit to be deleted(0) in room 1
global DeleteExit2; %Define the exit to be deleted(0) in room 2
global MyMax; %Biggest value for the path search
global N; %Amount of people
global A; %Agent Matrix 4D
global R; %Room Matrix 3D
global P; %Path Matrix with integer value 3D
global PA; %Path Matrix with arrows 3D
global PE; %Information about Density near to the Exits
global PP; %Outputflow for each exit for all loops
global PD; %Outputflow for each exit
global ThreshNear; %for Room 1 for 2 exit aside
global ThreshDens; %Threshold for Density at exit cones
global ExitConeR; %Length of the exit corners
global AgWin; %Velocity distribution
global T; %Sorted Persons depending on time
global TT; %For each time numbers of people
global MyNorm; %Norming factor for Velocity Distribution
global NrExits; %Number of Exits
global NumVelo; %Number of Velocities
global Sigma; %Derivation factor for Velocity Distribution
%
%Define your Values for the Simulation
ActRoom =1;
DeleteExit1 = [1,1,1,1,1,1,1,1,1,1,1,1];
DeleteExit2 = [1 1 1 1];
ActRand = 1;
PlotData = [1,1]; %first for the PrintAgent, second for last plots
PFS = 1;
MyPause = 0;
MaxPersonPerUnit = 4;
MaxOutFlow = 2;
ThreshNear = 10; %this threshold is for room 1
ThreshDens = 20*MaxPersonPerUnit; %optimized for room 1
ExitConeR = 8;
NumVelo = 3;
Sigma = 2;
LimitLoop = 500;

if PFS == 1
    scrsz = get(0,'Screensize')-[0 0 0 100];
end

%load Room architectur from csv file
LoadRoom(ActRoom);

%Place the Crowd (a,b)
a = style : 1==Full,2==Random
b = amount of people (only for style 2)
PlaceAgentIni(1,1);
PlaceAgentVelocity(NumVelo,'gauss',Sigma);
GetNearestExit();

%Prepare for simulation
%Set Priority update
MyMin = PriorityUpdate();
%Initializing integration Matrize
R(:,:,3) = A(:,:,4);
R(:,:,2) = sum(A(:,:,1,:),4);
%Print Mains figure
if PlotData(1) == 1
    PrintAgent(2);
end

%Simulation Loop
MyCounter = 0;
PECounter = 1;
PPCounter = 1;
while N > 0
    i = MyMax;
    %Loop for Crowd Update
    while(i >= MyMin)
        if find(A(:,:,2,1)==i)
            [Fx,Fy] = find(A(:,:,2,1)==i);
            [Fx,Fy] = Permutation(Fx,Fy); %mixing the orders
            l = length(Fx);
            for k=1:l
                AgentUpdate(Fx(k),Fy(k));
            end
        end
        i = i-1;
    end
    MyCounter = MyCounter + 1;
    PPCounter = PPCounter + 1;
    No = N;
% Count all the persons still in the room
N = sum(sum(sum(A(:,:,1,:))));

% Readapt priority
MyMin = PriorityUpdate();

% Integrate density of persons
R(:,:,2) = R(:,:,2) + sum(A(:,:,1,:),4);

% Integrate waiting time of people
R(:,:,3) = real(R(:,:,3))+real(A(:,:,4,1));

% Prepare People Counters
GetAgentDensity();
if (PlotData(1) == 1)
    PrintAgent(2);
end

CounterUpdate();

% Security
Test = No-N;
if Test > (sum(DeleteExit1)*MaxOutFlow)
    disp('ERROR 201: Too many got out!!');
end
if LimitLoop == MyCounter
    break;
end
clc;
disp(['Steps : ' num2str(MyCounter/MyNorm)])
end

%%
% Last plots
if MyPause == 1
    pause();
end
if PlotData(2) == 1
    % Plot Density
    figure();
    if PFS
        set(gcf,'Position',scrsz)
    end
    B = (R(:,:,2));
    h = bar3(B);
    % Source from http://efreedom.com/Question/1-2050367/Hide-Zero-Values-Bar3-%20MATLAB
    for i = 1:numel(h)
        zData = get(h(i), 'ZData');
        index = logical(kron(zData(2:6:end,2) == 0,ones(6,1)));
        zData(index,:) = nan;
        set(h(i), 'ZData', zData);
    end
    title('Integration of density of persons')
xlabel('x')
ylabel('y')
zlabel('Number of persons')
% Plot Wait position
figure();
if PFS
    set(gcf,'Position',scrsz)
end
B = (R(:,:,3)./MyNorm);
imagesc(B);
title('Integration of Waiting times of persons')
colorbar
xlabel('x')
ylabel('y')
zlabel('Steps')

%Plot the time Wait value
figure()
set(gca,'FontSize',16)
if PFS
    set(gcf,'Position',scrsz)
end
B = (R(:,:,4)./MyNorm);
imagesc(B);
title('Time needed to go to the nearest exit per seat')
colorbar
xlabel('x')
ylabel('y')
zlabel('Steps')

%Plot Time
TimeAnalysis();
save Simulation_Data
end

%%
%functions

function [P] = AddExit_v2(R, P, c, value, depth)
%Adds exit at 'c' with value 'value' and path values to room
%matrix R
    if depth == 90 % to prevent the recursion to get too far
        return
    end
    [m,n] = size(R);
    x = c(1);
    y = c(2);
    newcoords = [];
    coordcount = 0;

    % "check" directions: left, down, right, up
    % plot(y,x,marker,');
    % pause(0.001)
    %check left
    cx = x;
    cy = y-1;
    if cy > 0 % still in matrix range
        while R(cx,cy) == -2
            cy = cy-1;
        end
        if (value > P(cx, cy) && R(cx, cy) >= 0) %if its path
            newcoords = [newcoords; cx cy];
            coordcount = coordcount+1;
            P(cx, cy) = value;
        end
    end

    %check down
    cx = x-1;
    cy = y;
    if cx > 0 % still in matrix range
        while R(cx,cy) == -2
            cx = cx-1;
        end
        if (value > P(cx, cy) && R(cx, cy) >= 0) %if its path
            newcoords = [newcoords; cx cy];
            coordcount = coordcount+1;
            P(cx, cy) = value;
        end
    end

end
cx = cx-1;
end
if (value > P(cx, cy) && R(cx,cy) >= 0) %if its path
    newcoords = [newcoords; cx cy];
    coordcount = coordcount+1;
    P(cx, cy) = value;
end
end
%check right
cx = x;
cy = y+1;
if cy <= n % still in matrix range
    while R(cx,cy) == -2
        cy = cy+1;
    end
    if (value > P(cx, cy) && R(cx,cy) >= 0) %if its path
        newcoords = [newcoords; cx cy];
        coordcount = coordcount+1;
        P(cx, cy) = value;
    end
end
%check up
cx = x+1;
cy = y;
if cx <= m % still in matrix range
    while R(cx,cy) == -2
        cx = cx+1;
    end
    if (value > P(cx, cy) && R(cx,cy) >= 0) %if its path
        newcoords = [newcoords; cx cy];
        coordcount = coordcount+1;
        P(cx, cy) = value;
    end
end
depth = depth + 1;
for i=1:coordcount
    P = max(AddExit_v2(R,P,newcoords(i,:),value-1,depth),P);
end
end

function AddExitCones()
%Define cones near the exit
%l have been optimized for the room 1
% Globals
global MyPause; % Activate pause in Simulation
global scrsz; % Your actual Screen Size
global PlotData; % Activate the plot functions [a,b]
global ActRand; % Activate the Random function in Permutation and

GetNextMove

global PFS; %Plot full screen
global ActRoom; %Active Room Nr X

global MyCounter; %Main Loop Counter
global PECounter; %Counter for PE Matrix
global PPCounter; %Counter for PP Matrix
global LimitLoop; %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit; %Define Maximum Person Density per Unit
global MaxOutFlow; %Define how much people can go out at one time
global DeleteExit1; %Define the exit to be deleted(0) in room 1
global DeleteExit2; %Define the exit to be deleted(0) in room 2
global MyMax; %Biggest value for the path search
global N; %Amount of people
global A; %Agent Matrix 4D
global R; %Room Matrix 3D
global P; %Path Matrix with integer value 3D
global PA; %Path Matrix with arrows 3D
global PE; %Information about Density near to the Exits
global PP; %Outputflow for each exit for all loops
global PD; %Outputflow for each exit
global ThreshNear; %for Room 1 for 2 exit aside
global ThreshDens; %Threshold for Density at exit cones
global ExitConeR; %Length of the exit corners
global AgWin; %Velocity distribution
global T; %Sorted Persons depending on time
global TT; %For each time numbers of people
global MyNorm; %Norming factor for Velocity Distribution
global NrExits; %Number of Exits
global NumVelo; %Number of Velocities
global Sigma; %Derivation factor for Velocity Distribution

%%
%Code
%Initialize
[m,n] = size(P(:,:,1));
PE = zeros(m,n,2+LimitLoop);
PP = zeros(m,n,LimitLoop);
l = ExitConeR; %length of the cones
for k=0:1-l
    if (find(P(:,:,end)==MyMax-k))
        [x,y] = find(P(:,:,end)==MyMax-k);
        for i=1:length(x)
            if (PE(x(i),y(i),2) < l-k)
                PE(x(i),y(i),2) = l-k;
                PE(x(i),y(i),1) = 1;
            end
        end
    end
end

function AgentUpdate(xM,yM)
%Move Agent to next place
%call function to search next position
%erase last position and shows with an arrows where he cames from
%%
%Globals
global MyPause;             %Activate pause in Simulation
global scrsz;               %Your actual Screen Size
global PlotData;            %Activate the plot functions [a,b]
global ActRand;             %Activate the Random function in Permutation and
GetNExtMove
global PFS;                 %Plot full screen
global ActRoom;             %Active Room Nr X

global MyCounter;           %Main Loop Counter
global PECounter;           %Counter for PE Matrix
global PPCounter;           %Counter for PP Matrix
global LimitLoop;           %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit;    %Define Maximum Person Density per Unit
global MaxOutFlow;          %Define how much people can go out at one time
global DeleteExit1;         %Define the exit to be deleted(0) in room 1
global DeleteExit2;         %Define the exit to be deleted(0) in room 2
global MyMax;               %Biggest value for the path search
global N;                   %Amount of people
global A;                   %Agent Matrix 4D
global R;                   %Room Matrix 3D
global P;                   %Path Matrix with integer value 3D
global PA;                  %Path Matrix with arrows 3D
global PE;                  %Information about Density near to the Exits
global PP;                  %Outputflow for each exit for all loops
global PD;                  %Outputflow for each exit
global ThreshNear;          %for Room 1 for 2 exit aside
global ThreshDens;          %Threshold for Density at exit cones
global ExitConeR;           %Length of the exit corners
global AgWin;               %Velocity distribution
global T;                   %Sorted Persons depending on time
global TT;                  %For each time numbers of people
global MyNorm;              %Norming factor for Velocity Distribution
global NrExits;             %Number of Exits
global NumVelo;             %Number of Velocities
global Sigma;               %Derivation factor for Velocity Distribution
"
%%%Code
    %if there is an Person try to update position
    for g=1:MaxPersonPerUnit
        if (A(xM,yM,1,g) == 1) && (mod(MyCounter,A(xM,yM,8,g)) == 0)
            [xN,yN,temp,MyUp,gg] = GetAgentMove(xM,yM,g,A(xM,yM,5,g));
            l = length(A(1,1,:,g));
            %if he is next to the exit, he will be erased
            if (temp == MyMax) && (g<=MaxOutFlow)
                %Save the number of loop needed in the first position
                R(real(A(xM,yM,6,g)),imag(A(xM,yM,6,g)),4) = A(xM,yM,7,g);
                A(xM,yM,:,g) = zeros(1,1,l);
                PP(xM,yM,PPCounter) = PP(xM,yM,PPCounter)+1;    %for outputflow plot
            end
            %normal update
            elseif (temp == MyMax) && (g>MaxOutFlow) %shift down
                A(xM,yM,:,g-MaxOutFlow) = A(xM,yM,:,g);
                A(xM,yM,:,g) = zeros(1,1,l);
            end
        end
        %Wait Counter
elseif (xM-xN == 0) && (yM-yN == 0)
A(xM,yM,4,g) = A(xM,yM,4,g)+1;
mn = 1;
%Layer down shift
while (mn<g)
    if A(xM,yM,1,mn) == 0
        A(xM,yM,:,mn) = A(xM,yM,:,g);
        A(xM,yM,:,g) = zeros(1,1,l);
        mn = g;
    end
    mn = mn+1;
end
end
end
end
end
end

function CorrRoom1()
%Hard correction on Room ETA F5
% this Correction is for optimising the crowd distribution
% it’s because 2 entrance are next to each other
% on a room with no exit next to each other it’s useless only the exit
% delete part will be still needed
%
%Globals
global MyPause;             %Activate pause in Simulation
global scrsz;               %Your actual Screen Size
global PlotData;            %Activate the plot functions [a,b]
global ActRand;             %Activate the Random function in Permutation and
GetNExtMove
global PFS;                 %Plot full screen
global ActRoom;             %Active Room Nr X
global MyCounter;           %Main Loop Counter
global PECounter;           %Counter for PE Matrix
global PPCounter;           %Counter for PP Matrix
global LimitLoop;           %Set a limit for the maximum numbers of loops
global MaxPersonPerUnit;    %Define Maximum Person Density per Unit
global MaxOutFlow;          %Define how much people can go out at one time
global DeleteExit1;         %Define the exit to be deleted(0) in room 1
global DeleteExit2;         %Define the exit to be deleted(0) in room 2
global MyMax;               %Biggest value for the path search
global N;                   %Amount of people
global A;                   %Agent Matrix 4D
global R;                   %Room Matrix 3D
global P;                   %Path Matrix with integer value 3D
global PA;                  %Path Matrix with arrows 3D
global PE;                  %Information about Density near to the Exits
global PP;                  %Outputflow for each exit for all loops
global PD;                  %Outputflow for each exit
global ThreshNear;          %for Room 1 for 2 exit aside
global ThreshDens;          %Threshold for Density at exit cones
global ExitConer;           %Length of the exit corners
global AgWin;               %Velocity distribution
global T;                   %Sorted Persons depending on time
global TT;                  %For each time numbers of people
global MyNorm;              %Norming factor for Velocity Distribution
global NrExits;             %Number of Exits
global NumVelo;             %Number of Velocities
global Sigma;               %Derivation factor for Velocity Distribution
%
%Code
%Switching layers to have it in the same order than the exits
temp = P(:,:,1);
P(:,:,1) = P(:,:,3);
P(:,:,3) = temp;

%Delete the exits
if DeleteExit1(1) == 0
P(:,:,1) = 0.*P(:,:,1);
end
if DeleteExit1(2) == 0
P(:,:,2) = 0.*P(:,:,2);
end
if DeleteExit1(3) == 0
P(:,:,3) = 0.*P(:,:,3);
end
if DeleteExit1(4) == 0
P(:,:,4) = 0.*P(:,:,4);
end
if DeleteExit1(5) == 0
P(:,:,5) = 0.*P(:,:,5);
end
if DeleteExit1(6) == 0
P(:,:,6) = 0.*P(:,:,6);
end
if DeleteExit1(7) == 0
P(:,:,7) = 0.*P(:,:,7);
end
if DeleteExit1(8) == 0
P(:,:,8) = 0.*P(:,:,8);
end
if DeleteExit1(9) == 0
P(:,:,9) = 0.*P(:,:,9);
end
if DeleteExit1(10) == 0
P(:,:,10) = 0.*P(:,:,10);
end
if DeleteExit1(11) == 0
P(:,:,11) = 0.*P(:,:,11);
end
if DeleteExit1(12) == 0
P(:,:,12) = 0.*P(:,:,12);
end

%Optimising path for two exits next to each other
[m,n] = size(P(:,:,1));
vec = [1,3,5,7,9,11];
for i=1:6
    for j=1:m
        for k=1:n
            P(j,k,vec(i)) = max(P(j,k,vec(i):vec(i)+1));
            P(j,k,vec(i)+1) = max(P(j,k,vec(i):vec(i)+1));
        end
    end
end

%Prepare Matrix for making lose priority to people between banks
R(:,:,5) = 0.*P(:,:,1);
[x,y] = find(R(:,:,1)==1);
for i=1:length(x)
R(x(i),y(i),5) = R(x(i),y(i),1);
end
[x,y] = find(R(:,:,1)==1);
for i=1:length(x)
    R(x(i),y(i),5) = R(x(i),y(i),1);
end
end

function CorrRoom2()
% Hard correction on Room ETA F5
% delete part will be still needed

%Globals
global MyPause;             %Activate pause in Simulation
global scrsz;               %Your actual Screen Size
global PlotData;            %Activate the plot functions [a,b]
global ActRand;             %Activate the Random function in Permutation and
GetNEXTMove
global PFS;                 %Plot full screen
global ActRoom;             %Active Room Nr X

global MyCounter;           %Main Loop Counter
global PECounter;           %Counter for PE Matrix
global PPCounter;           %Counter for PP Matrix
global LimitLoop;           %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit;    %Define Maximum Person Density per Unit
global MaxOutFlow;          %Define how much people can go out at one time
global DeleteExit1;         %Define the exit to be deleted(0) in room 1
global DeleteExit2;         %Define the exit to be deleted(0) in room 2
global MyMax;               %Biggest value for the path search
global N;                   %Amount of people
global A;                   %Agent Matrix 4D
global R;                   %Room Matrix 3D
global P;                   %Path Matrix with integer value 3D
global PA;                  %Path Matrix with arrows 3D
global PE;                  %Information about Density near to the Exits
global PP;                  %Outputflow for each exit for all loops
global PD;                  %Outputflow for each exit
global ThreshNear;          %for Room 1 for 2 exit aside
global ThreshDens;          %Threshold for Density at exit cones
global ExitConer;           %Length of the exit corners
global AgWin;               %Velocity distribution
global T;                   %Sorted Persons depending on time
global TT;                  %For each time numbers of people
global MyNorm;              %Nroming factor for Velocity Distribution
global NrExits;             %Number of Exits
global NumVelo;             %Number of Velocities
global Sigma;               %Derivation factor for Velocity Distribution

%Code
if DeleteExit2(1) == 0
    P(:,:,1) = 0.*P(:,:,1);
end
if DeleteExit2(2) == 0
    P(:,:,2) = 0.*P(:,:,2);
end
if DeleteExit2(3) == 0
P(:,:,3) = 0.*P(:,:,3);
end
if DeleteExit2(4) == 0
P(:,:,4) = 0.*P(:,:,4);
end
end

function CounterUpdate()
% Updates by every person the loop counter

%Globals
global MyPause;             % Activate pause in simulation
global scrsz;               % Your actual screen size
global PlotData;            % Activate the plot functions [a,b]
global ActRand;             % Activate the random function in permutation and
                           % GetNExtMove
global PFS;                 % Plot full screen
global ActRoom;             % Active room nr x
global MyCounter;           % Main loop counter
global PECounter;           % Counter for PE matrix
global PPCounter;           % Counter for PP matrix
global LimitLoop;           % Set a limit for the maximum numbers of loops

global MaxPersonPerUnit;    % Define maximum person density per unit
global MaxOutFlow;          % Define how much people can go out at one time
global DeleteExit1;         % Define the exit to be deleted(0) in room 1
global DeleteExit2;         % Define the exit to be deleted(0) in room 2
global MyMax;               % Biggest value for the path search
global N;                   % Amount of people
global A;                   % Agent matrix 4D
global R;                   % Room matrix 3D
global P;                   % Path matrix with integer value 3D
global PA;                  % Path matrix with arrows 3D
global PE;                  % Information about density near to the exits
global PP;                  % Outputflow for each exit for all loops
global PD;                  % Outputflow for each exit
global ThreshNear;          % For room 1 for 2 exit aside
global Thresholds;          % Threshold for density at exit cones
global ExitConeR;           % Length of the exit corners
% global AgWin;              % Velocity distribution
% global T;                  % Sorted persons depending on time
% global TT;                 % For each time numbers of people
% global MyNorm;             % Norming factor for velocity distribution
% global NrExits;            % Number of exits
% global NumVelo;            % Number of velocities
% global Sigma;              % Derivation factor for velocity distribution

% Code
ll = length(A(1,1,1,:));
for k=1:ll
  if find(A(:,:,1,k)==1)
    [x,y] = find(A(:,:,1,k)==1);
    for i=1:length(x)
      A(x(i),y(i),7,k) = A(x(i),y(i),7,k)+1;
    end
  end
end
function GetAgentDensity()
%Calculate the amount of people situate in the exit cones,
%the evolution is saved in the matrix PE

%Globals
global MyPause; %Activate pause in Simulation
global scrsz; %Your actual Screen Size
global PlotData; %Activate the plot functions [a,b]
global ActRand; %Activate the Random function in Permutation and GetNextMove
global PFS; %Plot full screen
global ActRoom; %Active Room Nr X

global MyCounter; %Main Loop Counter
global PECounter; %Counter for PE Matrix
global PPCounter; %Counter for PP Matrix
global LimitLoop; %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit; %Define Maximum Person Density per Unit
global MaxOutFlow; %Define how much people can go out at one time
global DeleteExit1; %Define the exit to be deleted(0) in room 1
global DeleteExit2; %Define the exit to be deleted(0) in room 2
global MyMax; %Biggest value for the path search
global N; %Amount of people
global A; %Agent Matrix 4D
global R; %Room Matrix 3D
global P; %Path Matrix with integer value 3D
global PA; %Path Matrix with arrows 3D
global PE; %Information about Density near to the Exits
global PP; %Outputflow for each exit for all loops
global PD; %Outputflow for each exit
global ThreshNear; %for Room 1 for 2 exit aside
global ThreshDens; %Threshold for Density at exit cones
global ExitConeR; %Length of the exit corners
global AgWin; %Velocity distribution
global T; %Sorted Persons depending on time
global TT; %For each time numbers of people
global MyNorm; %Norming factor for Velocity Distribution

%Code
PD(1:NrExits) = 0;
for i=1:NrExits
    AA = PE(:,:,1,1:5,m); %analyze only exit cones
    if (AA==i)
        AA = PE(:,:,1,1:5,m);
        [x,y] = find(AA==i);
        for h=1:length(x)
            PD(i) = PD(i) + PD(i)+(1👀AA(x(h),y(h)))
        end
    end
end

if ActRoom == 1
    PD(1) = PD(1)+PD(2);
    PD(2) = PD(1)+PD(2);
PD(3) = PD(3)+PD(4);
PD(4) = PD(3)+PD(4);
PD(5) = PD(5)+PD(6);
PD(6) = PD(5)+PD(6);
PD(7) = PD(7)+PD(8);
PD(8) = PD(7)+PD(8);
PD(9) = PD(9)+PD(10);
PD(10) = PD(9)+PD(10);
PD(11) = PD(11)+PD(12);
PD(12) = PD(11)+PD(12);
end

PE(:,:,PECounter+1) = PE(:,:,PECounter).*sum(A(:,:,1,:),4);
PECounter = PECounter + 1;
end

function [xN,yN,temp,MyUp,gg] = GetAgentMove(x,y,Layer,Exit)
%Checks the density in the exits cones only if not situated in one
%then search the best next position
%TreshNear equaliz the flow between 2 exits next to each other
%%
%Globals
global MyPause; %Activate pause in Simulation
global scrsz; %Your actual Screen Size
global PlotData; %Activate the plot functions [a,b]
global ActRand; %Activate the Random function in Permutation and GetNExtMove
global PFS; %Plot full screen
global ActRoom; %Active Room Nr X
global MyCounter; %Main Loop Counter
global PECounter; %Counter for PE Matrix
global PPCounter; %Counter for PP Matrix
global LimitLoop; %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit; %Define Maximum Person Density per Unit
global MaxOutFlow; %Define how much people can go out at one time
global DeleteExit1; %Define the exit to be deleted(0) in room 1
global DeleteExit2; %Define the exit to be deleted(0) in room 2
global MyMax; %Biggest value for the path search
global N; %Amount of people
global A; %Agent Matrix 4D
global R; %Room Matrix 3D
global P; %Path Matrix with integer value 3D
global PA; %Path Matrix with arrows 3D
global PE; %Information about Density near to the Exits
global PP; %Outputflow for each exit for all loops
global PD; %Outputflow for each exit
global ThreshNear; %for Room 1 for 2 exit aside
global ThreshDens; %Threshold for Density at exit cones
global ExitConeR; %Length of the exit corners
global AgWin; %Velocity distribution
global T; %Sorted Persons depending on time
global TT; %For each time numbers of people
global MyNorm; %Norming factor for Velocity Distribution
global NrExits; %Number of Exits
global NumVelo; %Number of Velocities
global Sigma; %Derivation factor for Velocity Distribution
%%
%Code
%if waiting too long go back to nearest exit
if A(x,y,4,Layer) >2
    A(x,y,5,Layer) = P(x,y,end-1);
    A(x,y,4,Layer) = 0;
end
%if not in exit cone check density
if PE(x,y,1,1) == 0
    %if between seats break;
    if ActRoom == 1
        bobo = (sum(sum(R(x-1:x+1,y-1:y+1,5)))>0);
    else
        bobo = 1;
    end
    if (PD(Exit) >ThreshDens) && bobo
        a= 1;
        Exit = GetOnBetterExit(x,y,Layer,Exit,a);
    end
    if (PD(Exit) >ThreshNear) && (ActRoom == 1)
        a=2;
        Exit = GetOnBetterExit(x,y,Layer,Exit,a);
    end
end
%search next position
[xN,yN,temp,MyUp,gg] = GetNextMove(x,y,Layer,Exit);
end

function  GetNearestExit()
%Save Active Exit Layer in Agent Matrix
%for the first time caled it saves the Layer for the nearest exit
%%
%Globals
%Global MyPause;             %Activate pause in Simulation
%Global scrsz;               %Your actual Screen Size
%Global PlotData;            %Activate the plot functions [a,b]
%Global ActRand;             %Activate the Random function in Permutation and
GetNExtMove
%Global PFS;                 %Plot full screen
%Global ActRoom;             %Active Room Nr X
%Global MyCounter;           %Main Loop Counter
%Global PECounter;           %Counter for PE Matrix
%Global PPFCounter;          %Counter for PP Matrix
%Global LimitLoop;           %Set a limit for the maximum numbers of loops
%Global MaxPersonPerUnit;    %Define Maximum Person Density per Unit
%Global MaxOutFlow;          %Define how much people can go out at one time
%Global DeleteExit1;         %Define the exit to be deleted(0) in room 1
%Global DeleteExit2;         %Define the exit to be deleted(0) in room 2
%Global MyMax;               %Biggest value for the path search
%Global N;                   %Amount of people
%Global A;                   %Agent Matrix 4D
%Global R;                   %Room Matrix 3D
%Global P;                   %Path Matrix with integer value 3D
%Global PA;                  %Path Matrix with arrows 3D
%Global PE;                  %Information about Density near to the Exits
%Global PP;                  %Outputflow for each exit for all loops
global PD;                  %Outputflow for each exit
global ThreshNear;          %for Room 1 for 2 exit aside
global ThreshDens;          %Threshold for Density at exit cones
global ExitConeR;           %Length of the exit corners
global AgWin;               %Velocity distribution
global T;                   %Sorted Persons depending on time
global TT;                  %For each time numbers of people
global MyNorm;              %Norming factor for Velocity Distribution
global NrExits;             %Number of Exits
global NumVelo;             %Number of Velocities
global Sigma;               %Derivation factor for Velocity Distribution

%%
%Code
[x,y] = find(A(:,:,1,1)==1);
for i=1:length(x)
    A(x(i),y(i),5,1) = P(x(i),y(i),end)
end
end

function [xN,yN,temp,MyUp,gg] = GetNextMove(x,y,Layer,Exit)
%Search a path following the arrows defined for the active
%exit layers, look first in front and side ways and then backwards
%%
%Globals
global MyPause;             %Activate pause in Simulation
global scrsz;               %Your actual Screen Size
global PlotData;            %Activate the plot functions [a,b]
global ActRand;             %Activate the Random function in Permutation and
GetNextMove
global PFS;                 %Plot full screen
global ActRoom;             %Active Room Nr X

global MyCounter;           %Main Loop Counter
global PECounter;           %Counter for PE Matrix
global PPCounter;           %Counter for PP Matrix
global LimitLoop;           %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit;    %Define Maximum Person Density per Unit
global MaxOutFlow;          %Define how much people can go out at one time
global DeleteExit1;         %Define the exit to be deleted(0) in room 1
global DeleteExit2;         %Define the exit to be deleted(0) in room 2
global MyMax;               %Biggest value for the path search
global N;                   %Amount of people
global A;                   %Agent Matrix 4D
global R;                   %Room Matrix 3D
global P;                   %Path Matrix with integer value 3D
global PA;                  %Path Matrix with arrows 3D
global PE;                  %Information about Density near to the Exits
global PP;                  %Outputflow for each exit for all loops

global PD;                  %Outputflow for each exit
global ThreshNear;          %for Room 1 for 2 exit aside
global ThreshDens;          %Threshold for Density at exit cones
global ExitConeR;           %Length of the exit corners
global AgWin;               %Velocity distribution
global T;                   %Sorted Persons depending on time
global TT;                  %For each time numbers of people
global MyNorm;              %Norming factor for Velocity Distribution
global NrExits;             %Number of Exits
global NumVelo;             %Number of Velocities
global Sigma;  %Derivation factor for Velocity Distribution
%
%Code
temp = P(x,y,Exit);
MyUp = [0,0];
PathSearching = 1;
gg = 1;
%define search order
SearchOrder = [1,2,3,4,5,6,7,8];
Direc = PA(x,y,Exit);
if ActRand == 1
    MyRan = randi([0,1],1);  %if MyRan ==1 first check left
else
    MyRan = 0;
end
%Define Search order
switch Direc
    case 1
        if MyRan == 1
            SearchOrder = [1,5,2,8,4,7,3,6];
        else
            SearchOrder = [1,8,4,5,2,6,3,7];
        end
    case 1i
        if MyRan == 1
            SearchOrder = [2,6,3,5,1,8,4,7];
        else
            SearchOrder = [2,5,1,6,3,7,4,8];
        end
    case -1
        if MyRan == 1
            SearchOrder = [3,7,4,6,2,5,1,8];
        else
            SearchOrder = [3,6,2,7,4,8,1,5];
        end
    case -1i
        if MyRan == 1
            SearchOrder = [4,8,1,7,3,6,2,5];
        else
            SearchOrder = [4,7,3,8,1,5,2,6];
        end
    case 1+1i
        if MyRan == 1
            SearchOrder = [5,2,6,1,8,4,7,3];
        else
            SearchOrder = [5,1,8,2,6,3,7,4];
        end
    case -1+1i
        if MyRan == 1
            SearchOrder = [6,3,7,2,5,1,9,4];
        else
            SearchOrder = [6,2,5,3,7,4,8,1];
        end
    case -1-1i
        if MyRan == 1
            SearchOrder = [7,4,8,3,6,2,5,1];
        else
            SearchOrder = [7,3,6,4,8,1,5,2];
        end
    case 1-1i
        if MyRan == 1
            SearchOrder = [8,7,3,6,2,5,1,9];
        else
            SearchOrder = [8,6,1,9,4,7,3,2];
        end
end
SearchOrder = [8,1,5,4,7,3,6,2];
else
    SearchOrder = [8,4,7,1,5,2,6,3];
end

%%% Find next move according to search order priority
%%% if no move is found, then wait
h = 1;
while (h<6) && (PathSearching == 1)
    [MyUp,PathSearching,gg] = SearchMe(x,y,Exit,temp,gg,MyUp,PathSearching,SearchOrder(h));
    h = h+1;
end
h = 6;
while (h<9) && (PathSearching == 1)
    [MyUp,PathSearching,gg] = SearchMe(x,y,Exit,temp,gg,MyUp,PathSearching,SearchOrder(h));
    h = h+1;
end

%%% if no way found with greater value, search for the same value
if PathSearching == 1
    temp = temp-1;
    h = 1;
    while (h<6) && (PathSearching == 1)
        [MyUp,PathSearching,gg] = SearchMe(x,y,Exit,temp,gg,MyUp,PathSearching,SearchOrder(h));
        h = h+1;
    end
    h = 6;
    while (h<9) && (PathSearching == 1)
        [MyUp,PathSearching,gg] = SearchMe(x,y,Exit,temp,gg,MyUp,PathSearching,SearchOrder(h));
        h = h+1;
    end
    temp = temp+1;
end

aa = [x,y]+MyUp;
MyUp = [sign(MyUp(1)),sign(MyUp(2))]; %for saving direction in C
xN = aa(1);
yN = aa(2);

function Exit = GetOnBetterExit(x,y,Layer,Exit,a)
%%% Search the best alternative to avoid the crowd near the exits
%%% The second part for the near exits have to be only used for room 1
%%% %Globals
global MyPause; %Activate pause in Simulation
global scrsz; %Your actual Screen Size
global PlotData; %Activate the plot functions [a,b]
global ActRand; %Activate the Random function in Permutation and
GetNExtMove
global PFS; %Plot full screen
global ActRoom; %Active Room Nr X

global MyCounter; %Main Loop Counter
global PECounter;           %Counter for PE Matrix
global PPCounter;           %Counter for PP Matrix
global LimitLoop;           %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit;    %Define Maximum Person Density per Unit
global MaxOutFlow;          %Define how much people can go out at one time

global DeleteExit1;         %Define the exit to be deleted(0) in room 1

global DeleteExit2;         %Define the exit to be deleted(0) in room 2

global MyMax;               %Biggest value for the path search

global N;                   %Amount of people

global A;                   %Agent Matrix 4D

global R;                   %Room Matrix 3D

global P;                   %Path Matrix with integer value 3D

global PA;                  %Path Matrix with arrows 3D

global PE;                  %Information about Density near to the Exits

global PP;                  %Outputflow for each exit for all loops

global PD;                  %Outputflow for each exit

global ThreshNear;          %for Room 1 for 2 exit aside

global ThreshDens;          %Threshold for Density at exit cones

global ExitConesR;          %Length of the exit corners

global AgWin;               %Velocity distribution

global T;                   %Sorted Persons depending on time

global TT;                  %For each time numbers of people

global MyNorm;              %Normalization factor for Velocity Distribution

global NrExits;             %Number of Exits

global NumVelo;             %Number of Velocities

global Sigma;               %Derivation factor for Velocity Distribution

%%
%Code
Pt = P;
temp = P(x,y,Exit);
Cor = Exit;
if a ==1
    while PD(Cor) > ThreshDens
        Pt(x,y,Cor) = 0;
        [Val,Cor] = max(Pt(x,y,1:end-2));
        while Val >= temp
            Pt(x,y,Cor) = 0;
            [Val,Cor] = max(Pt(x,y,1:end-2));
        end
    end
    Exit = Cor;
    A(x,y,5,Layer) = Cor;
else
    if (Exit==1)
        if PD(Exit)>PD(2)
            Exit = 2;
            A(x,y,5,Layer) = 2;
        end
    elseif (Exit==2)
        if PD(Exit)>PD(1)
            Exit = 1;
            A(x,y,5,Layer) = 1;
        end
    elseif (Exit==3)
        if PD(Exit)>PD(4)
            Exit = 4;
            A(x,y,5,Layer) = 4;
        end
    elseif (Exit==4)
        if PD(Exit)>PD(3)
Exit = 3;
A(x,y,5,Layer) = 3;
end
elseif (Exit==5)
  if PD(Exit)>PD(6)
    Exit = 6;
    A(x,y,5,Layer) = 6;
  end
elseif (Exit==6)
  if PD(Exit)>PD(5)
    Exit = 5;
    A(x,y,5,Layer) = 5;
  end
elseif (Exit==7)
  if PD(Exit)>PD(8)
    Exit = 8;
    A(x,y,5,Layer) = 8;
  end
elseif (Exit==8)
  if PD(Exit)>PD(7)
    Exit = 7;
    A(x,y,5,Layer) = 7;
  end
elseif (Exit==9)
  if PD(Exit)>PD(10)
    Exit = 10;
    A(x,y,5,Layer) = 10;
  end
elseif (Exit==10)
  if PD(Exit)>PD(9)
    Exit = 9;
    A(x,y,5,Layer) = 9;
  end
elseif (Exit==11)
  if PD(Exit)>PD(12)
    Exit = 12;
    A(x,y,5,Layer) = 12;
  end
elseif (Exit==12)
  if PD(Exit)>PD(11)
    Exit = 11;
    A(x,y,5,Layer) = 11;
  end
end
end

function LoadRoom(a)
%Load Room Variable in Workspace
%a==0 load from csv || Take a long time
%specify in which folder is situated the file
%example: room('Documents/.../Room=Room.csv')
%a==1 load the room 1 from mat file P.mat and R.mat
%a==2 load the room 2 from mat file P2.mat and R2.mat
%
%Globals
global MyPause; %Activate pause in Simulation
global scrsz; %Your actual Screen Size
global PlotData; %Activate the plot functions [a,b]
global ActRand; %Activate the Random function in Permutation and
GetNExtMove
global PFS; %Plot full screen
global ActRoom; %Active Room Nr X

global MyCounter; %Main Loop Counter
global PECounter; %Counter for PE Matrix
global PPCounter; %Counter for PP Matrix
global LimitLoop; %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit; %Define Maximum Person Density per Unit
global MaxOutFlow; %Define how much people can go out at one time
global DeleteExit1; %Define the exit to be deleted(0) in room 1
global DeleteExit2; %Define the exit to be deleted(0) in room 2
global MyMax; %Biggest value for the path search
global N; %Amount of people
global A; %Agent Matrix 4D
global R; %Room Matrix 3D

global P; %Path Matrix with integer value 3D
global PA; %Path Matrix with arrows 3D
global PE; %Information about Density near to the Exits
global PP; %Outputflow for each exit for all loops
global PD; %Outputflow for each exit
global ThreshNear; %for Room 1 for 2 exit aside
global ThreshDens; %Threshold for Density at exit cones
global ExitConeR; %Length of the exit corners

global AgWin; %Velocity distribution
global T; %Sorted Persons depending on time
global TT; %For each time numbers of people
global MyNorm; %Norming factor for Velocity Distribution
global NrExits; %Number of Exits
global NumVelo; %Number of Velocities
global Sigma; %Derivation factor for Velocity Distribution
%
%Code
if a == 0
    [R, P] = room();
elseif a == 1
    load R
    load P
    %delete the wanted exits
    CorrRoom1();
elseif a == 2
    load R2
    load P2
    %delete wanted exits
    CorrRoom2();
end
%
%Initializing
R(:,:,2:4) = 0;
MyMax = max(max(P(:,:,end)));
temp = P(:,:,end);
NrExits = numel(temp(temp == MyMax));
%Calculate shortest Path for every position
ShortestPathLink();
PD(1:NrExits) = 0; %Initialize
AddExitCones(); %Prepare Exit cones
PlacePathArrows(); %Calculate and save Arrows for the Pathes
%
%Plots
%Room
if PlotData(1) == 1
    figure(1)
    if PFS
        set(gcf,'Position',scrsz)
    end
    PlotRoom(R(:,:,1));
end
end

function [xN,yN] = Permutation(xO,yO)
%Permutate randomly one vector and adjust the second
%xO,yO have to be the same length
%
%Globals
global ActRand;             %Activate the Random function in Permutation and
GetNExtMove
%
%Code
l = length(xO);

if ActRand == 1
    vR = randperm(l);
else
    vR = linspace(1,l,l);
end

for i=1:l
    xN(i) = xO(vR(i));
    yN(i) = yO(vR(i));
end
end

function PlaceAgentIni(PlaceStyle,n)
%Give out the crowd initial state of the room A and the amount of people
%PlaceStyle give differents way of placement
%PlaceStyle==1 is the full version of the room
%PlaceStyle==2 is a random placement of maximum n people
%
%Globals
global MyPause;             %Activate pause in Simulation
global scrsz;               %Your actual Screen Size
global PlotData;            %Activate the plot functions [a,b]
global ActRand;             %Activate the Random function in Permutation and
GetNExtMove
global PFS;                 %Plot full screen
global ActRoom;             %Active Room Nr X

global MyCounter;           %Main Loop Counter
global PECounter;           %Counter for PE Matrix
global PPCCounter;          %Counter for PP Matrix
global LimitLoop;           %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit;    %Define Maximum Person Density per Unit
global MaxOutFlow;          %Define how much people can go out at one time
global DeleteExit1;         %Define the exit to be deleted(0) in room 1
global DeleteExit2; %Define the exit to be deleted(0) in room 2
global MyMax; %Biggest value for the path search
global N; %Amount of people
global A; %Agent Matrix 4D
global R; %Room Matrix 3D
global P; %Path Matrix with integer value 3D
global PA; %Path Matrix with arrows 3D
global PE; %Information about Density near to the Exits
global PP; %Outputflow for each exit for all loops
global PD; %Outputflow for each exit
global ThreshNear; %for Room 1 for 2 exit aside
global ThreshDens; %Threshold for Density at exit cones
global ExitConeR; %Length of the exit corners
global AgWin; %Velocity distribution
global T; %Sorted Persons depending on time
global TT; %For each time numbers of people
global MyNorm; %Norming factor for Velocity Distribution
global NrExits; %Number of Exits
global NumVelo; %Number of Velocities
global Sigma; %Derivation factor for Velocity Distribution

%%
%Code
%Find free places (R(...)==0)
[m1,l1] = size(R(:,:,1));
[m,l] = find(R(:,:,1)==0);
[m2,l2] = find(R(:,:,1)==1);
A = zeros(m1,l1,7,MaxPersonPerUnit);

%%
%Fill up
switch PlaceStyle
    case 1
        %whole room
        for j=1:length(m)
            A(m(j),l(j),1,1) = 1;
            A(m(j),l(j),6,1) = m(j)+sqrt(1)*l(j); %save first position in C
        end
    case 2
        %random
        ff = length(m);
        vR = randperm(ff);
        for i=1:ff
            mt(i) = m(vR(i));
            lt(i) = l(vR(i));
        end
        m = mt;
        l = lt;
        for j=1:n
            A(m(j),l(j),1,1) = 1;
            A(m(j),l(j),6,1) = m(j)+sqrt(-1)*l(j); %save first position in C
        end
    case 3
        %with path
        for j=1:length(m)
            A(m(j),l(j),1,1) = 1;
            A(m(j),l(j),6,1) = m(j)+sqrt(-1)*l(j); %save first position in C
        end
        for j=1:length(m2)
            A(m2(j),l2(j),1,1) = 1;
A(m2(j),l2(j),6,1) = m2(j)+sqrt(-1)*l2(j); %save first position in C
end

function PlaceAgentVelocity(NV,WinN,Q)
%Make a distribution of velocities
%NV are the different velocities
%Win is the distribution style
%
%Globals
global MyPause; %Activate pause in Simulation
global scrsz; %Your actual Screen Size
global PlotData; %Activate the plot functions [a,b]
global ActRand; %Activate the Random function in Permutation and GetNExtMove
global PFS; %Plot full screen
global ActRoom; %Active Room Nr X

global MyCounter; %Main Loop Counter
global PECounter; %Counter for PE Matrix
global PPCounter; %Counter for PP Matrix
global LimitLoop; %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit; %Define Maximum Person Density per Unit
global MaxOutFlow; %Define how much people can go out at one time
global DeleteExit1; %Define the exit to be deleted(0) in room 1
global DeleteExit2; %Define the exit to be deleted(0) in room 2
global MyMax; %Biggest value for the path search

global N; %Amount of people
global A; %Agent Matrix 4D
global R; %Room Matrix 3D

global P; %Path Matrix with integer value 3D
global PA; %Path Matrix with arrows 3D
global PE; %Information about Density near to the Exits

global PP; %Outputflow for each exit for all loops

global PD; %Outputflow for each exit

global ThreshNear; %for Room 1 for 2 exit aside

global ThreshDens; %Threshold for Density at exit cones

global ExitConeR; %Length of the exit corners

global AgWin; %Velocity distribution

global T; %Sorted Persons depending on time

global TT; %For each time numbers of people

global MyNorm; %Norming factor for Velocity Distribution

global NrExits; %Number of Exits

global NumVelo; %Number of Velocities

global Sigma; %Derivation factor for Velocity Distribution

%Code
if NV>2
    if WinN == 'gauss'
        Win = gausswin(NV,Q);
    elseif WinN == 'cheb'

%Code
end
Win = chebwin(NV,Q);
elseif WinN == 'kaiser'
    Win = kaiser(NV,Q);
elseif WinN == 'tukey'
    Win = tukeywin(NV,Q);
elser
    disp('ERROR666: No such Window')
    disp('default use : gauss')
    Win = gausswin(NV,Q);
end
WinNorm = Win./sum(Win);
AgWin = fix(N.*WinNorm);
AgWin(fix(length(AgWin)/2)+1) = AgWin(fix(length(AgWin)/2)+1)+N-sum(AgWin);
MyMod = [];
for i=1:NV
    for k=1:AgWin(i)
        MyMod = [MyMod;i];
    end
end
if find(A(:,:,1,1))
    [x,y] = find(A(:,:,1,1)==1);
    [x,y] = Permutation(x,y);
    for i=1:length(x);
        A(x(i),y(i),8,1) = MyMod(i);
    end
end
else
    AgWin = N;
    if find(A(:,:,1,1))
        [x,y] = find(A(:,:,1,1)==1);
        [x,y] = Permutation(x,y);
        for i=1:length(x);
            A(x(i),y(i),8,1) = 1;
        end
    end
end
if mod(NV,2) == 0
    MyNorm = NV/2+0.5;
else
    MyNorm = fix(NV/2)+1;
end
end

function PlacePathArrows()
%Calculate the direction to the nearest exit and save it in
%complex number to be plotted as arrows
%the layer order for PA is the same as P
%
%Globals
global MyPause;                 %Activate pause in Simulation
global scrsz;                   %Your actual Screen Size
global PlotData;               %Activate the plot functions [a,b]
global ActRand;                %Activate the Random function in Permutation and
GetNExtMove
global PFS;                    %Plot full screen
global ActRoom; %Active Room Nr X

global MyCounter; %Main Loop Counter
global PECounter; %Counter for PE Matrix
global PPCounter; %Counter for PP Matrix
global LimitLoop; %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit; %Define Maximum Person Density per Unit
global MaxOutFlow; %Define how much people can go out at one time
global DeleteExit1; %Define the exit to be deleted(0) in room 1
global DeleteExit2; %Define the exit to be deleted(0) in room 2
global MyMax; %Biggest value for the path search

global N; %Amount of people

global A; %Agent Matrix 4D

global R; %Room Matrix 3D

global P; %Path Matrix with integer value 3D

global PA; %Path Matrix with arrows 3D

global PE; %Information about Density near to the Exits

global PP; %Outputflow for each exit for all loops

global PD; %Outputflow for each exit

global ThreshNear; %for Room 1 for 2 exit aside

global ThreshDens; %Threshold for Density at exit cones

global ExitConeR; %Length of the exit corners

global AgWin; %Velocity distribution

global T; %Sorted Persons depending on time

global TT; %For each time numbers of people

global NumVelo; %Number of Velocities

global Sigma; %Derivation factor for Velocity Distribution

%%

%%
%Code

%%
PA = 0.*P; %m,n = size(PA(:,:,1));
temp = 0;
for h=1:length(P(1,1,1:end-2))
    for j=2:n-1
        for i=2:m-1
            k=1;
            if(R(i+k,j,1)==-2)
                while(R(i+k,j,1)==-2)
                    k=k+1;
                end
            end
            if (P(i,j,h) < P(i+k,j,h)) && (P(i,j,h) > 0) && (temp < P(i+k,j,h))
                temp = P(i+k,j,h);
                PA(i,j,h) = 1;
            end
            k=1;
            if(R(i,j+k,1)==-2)
                while(R(i,j+k,1)==-2)
                    k=k+1;
                end
            end
            if (P(i,j,h) < P(i,j+k,h)) && (P(i,j,h) > 0) && (temp < P(i,j+k,h))
                temp = P(i,j+k,h);
                PA(i,j,h) = 1;
            end
            k=1;
function PlotArrows(An)
%Arrow Plot of a 2D matrix with complex numbers
%shows the last direction
%
%Globals
global MyPause; %Activate pause in Simulation
global scrsz; %Your actual Screen Size
global PlotData; %Activate the plot functions [a,b]
global ActRand; %Activate the Random function in Permutation and GetNExtMove
global PFS; %Plot full screen
global ActRoom; %Active Room Nr X
global MyCounter; %Main Loop Counter
global PECounter; %Counter for PE Matrix
global PPCounter; %Counter for PP Matrix
global LimitLoop; %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit; %Define Maximum Person Density per Unit
global MaxOutFlow; %Define how much people can go out at one time
global DeleteExit1; %Define the exit to be deleted(0) in room 1
global DeleteExit2; %Define the exit to be deleted(0) in room 2
global MyMax; %Biggest value for the path search
global N; %Amount of people
global A; %Agent Matrix 4D
global R; %Room Matrix 3D
global P; %Path Matrix with integer value 3D
global PA; %Path Matrix with arrows 3D
global PE; %Information about Density near to the Exits
global PP; %Outputflow for each exit for all loops
global PD; %Outputflow for each exit
global ThreshNear; %for Room 1 for 2 exit aside
global ThreshDens;          %Threshold for Density at exit cones
global ExitConeR;           %Length of the exit corners
global AgWin;               %Velocity distribution
global T;                   %Sorted Persons depending on time
global TT;                  %For each time numbers of people
global MyNorm;              %Norming factor for Velocity Distribution
global NrExits;             %Number of Exits
global NumVelo;             %Number of Velocities
global Sigma;               %Derivation factor for Velocity Distribution

% Code
[x0,y0] = find(A(:,:,3,1) == 1);
[x1,y1] = find(A(:,:,3,1) == -1);
[x2,y2] = find(A(:,:,3,1) == -1i);
[x3,y3] = find(A(:,:,3,1) == 1i);
[x4,y4] = find(A(:,:,3,1) == 1+1i);
[x5,y5] = find(A(:,:,3,1) == 1-1i);
[x6,y6] = find(A(:,:,3,1) == -1+1i);
[x7,y7] = find(A(:,:,3,1) == -1-1i);
x = [x0;x1;x2;x3;x4;x5;x6;x7];
y = [y0;y1;y2;y3;y4;y5;y6;y7];
l = length(x);
if l > 0
    for k=1:l
        m(k) = imag(A(x(k),y(k),3,1));
        n(k) = real(A(x(k),y(k),3,1));
    end
    quiver(y,x,m',n')
end
hold on

function PlotRoom(R)
% plots a room Matrix R a bit fancy (hopefully)

% define identifiers for the several objects
markersize = 20;
if nargin == 1
    if find(R==0)
        Walls = -inf;
        wallmarker = 's';
        wallmarkersize = markersize;
        walllinestyle = 'none';
        wallcolor = [93 93 93]./256;
    end
    if find(R==0)
        [j, k] = find(R == Walls);
        plot(k,j,'marker',wallmarker,'markersize',markersize,'linestyle',walllinestyle,'color',wallcolor,'MarkerFaceColor',wallcolor);
        hold on;
        Paths = 1;
        pathmarker = 's';
        pathmarkersize = markersize;
pathlinestyle = 'none';
pathcolor = [88 201 61]./256;

[j, k] = find(R >= Paths);
plot(k,j,'marker','pathmarker','markersize','pathmarkersize',
'linestyle',pathlinestyle,'color',pathcolor,'MarkerFaceColor',pathcolor);
hold on;

Seats = 0;
seatsmarker = 's';
seatsmarkersize = markersize;
seatslinestyle = 'none';
seatscolor = [72 125 253]./256;

[j, k] = find(R == Seats);
plot(k,j,'marker',seatsmarker,'markersize',seatsmarkersize,
'linestyle',seatslinestyle,'color',seatscolor,'MarkerFaceColor',seatscolor);

Tables = -1;
tablesmarker = 's';
tablesmarkersize = markersize;
tableslinestyle = 'none';
tablescolor = [254 158 57]./256;

[j, k] = find(R == Tables);
plot(k,j,'marker',tablesmarker,'markersize',tablesmarkersize,
'linestyle',tableslinestyle,'color',tablescolor,'MarkerFaceColor',tablescolor);

Exits = inf;
exitsmarker = 's';
exitsmarkersize = markersize;
exitslinestyle = 'none';
exitscolor = [254 249 44]./256;
[j, k] = find(R == Exits);
plot(k,j,'marker',exitsmarker,'markersize',exitsmarkersize,
'linestyle',exitslinestyle,'color',exitscolor,'MarkerFaceColor',exitscolor);

Fillups = -2;
fillupsmarker = 's';
fillupsmarkersize = markersize;
fillupslinestyle = 'none';
fillupscolor = [254 93 104]./256;
[j, k] = find(R == Fillups);
plot(k,j,'marker',fillupsmarker,'markersize',fillupsmarkersize,
'linestyle',fillupslinestyle,'color',fillupscolor,'MarkerFaceColor',fillupscolor);

legend('Walls', 'Paths', 'Tables','Exits','Fills');
title(sprintf('plot of the room matrix. Exit value = %d', Exits));
else

Walls = -inf;
wallmarker = 's';
wallmarkersize = markersize;
walllinestyle = 'none';
wallcolor = [93 93 93]./256;
[j, k] = find(R == Walls);
plot(k,j,'marker',wallmarker,'markersize',wallmarkersize, 'linestyle',
walllinestyle,'color',wallcolor, 'MarkerFaceColor',wallcolor);
hold on;

Paths = 1;
pathmarker = 's';
pathmarkersize = markersize;
pathlinestyle = 'none';
pathcolor = [88 201 61]./256;

[j, k] = find(R >= Paths);
plot(k,j,'marker',pathmarker,'markersize',pathmarkersize, 'linestyle',
pathlinestyle,'color',pathcolor, 'MarkerFaceColor',pathcolor);
hold on;

Seats = 0;
seatsmarker = 's';
seatsmarkersize = markersize;
seatslinestyle = 'none';
seatscolor = [72 125 253]./256;

[j, k] = find(R == Seats);
plot(k,j,'marker',seatsmarker,'markersize',seatsmarkersize, 'linestyle',
seatslinestyle,'color',seatscolor,'MarkerFaceColor',seatscolor);

Tables = 1;
tablesmarker = 's';
tablesmarkersize = markersize;
tableslinestyle = 'none';
tablescolor = [254 158 57]./256;

[j, k] = find(R == Tables);
plot(k,j,'marker',tablesmarker,'markersize',tablesmarkersize, 'linestyle',
tableslinestyle,'color',tablescolor,'MarkerFaceColor',tablescolor);

Exits = max(R(:));
exitsmarker = 's';
exitsmarkersize = markersize;
exitslinestyle = 'none';
exitscolor = [254 249 44]./256;
[j, k] = find(R == Exits);
plot(k,j,'marker',exitsmarker,'markersize',exitsmarkersize, 'linestyle',
exitslinestyle,'color',exitscolor,'MarkerFaceColor',exitscolor);

Fillups = -2;
fillupsmarker = 's';
fillupsmarkersize = markersize;
fillupslinestyle = 'none';
fillupscolor = [254 93 104]./256;
[j, k] = find(R == Fillups);
plot(k,j,'marker',fillupsmarker,'markersize',fillupsmarkersize, 'linestyle',
fillupslinestyle,'color',fillupscolor,'MarkerFaceColor',fillupscolor);

legend('Walls', 'Paths', 'Tables','Exits','Fills');
title(sprintf('plot of the room matrix. Exit value = %d', Exits));
function P = PourPathValues(R)
% Fills a tensor P with path values from room matrix R
% P = PourPathValues(R)
% Detects exits by 'inf' and tries to "pour" the values through the paths
% get max possible value for a path (just to have enough overhead for
% filling the paths) = n*m

[m,n] = size(R);
maxval = m*n;
% locate every exit
[x,y] = find(R==inf);
P = zeros(m,n,length(x)); % tensor = matrix for each exit
for i=1:size(x,1);
P(x(i),y(i), i) = maxval;
P(1:end,1:end,i) = AddExit_v2(R, P(:,:,i), [x(i) y(i)], maxval-1, 0);
end
end

function PrintAgent(n)
% Plot the Room with actual Crowd placement and direction
% the first plot is the room(blue/red) with on it every position were are
% one or more people and the last direction of the person in Layer 1
% %
% %Globals
global MyPause; % Activate pause in Simulation
global scrsz; % Your actual Screen Size
global PlotData; % Activate the plot functions [a,b]
global ActRand; % Activate the Random function in Permutation and
GetNExtMove
global PFS; % Plot full screen
global ActRoom; % Active Room Nr X

global MyCounter; % Main Loop Counter
global PECounter; % Counter for PE Matrix
global PPCounter; % Counter for PP Matrix
global LimitLoop; % Set a limit for the maximum numbers of loops

global MaxPersonPerUnit; % Define Maximum Person Density per Unit
global MaxOutFlow; % Define how much people can go out at one time
global DeleteExit1; % Define the exit to be deleted(0) in room 1
global DeleteExit2; % Define the exit to be deleted(0) in room 2
global MyMax; % Biggest value for the path search
global N; % Amount of people
global A;                   %Agent Matrix 4D
global R;                   %Room Matrix 3D
global P;                   %Path Matrix with integer value 3D
global PA;                  %Path Matrix with arrows 3D
global PE;                  %Information about Density near to the Exits
global PP;                  %Outputflow for each exit for all loops
global PD;                  %Outputflow for each exit
global ThreshNear;          %for Room 1 for 2 exit aside
global ThreshDens;          %Threshold for Density at exit cones
global ExitConeR;           %Length of the exit corners
global AgWin;               %Velocity distribution
global T;                   %Sorted Persons depending on time
global TT;                  %For each time numbers of people
global MyNorm;              %Norming factor for Velocity Distribution
global NrExits;             %Number of Exits
global NumVelo;             %Number of Velocities
global Sigma;               %Derivation factor for Velocity Distribution

%%
%Code
figure(n);
clf;
if PFS
    scrsz = get(0,'Screensize')- [0 0 0 100];
    set(gcf,'Position',scrsz)
end
subplot(2,1,1)
surf(P(:,:,end))
hold on
set(gca,'FontSize',16)
spy(A(:,:,1,1),'y')
if N >0
    PlotArrows(A)
%legend('Room','People','Direction')
end
title('Agent Move : Red/Blue is Room Surface & yellow dot are agents')
xlabel('x')
ylabel('y')
hold off
subplot(2,1,2)
hold on
set(gca,'FontSize',16)
title('Agent Density')
xlabel('x')
ylabel('y')
zlabel('Person Per Unit')
hold off
h=bar3(sum(A(:,:,1,:),4));
for i = 1:numel(h)
    zData = get(h(i),'ZData');
    index = logical(kron(zData(2:6:end,2) == 0,ones(6,1)));
    zData(index,:) = nan;
    set(h(i),'ZData',zData);
end
[m,n] = size(R(:,:,1));
xlim([1 m])
ylim([1 n])
zlim([0 MaxPersonPerUnit])
hold off
if MyPause == 1
    pause()
end
function [MyMin] = PriorityUpdate()
%Calculate the priority of one person relativ of his
%position in the room (nearest to exit have the highest priority
%can be used once at the beginning and after every crowd update
%\
%%Globals
%global MyPause;                    %Activate pause in Simulation
%global scrsz;                      %Your actual Screen Size
%global PlotData;                   %Activate the plot functions [a,b]
global ActRand;                     %Activate the Random function in Permutation and
GetNExtMove
%global PFS;                        %Plot full screen
%global ActRoom;                    %Active Room Nr X
%global MyCounter;                  %Main Loop Counter
%global PECounter;                  %Counter for PE Matrix
%global PPCounter;                  %Counter for PP Matrix
%global LimitLoop;                  %Set a limit for the maximum numbers of loops
%global MaxPersonPerUnit;           %Define Maximum Person Density per Unit
%global MaxOutFlow;                 %Define how much people can go out at one time
%global DeleteExit1;                %Define the exit to be deleted(0) in room 1
%global DeleteExit2;                %Define the exit to be deleted(0) in room 2
%global MyMax;                      %Biggest value for the path search
%global N;                          %Amount of people
%global A;                          %Agent Matrix 4D
%global R;                          %Room Matrix 3D
%global P;                          %Path Matrix with integer value 3D
%global PA;                         %Path Matrix with arrows 3D
%global PE;                         %Information about Density near to the Exits
%global PP;                         %Outputflow for each exit for all loops
%global PD;                         %Outputflow for each exit
%global ThreshNear;                 %for Room 1 for 2 exit aside
%global ThreshDens;                 %Threshold for Density at exit cones
%global ExitConeR;                   %Length of the exit corners
%global AgWin;                      %Velocity distribution
%global T;                          %Sorted Persons depending on time
%global TT;                         %For each time numbers of people
%global MyNorm;                     %Norming factor for Velocity Distribution
%global NrExits;                    %Number of Exits
%global NumVelo;                    %Number of Velocities
%global Sigma;                      %Derivation factor for Velocity Distribution
%\%
%Update priority
i = MyMax;
while(i >= 0)
    if find(P(:,:,end)==i)
        [Fx,Fy] = find(P(:,:,end)==i);
        [Fx,Fy] = Permutation(Fx,Fy);       %mixing the orders
        l = length(Fx);
        for h=1:MaxPersonPerUnit
            for k=1:l
                if A(Fx(k),Fy(k),1,h)==1
                    A(Fx(k),Fy(k),2,h) = P(Fx(k),Fy(k),A(Fx(k),Fy(k),5,h));
                end
            end
        end
    end
end
function [R, P] = room(csvfile)
    % [R] = room(csvfile) reads in a .csv file and generates a corresponding
    % Matlab room Matrix R and exit path values in P.
    %
    % example call: [R, P] = room(’Documents/room_abstract/Room-Room.csv’);
    if nargin == 0
        R = csvread(’Documents/room_abstract/Room-Room.csv’); % updated!
    else
        R = csvread(csvfile);
    end
    R = R(2:end-1,2:end-1); % cut out the borders (coordinates from numbers.app
    for better drawing)
    P = PourPathValues(R);
    % TODO: replace inf by maxval?
end

function [MyUp, PathSearching, gg] = SearchMe(x, y, Exit, temp, gg, MyUp, PathSearching, i)
    %Search the next best position
    %
    %Globals
    global MyPause; %Activate pause in Simulation
    global scrsz; %Your actual Screen Size
    global PlotData; %Activate the plot functions [a,b]
    global ActRand; %Activate the Random function in Permutation and
    GetNextMove
    global PFS; %Plot full screen
    global ActRoom; %Active Room Nr X

    global MyCounter; %Main Loop Counter
    global PECounter; %Counter for PE Matrix
    global PPCounter; %Counter for PP Matrix
    global LimitLoop; %Set a limit for the maximum numbers of loops

    global MaxPersonPerUnit; %Define Maximum Person Density per Unit
    global MaxOutFlow; %Define how much people can go out at one time
    global DeleteExit1; %Define the exit to be deleted(0) in room 1
    global DeleteExit2; %Define the exit to be deleted(0) in room 2
    global MyMax; %Biggest value for the path search
    global N; %Amount of people
    global A; %Agent Matrix 4D
    global R; %Room Matrix 3D
    global P; %Path Matrix with integer value 3D
    global PA; %Path Matrix with arrows 3D
    global PE; %Information about Density near to the Exits
    global PP; %Outputflow for each exit for all loops
    global PD; %Outputflow for each exit
switch i
  case 1
    k=1;
    if(R(x+k,y,1)==-2)
      while(R(x+k,y,1)==-2)
        k=k+1;
      end
    end
    for gg=1:MaxPersonPerUnit
      if (P(x+k,y,Exit)>temp) && (A(x+k,y,1,gg)==0)
        MyUp = [k,0];
        PathSearching = 0;
        break;
      end
    end
  case 2
    k=1;
    if(R(x,y+k,1)==-2)
      while(R(x,y+k,1)==-2)
        k=k+1;
      end
    end
    for gg=1:MaxPersonPerUnit
      if (P(x,y+k,Exit)>temp) && (A(x,y+k,1,gg)==0)
        MyUp = [0,k];
        PathSearching = 0;
        break;
      end
    end
  case 3
    k=1;
    if(R(x-k,y,1)==-2)
      while(R(x-k,y,1)==-2)
        k=k+1;
      end
    end
    for gg=1:MaxPersonPerUnit
      if (P(x-k,y,Exit)>temp) && (A(x-k,y,1,gg)==0)
        MyUp = [-k,0];
        PathSearching = 0;
        break;
      end
    end
  case 4
    k=1;
    if(R(x,y-k,1)==-2)
      while(R(x,y-k,1)==-2)
        k=k+1;
      end
    end
for gg=1:MaxPersonPerUnit
    if (P(x,y-k,Exit) > temp) && (A(x,y-k,1,gg)==0)
        MyUp = [0,-k];
        PathSearching = 0;
        break;
    end
end

case 5
k=1;
if(R(x+k,y+k,1)==-2)
    while(R(x+k,y+k,1)==-2)
        k=k+1;
    end
end
for gg=1:MaxPersonPerUnit
    if (P(x+k,y+k,Exit) > temp) && (A(x+k,y+k,1,gg)==0)
        MyUp = [k,k];
        PathSearching = 0;
        break;
    end
end

end

case 6
k=1;
if(R(x-k,y+k,1)==-2)
    while(R(x-k,y+k,1)==-2)
        k=k+1;
    end
end
for gg=1:MaxPersonPerUnit
    if (P(x-k,y+k,Exit) > temp) && (A(x-k,y+k,1,gg)==0)
        MyUp = [-k,k];
        PathSearching = 0;
        break;
    end
end

end

case 7
k=1;
if(R(x-k,y-k,1)==-2)
    while(R(x-k,y-k,1)==-2)
        k=k+1;
    end
end
for gg=1:MaxPersonPerUnit
    if (P(x-k,y-k,Exit) > temp) && (A(x-k,y-k,1,gg)==0)
        MyUp = [-k,-k];
        PathSearching = 0;
        break;
    end
end

end

case 8
k=1;
if(R(x+k,y-k,1)==-2)
    while(R(x+k,y-k,1)==-2)
        k=k+1;
    end
end
for gg=1:MaxPersonPerUnit
    if (P(x+k,y-k,Exit) > temp) && (A(x+k,y-k,1,gg)==0)
        MyUp = [k,-k];
        PathSearching = 0;
        break;
    end
end
function ShortestPathLink()
%link the shortest path information (Value and Layer
%Coordinate in the last 2 layers of P

globals
 MyPause; %Activate pause in Simulation
 scrsz; %Your actual Screen Size
 PlotData; %Activate the plot functions [a,b]
 ActRand; %Activate the Random function in Permutation and
 GetNExtMove
 PFS; %Plot full screen
 ActRoom; %Active Room Nr X
 MyCounter; %Main Loop Counter
 PECounter; %Counter for PE Matrix
 PPCounter; %Counter for PP Matrix
 LimitLoop; %Set a limit for the maximum numbers of loops

 MaxPersonPerUnit; %Define Maximum Person Density per Unit
 MaxOutFlow; %Define how much people can go out at one time
 DeleteExit1; %Define the exit to be deleted(0) in room 1
 DeleteExit2; %Define the exit to be deleted(0) in room 2
 MyMax; %Biggest value for the path search
 N; %Amount of people
 A; %Agent Matrix 4D
 R; %Room Matrix 3D
 P; %Path Matrix with integer value 3D
 PA; %Path Matrix with arrows 3D
 PE; %Information about Density near to the Exits
 PP; %Outputflow for each exit for all loops
 PD; %Outputflow for each exit
 ThresholdR; %for Room 1 for 2 exit aside
 ThresholdD; %Threshold for Density at exit cones
 ExitConeR; %Length of the exit corners
 AgWin; %Velocity distribution
 T; %Sorted Persons depending on time
 TT; %For each time numbers of people
 MyNorm; %Norming factor for Velocity Distribution
 NrExits; %Number of Exits
 NumVelo; %Number of Velocities
 Sigma; %Derivation factor for Velocity Distribution

%Code
P(:,end) = 0.*P(:,end);
P(:,end+1) = P(:,end);
[m,n] = size(P(:,1:end));
for j=1:m
  for k=1:n
    [Val,Cor] = max(P(j,k,1:end-2));
    P(j,k,end-1) = Cor;
    P(j,k,end) = Val;
  end
end
function TimeAnlysis()
%Analise the time needed to go out
%%%
%Globals
global MyPause;             %Activate pause in Simulation
global scrsz;               %Your actual Screen Size
global PlotData;            %Activate the plot functions [a,b]
GetNExtMove
global PFS;                 %Plot full screen
global ActRoom;             %Active Room Nr X

global MyCounter;           %Main Loop Counter
global PECounter;           %Counter for PE Matrix
global PPCounter;           %Counter for PP Matrix
global LimitLoop;           %Set a limit for the maximum numbers of loops

global MaxPersonPerUnit;    %Define Maximum Person Density per Unit
global MaxOutFlow;          %Define how much people can go out at one time
global DeleteExit1;         %Define the exit to be deleted(0) in room 1
global DeleteExit2;         %Define the exit to be deleted(0) in room 2

global MyMax;               %Biggest value for the path search

global N;                   %Amount of people

global A;                   %Agent Matrix 4D

global R;                   %Room Matrix 3D


global P;                   %Path Matrix with integer value 3D

global PA;                  %Path Matrix with arrows 3D

global PE;                  %Information about Density near to the Exits

global PP;                  %Outputflow for each exit for all loops

global PD;                  %Outputflow for each exit


global ThreshNear;          %for Room 1 for 2 exit aside

global ThreshDens;          %Threshold for Density at exit cones

global ExitConeR;           %Length of the exit corners

global AgWin;               %Velocity distribution

global T;                   %Sorted Persons depending on time

global TT;                  %For each time numbers of people

global MyNorm;              %Norming factor for Velocity Distribution

global NrExits;             %Number of Exits

global NumVelo;             %Number of Velocities

global Sigma;               %Derivation factor for Velocity Distribution

%%%
%Code
[m,n] = size(R(:,:,4));
c = 1;
T = [];
for i=1:m
    for j=1:n
        if (R(i,j,4) ~= 0)
            T(cc) = R(i,j,4);
            cc = cc+1;
        end
    end
end
T = sort(T)./MyNorm;         %in sec
figure()
set(gca,'FontSize',16)
if PFS
    scrsz = get(0,'Screensize')+[0 0 0 100];
    set(gcf,'Position',scrsz)
end
hold on
plot(T,'x')
title('Time Analysis')
if PFS
    set(gcf,'Position',scrsz)
end
x = linspace(1,length(T),length(T));
t = 1:20:length(T);
plot(t,[pchip(x,T,t)],'r')
legend('Person','pchip of Person',4)
title('Number of steps needed to get out')
xlabel('Person Nr.')
ylabel('Number of Steps')
Temp = T.*MyNorm;
TT = zeros(max(Temp),2);
while max(Temp) ~= 0
    TT(max(Temp)) = numel(Temp(Temp == max(Temp)));
    Temp(Temp == max(Temp))=0;
end
figure()
if PFS
    set(gcf,'Position',scrsz)
end
set(gca,'FontSize',16)
tt = linspace(1,length(TT),length(TT));
bar(tt./MyNorm,TT)
title('Number of steps needed to get out')
xlabel('Number of Steps')
ylabel('Number of Persons')
end

%%%
MATLAB FS11 – Research Plan
(remove text between brackets)

• First, remove also the brackets and not only the text between the brackets 😊
• It’s a rather sketchy research plan, but clear in the aims. Please improve the General Introduction section.
• It’s important to compare different rooms scenario to gain insight of what are the parameters that matters
• Be more specific in what you want to explore (room squared meters/number of people, particular displacement of desks, evacuation time...)
• As an extension to the basic case, are you going to consider heterogeneity in agents/evacuation routine? (e.g. some agents may be instructed/trained to deal with emergencies, others may be just be panicked ...)
• Will you able to access real evacuation data? Real Evacuation Procedures?
• Research method: cellular automata
• Format the citation correctly
**Group participants names:** (Biner Dario, Brun Noé)

**General Introduction**

Our goal is to set up a simulation model of a lecture room through which we can find out which seats have the higher survival chance in case of an emergency. Crucial parameters such as the amount and the location of the exits, the amount of people, the size of the room and the distribution of the seats will be taken into account. It is not our aim to improve the room whatsoever. We rather try to find out how the actual lecture room is conceived regarding the emergency situation. The results should be verified with real life data if available, or at least be compared with the current worst case scenario of the security department. If the time and complexity allow we’ll gladly extend the model so that additional parameters can be taken into account such as differently behaving agents, statistical behaviour or parametrized rooms etc.

**Fundamental Questions**

How much time do we need to get out from the lecture room depending on our initial seat? What seats turn out to be better? What are the consequences if one exit is blocked?

Depending on :
1 the amount of exits
2 the amount of people
3 the initial crowd distribution

**Expected Results**

We hope that in the worst case everyone survives (if we set that after x minutes everyone left in the room dies). The results may be that the way out from seats in the middle of the room prevent people to get out fast enough.

**References**

Analytical Approach to Continuous and Intermittent Bottleneck Flows;
Dirk Helbing and Anders Johansson

**Research Methods**
- Cellular automata
- Matrix representation of the room, matrix calculation to represent the crowd movement
- Real modeling sticking close to reality (amount of seats, geometry, exits, etc)
- Information from the security department, if available