Modeling the 2010 Love Parade Evacuation

C. Fougner and J. Guthrie

19. Dezember 2010

Table Of Contents

1 Introduction

On the 24th of July the Love Parade 2010 took place in Duisberg, Germany. Through a combination of poor planning, poor crowd control and errors in communication a total of 21 people were killed and approximately 500 injured. We have based our research project on creating a model which can simulate the events of the Love Parade in order to try to recreate and represent the events of the Love Parade 2010.

We wanted to dimension our model in such a way as to allow for a realistic scaling of the physical dimensions of the location of the incident, so that we could run alternate simulations in order to examine which conditions would have been necessary to have prevented injury and death.

Throughout the event and in the aftermath there were a lot of misquoted figures relating to the incident. Fortunately, we have some clear information regarding the events, allowing us to select how to make an accurate model.

Contradictory to the quoted 1.4 million people at the event, police estimates stated that there must have been 200 000 to 300 000 attendees. The festival grounds were capable of holding approximately 250 000 attendees. The single main entry and exit point into the festival grounds was the tunnel in which the documented injuries took place. It was the poor management of people going into and out of this tunnel which resulted in the injuries which occurred.

We had assumed that the tragedy of the Love Parade 2010 had occurred after the event had ended, that some sort of panic had broken out as the attendees were streaming for the exits, caused by some sort of bottleneck in the exits.

From documentation of the event we discovered that the events leading up to the panic and following tragedy were actually quite complex and resulted as a combination of many different factors. In the end, panic in the tunnel causing extreme waves of pressure buildup was what caused injury to the attendees.

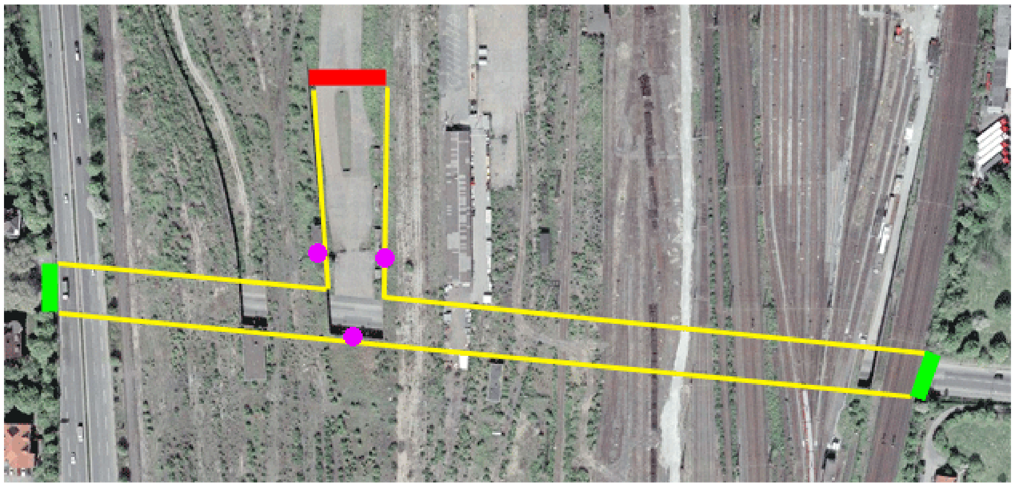
The most surprising fact is that the tragedy of the event could have somehow been avoided, because at the time that the panic in the tunnel broke out, the festival grounds were only 75% full.

Part of the problem in the management of the event was that there was miscommunication between the organisers and the police, who were helping keep the event under control. A large crowd of people had gathered leading up to the top of the ramp which fed into the festival grounds. The crowd was unable to move forward due to either gates at the top of the ramp leading into the festival area being closed, or due to the fact that people who were standing at the top of the ramp were not moving forwards because they were distracted by observing Lovemobiles and the festival. The reports are not quite clear as to exactly which of the two took place when, but it's safe so say that a combination of these two caused the large volume of people to be present. Marshalls who were instructed to close gates leading into the tunnel, which would reduce the flow of people into the tunnel, for some reason did not close these gates. Marshalls standing at the top of the ramp whose job it was to push people forward and away from the ramp area were also unable or very inefficient. The combination marshalls failures would have been enough to create a volatile situation without any additional panic-causing effects.

The final point which resulted in panic breaking out amongst the crowd was that there were three ”exits” through which people could exit the tunnel/ramp area and have the possibility of getting to the festival grounds. The first was a small stairwell, the second a lighting mast and the third a container next to a wall. People who were stuck in the crowd were under the impression that their only way out of the situation was through one of these three exits. The capacity of these exits was very low, allowing only a few people through at a time. The uncomfortable atmosphere of the crowd, combined with the existence of the ?exits? was what started waves of pressure to flow through the crowd, with people pushing and shoving to get to the exits. Eyewitness reports mention being entirely unable to control their movements as they were squashed and pushed around through the huge mass.

2 Grounds

The area which we have chosen to focus on is portrayed in the image below:

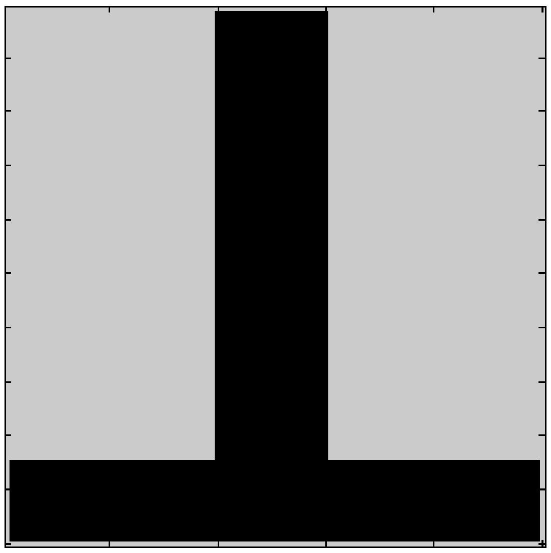


Picture of the grounds where the actual love parade took place

A few guidelines have been included to help point out the areas of interest in the image:

1. The yellow lines are the boundaries of the tunnel, and ramp area.
2. The green lines indicate the two entrances into the tunnel.
3. The red line indicates the transition between the top of the ramp and the entrance of the festival grounds.
4. The three purple points are where the stairwell, lighting mast and container were located.

We approximated this environment with an upside-down ”T” shape, as can be seen in the image below.


Our model of the grounds

The black area represents ”empty” space, the grey area represents ”walls”. People in the model are free to move around in the empty space, but are unable to move into walls.

We found it both necessary, and appropriate to approximate the model with these dimensions and size. The time required for simulations in the case of much larger matrices would become prohibitive to making any significant progress. We identified that the most important part of the scenario of the Love Parade was this T-junction where the three ”exits” were used to escape from the scene, any extra length of tunnel leading up the the ramp is relatively irrelevant to the calculations.

3 Goals

After having researched the exact events of the Love Parade, we realized that our original idea of the bottleneck being whilst people were leaving the event was incorrect, which resulted in us having to think hard about what it was we were trying to achieve with our model.

We wanted to start by creating a model which behaved in a ”predictable” manner, exhibiting what we consider to be normal human behavior. Once we had the basic model, we would start to alter the conditions of the simulations and compare the results against other simulations, as well as the events of the day of the Love Parade.

To perfect this model, we decided that we would need to make sure that two simulations worked in a predictable manner. These were:

1. A simple model in which people enter from both sides of the tunnel and proceed up onto the ramp and continue onto the festival arena
2. A model in which people enter from both sides of the tunnel and proceed up the ramp. At the top of the ramp was a wall, blocking off any exit from the tunnel and ramp area. This would allow us to observe how our model modeled the pressure buildup in the tunnel and ramp area.

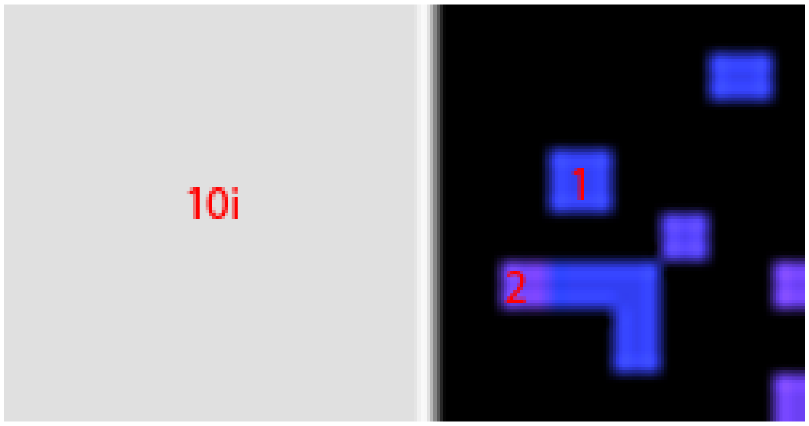
Once we had the model working with those two simulations, we wanted to do two more simulations which were more indicative of the conditions of the day, without making any changes to our established model. These simulations were as follows:

1. A model in which people enter from both sides of the tunnel and proceed to the top of the ramp. Once reaching the top of the ramp, they stop moving because they have ”arrived” at their goal. Once in this state, people only move forward up the ramp by being pushed by those coming from behind. The goal of this is to try to simulate an observed effect of people slowing down once they had reached the festival grounds, as they were watching the parade.
2. A more advanced model, which takes place in two stages:
    (a) People enter from both sides of the tunnel and proceed up the ramp. At the top of the ramp a wall blocks off any exit from the tunnel and ramp area.
    (b) Once the tunnel and ramp area is filled, the three ”exits” which were observed at the Love Parade are ”opened”. This is to simulate the desperate people leaving the Love Parade through any exit possible.

4 Description of Model

4.1 The Grounds

The grounds upon which the Love Parade took place was modeled by large matrix. A matrix cell is equivalent to one square meter. To define where people are physically allowed to move and where the walls are located, real and imaginary numbers were used. An imaginary number signifies a wall, a zero means there are no people on the square and a real positive number represents the number of people on a square. From reports it was estimated that at locations the person density was as high as 8 people per square meter.



Picture showing a wall (imaginary number, 10i), cells with 1 person and cells with 2 people

4.2 Goal, Exits and End Zones

In addition to walls there are three other ”special” areas, these are goals, exits and end zones:

Goals

A ”goal’ is the location which all people are reach and is represented by a single cell in our matrix. There can be multiple ”goals” and a person will always choose to move to the goal closest to him/her. In our scenario a goal is either the end of the ramp or one of the three ”exits”.

Exits

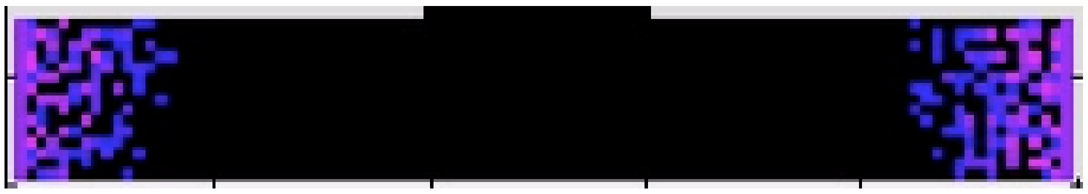
An ”exit” is an area from which a set of n people can be removed every round. In other words if a person moves to an ”exit” cell, and given that there are no more than $n-1$ people already in that cell, he will be removed from grounds in the next iteration,

End Zones

End Zones are areas where once entered, people no longer feel the need to move to anywhere else and only advance to a new cell if forced to do so by the interaction with others.

4.3 People

In the actual Love Parade, people entered from the two tunnel sides. Alas the grounds are initialized with no people and instead a constant number of people are added every round to the cells representing the tunnel exits. To account for fluctuating currents of people and to speed up the simulation, we only added more people every second iteration.



Picture showing people stream into the grounds from the tunnels

5 The Implementation

5.1 Time discretization

One of the major issues in computer simulation is that a computer can only execute one command at any given point in time, whilst real life dictates that actions can take place simultaneously. In our case we would like people to all move at the same time. To discretize time, a simple solution would be to iterate over the grounds from top left to bottom right, in other words moving the people closest to the top left hand corner first and the individuals in the bottom right last. This unfortunately creates a bottom-right bias (there are more people to the bottom and right than to the left), as the early movers get precedence and fill up the less densely populated cells first. To fix this issue we instead randomly choose the cells. To make sure all people get an equal opportunity to move a list is maintained with all the cells that have not yet been altered in the current iteration run.

The above is accomplished by the following code:

```
    [~, idxSortedList] = sort(rand(1,arenaSizex*arenaSizey));
    squaresList = [mod((idxSortedList-1),arenaSizex)+1; ...
        floor((idxSortedList-1)/arenaSizey)+1];
    %start moving people
    for randomSquare = squaresList   %iterate over all coordinates
        x = randomSquare(1);    %extract the current x coordinate
        y = randomSquare(2);    %... y coordinate
```

5.2 Movement

In implementing the movement of a person we used the concept of cellular automata wherein we were forced to made two simplifications/assumptions:

1. The mobility of an individual is only influenced by the people in one Moore neighborhood. This is justified, because one is only in direct contact with those within one Moore neighborhood.

2. People cannot move more than one cell every iteration. This is reasonable, as we are looking at such high densities of people that movement is very limited anyhow.

To model the behavior of single person we used a simple pressure model, which can be broken down as follows:

- Others will push you in the opposite direction to where they are standing, ie. people in front of you will push you backwards. Additionally, the more people, the more force they will exert on you.

- Assuming you have a ”goal”, you exert a force in that direction.

- If you are next to a wall and someone pushes you against the wall, the wall exerts an equal and opposite force on you (Newton's Third Law). The net effect being that you cannot be pushed through the wall.
- There is a maximum number of people that can occupy a cell. This is a physical constraint; it is not possible to fit more than seven to eight people in one square meter (source: reports from the Love Parade).

The function that determines how a person moves is called the `movementFactor`:

```matlab
function [pressure]=movementFactor(mobilitySquare, desiredDirection,meanWillFactor) %calculate a persons ability to move in a given direction
%variables
wallPressure =2; %how much additional force do walls apply, resp. people are unlikely to stay closer to walls
wallVariation = 0.5; %vary the amount of "pressure" that walls applay
cornerFactor = 0.707;  %1/sqrt(2), People at a diagonal to you exert a smaller force
standardDevWillFactor = 1; %how much does a persons will factor vary
claustrophobiaFactor = 1/5;    %how much is your movement inhibited by the presense of others

%Newtons third law
for j = 1:3
    for k = 1:3
        if imag(mobilitySquare(j,k))
            mobilitySquare(j,k) = mobilitySquare(4-j,4-k)+wallPressure+wallVariation*abs(randn(1));
        end
    end
end


peopleYplus = sum(sum(real(mobilitySquare).*[cornerFactor 1 cornerFactor;0 0 0;0 0 0])); %number of people in positive y  direction
peopleYminus = sum(sum(real(mobilitySquare).*[0 0 0;0 0 0;cornerFactor 1 cornerFactor])); %... negative y ..
peopleXplus = sum(sum(real(mobilitySquare).*[cornerFactor 1 cornerFactor;0 0 0;0 0 0]'));  %... positive x ..
peopleXminus = sum(sum(real(mobilitySquare).*[0 0 0;0 0 0;cornerFactor 1 cornerFactor]')); %... negative x ..
willFactor = standardDevWillFactor*(meanWillFactor+randn(1))*[1/(1+claustrophobiaFactor*min([peopleXplus,peopleXminus])); ...
    1/(1+claustrophobiaFactor*min([peopleYplus,peopleYminus]))]; %calculate a persons will and ability to move to a given cell
pressure =  (diag([cornerFactor 1 cornerFactor;-1 0 1]*(real(mobilitySquare).^2)*[1 cornerFactor;0 1;-1 cornerFactor]) + willFactor.*desiredDirection);   %vector sum of pressure plus personal will
end
```

The Arguments

- `mobilitySquare`: The first Moore neighborhood
- `desiredDirection`: A vector containing the normalized cartesian direction in which you need to move to reach your goal.

The Return Argument

`pressure` is a vector sum of the force you exert in the direction of your goal and that exerted by people around you. The magnitude of the vector corresponds to the magnitude of the force.

The Variables

To account for the laws of physics and fickle human psychology, a set of governing variables were introduced. These variables are used to calculate the `pressure`. They are:

- `wallPressure`: When trying to get somewhere, it is unwise to run directly next to a wall as you will be constantly bumping into the wall, instead you are more likely to stay a meter or so away. This can also be imagined when walking through a corridor, you tend to stay in the middle of the corridor and not up against the sides. This parameter is also of relevance when accounting for the fact, that if pushed up against a wall one is likely to push back in the opposite direction. To model this effect we simply said a wall is akin to a set number of people who cannot move, but exert a force on you. A `wallPressure` of 1 is therefore in practice the same as one person standing there.
- `wallVariation`: To account for the nonconformity of human behavior in that some people, when shoved against a wall, will react more aggressively than others, a random parameter was added. We decided this random parameter would most likely be normally distributed and is generated with Matlab's `randn` function. The `wallVariation` variable is just the standard deviation of this random parameter.
- `cornerFactor`: This variable factors in that people at a diagonal to you are further away and will therefore exert a smaller force than those directly in front of you. This comes from the conversion of polar to cartesian coordinates and for trigonometric reasons was set to $1/\sqrt{2}$
- `meanWillFactor`: This factor represents the average amount of force you exert on the people in front of you in the direction you are going. It is measured relative to pressure felt solely by the presence of other people around you.
- `standardDevWillFactor`: Nature dictates that some people are stronger and/or more aggressive than others, they will therefore push harder in the direction of their goal. We assumed this variation could be modeled by a normal distribution with a standard deviation of `standardDevWillFactor`.
- `claustrophobiaFactor`: The more people in front of you, the more your movement in that direction is inhibited and the less relevant it becomes if you are big, strong and aggressive. This factor describes how freely you can move when there are many people around you.

5.3 Functions

The program contains the following functions:

- `[avgPplPerSq] = testModel(runs)`

  This is the main function that runs the whole simulation.

  Argument:

  - `runs`: How many times should the program iterate over all cells.

Return Argument:

  - `avgPplPerSq`: On average how many squares contained one person, two people, etc...

- `[pressure] = movementFactor(mobilitySquare, desiredDirection)`
  ...see previous section

- `[square] = intSquare(dir)`
  This function maps a position in the 2D cartesian space to one of nine cells in a Moore neighborhood.

  Argument:

  - `dir`: A position in 2D cartesian space

  Return Argument:

  - `square`: One of nine cells in a Moore neighborhood, format: `[1 0]`, `[-1 1]`, `[0 0]`, etc...

- `[arena,exit,endZone,goal,titleText]=makeArenaNoExit(arenaSizex,arenaSizey)`
  This function creates an arena or a special case of the grounds where there is *no exit*.

  Argument:

  - `arenaSizex, arenaSizey`: The size of the grounds in x and y direction.

  Return Argument:

  - `arena`: A matrix containing the initial state of the grounds with people, walls and empty areas.
  - `exit`: A list of the locations of all ”exits” within the grounds.
  - `endZone`: A list of the areas defined as ”end zones”.
  - `goal`: A list containing the position of all "goals".
  - `titleText`: A title for graphical output purposes.

- `[arena,Exit,endZone,goal,titleText]=makeArenaOneExit(arenaSizex,arenaSizey)`
  Same as above except the grounds now has one exit.

- `[arena,Exit,endZone,goal,titleText]=makeArenaExitDawdle(arenaSizex,arenaSizey)`
  Same as above with people dawdling.

- `[arena,Exit,endZone,goal,titleText]=makeArenaLarge(arenaSizex,arenaSizey)`
  Same as above except optimized for a larger arena.

6 Results

Our results are tricky to discuss in an article such as this as they are based on videos which we have made of the simulations, and we are unable to show the videos in the document. We have taken images of the videos at critical times to illustrate what our point is, the videos will be available as well.

It is important to take into account the amount of time which went into calculating accurate parameters for the simulations, which we cannot present as part of our results. We found that once we had made the model, we would at times see strange behavior, which we would then correct by altering the necessary parameter and re-run the simulation. In total we must have run over 100 simulations whilst perfecting the model for the base cases of the simulation.

6.1 Simulation 1

Simulation 1 is the simulation involving people entering through both sides of the tunnel and heading towards the entrance into the Festival grounds.

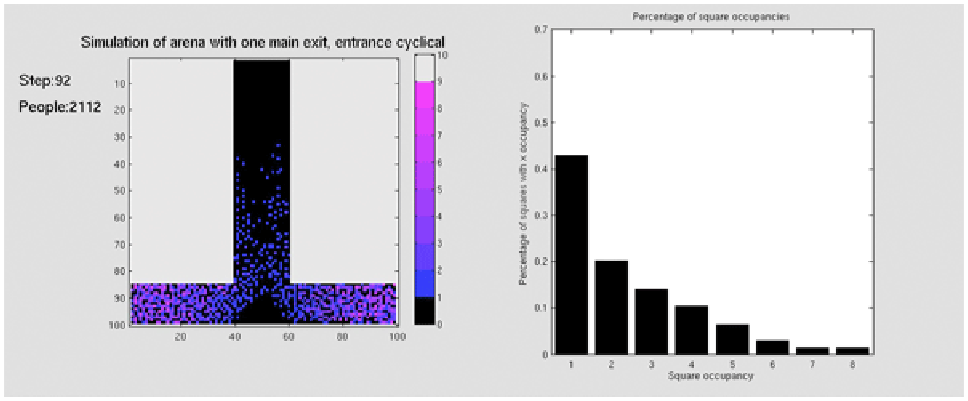


Image showing model population after 92 steps. This image gives a good impression of how people should be flowing when the density is not very high.

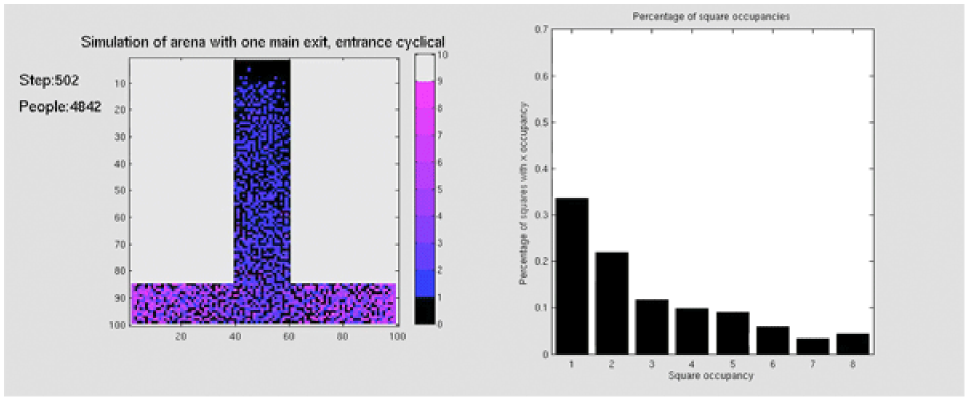


Image showing model population after approximately half of the steps have been completed.

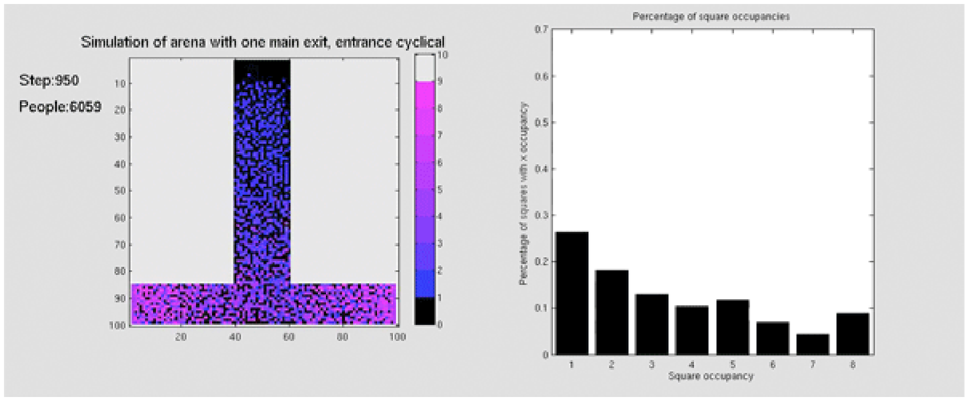


Image showing model population towards the end of the model testing.

The results of this simulation are pretty much as expected, there's a constant movement of people through the exits, after half time the model has filled up with almost 5000 people, at full time there are approximately 6000 people in the arena. This tells us that the exit has enough capacity to accommodate the flow of people through it. There are higher densities of people at the entrance to the tunnel, this density decreases as people move through the tunnel.

6.2 Simulation 2

Simulation 2 is the simulation involving people entering through both sides of the tunnel and heading towards the entrance into the festival grounds. There is a "wall" blocking off the entrance into the festival grounds.

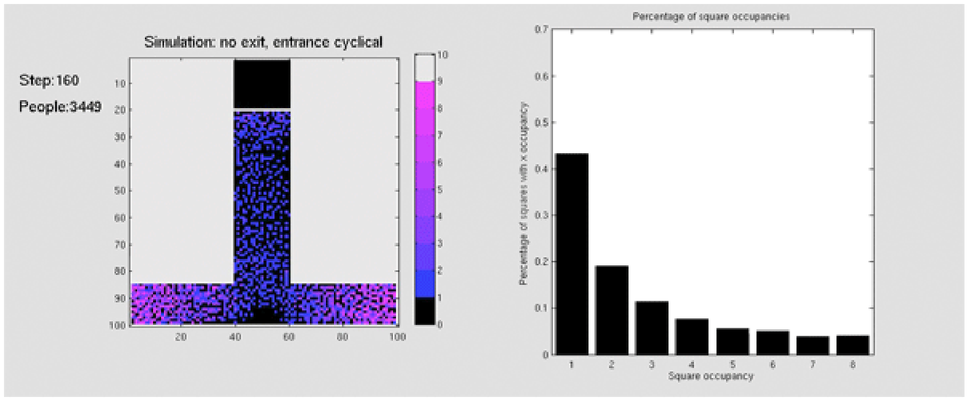


Image showing population after 160 steps

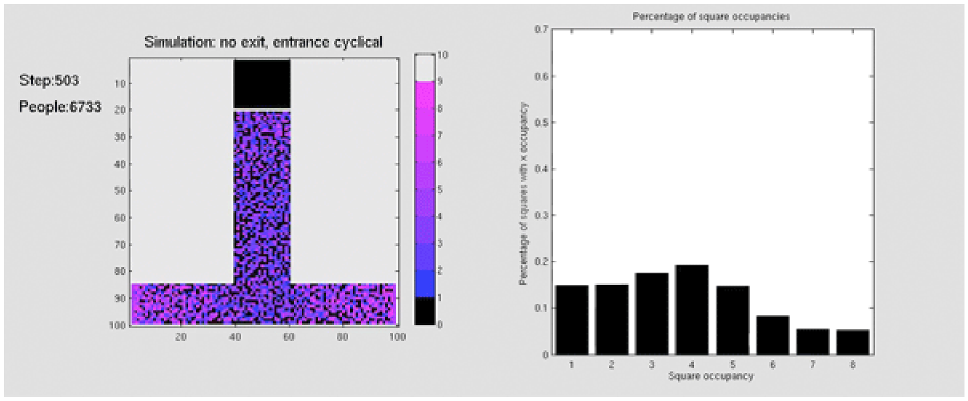


Image showing population and density distribution after 503 steps. Population approx. 6750.

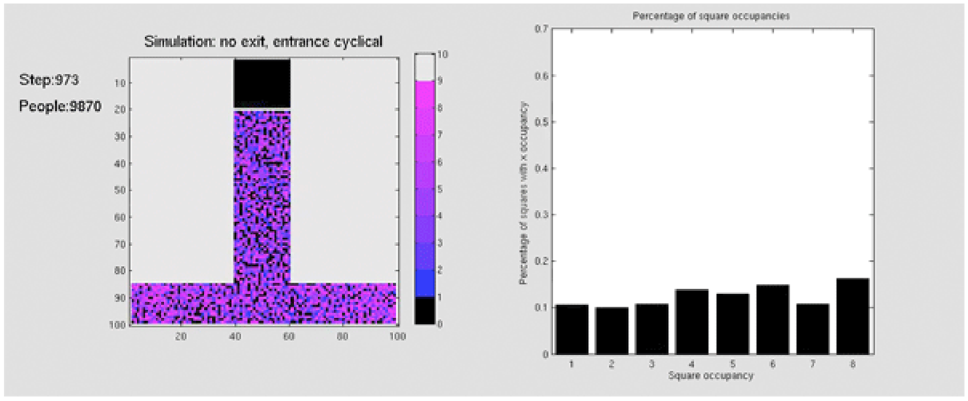


Image showing population and density distribution after 973 steps.

The results of this simulation are about what we were expecting. People entering the tunnel come up against the wall and begin to dam there as they want to continue moving towards the festival grounds. This is evidenced by the high density of people directly up against the wall. Images 2 and 3 from this simulation can be compared to images 4 and 3 of simulation 4 respectively for differences in the results.

6.3 Simulation 3

Simulation 3 is the simulation involving people entering through both sides of the tunnel and heading towards the festival grounds. Instead of entering directly into the festival grounds, the people ”dawdle” at the entrance to the festival grounds, only moving forwards when pushed by people coming behind them. The ”goal” that the people were moving towards was 10 pixels away from the exit area, which meant that a person would have had to have been pushed, or would have had to moved himself 10 pixels from his goal in order to enter the festival grounds.

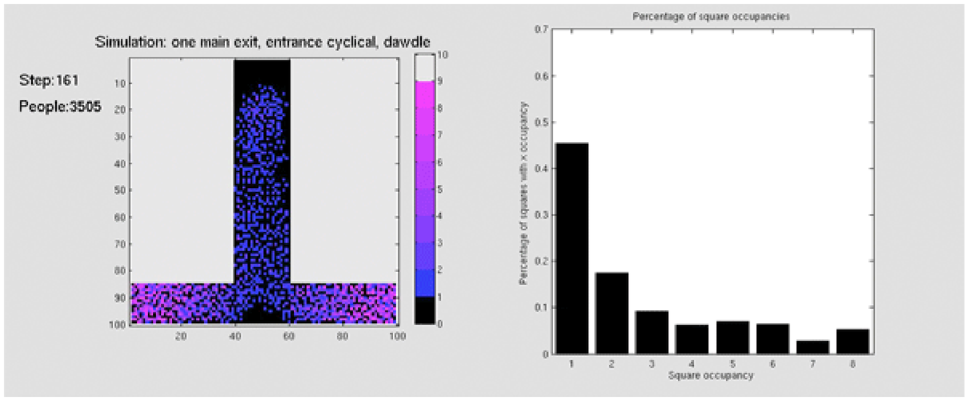


Image showing population after 160 steps

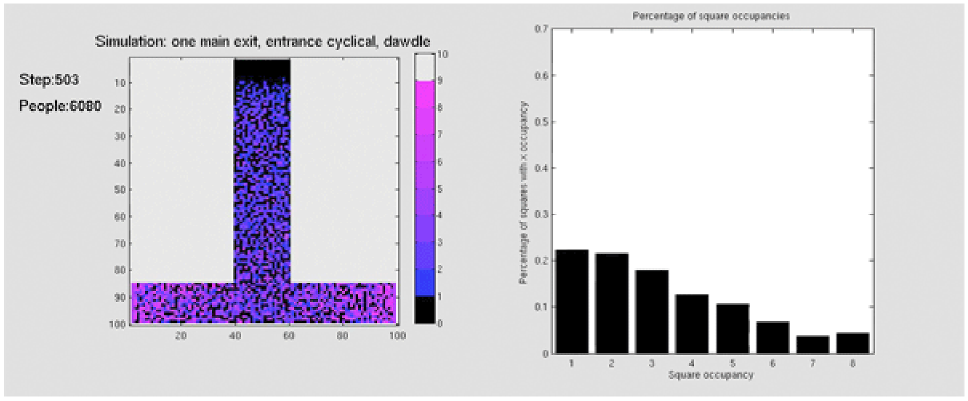


Image showing population after 503 steps

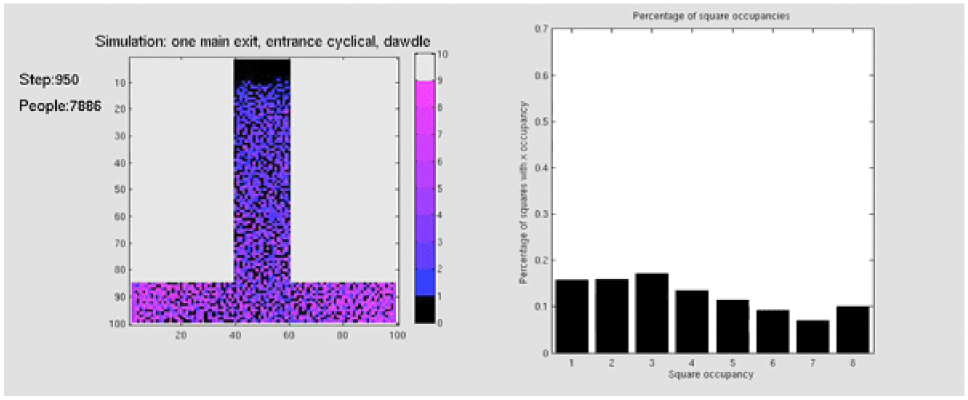


Image showing population after 950 steps

The results of this simulation are quite logical, in this case it is interesting to observe the difference between the results of this simulation and simulations 1 and 2. The amount of people in the arena in the last image is about halfway between the values at the corresponding time points in simulations 1 and 2. This shows us that people dawdling in front of the entrance to the festival grounds could have caused a similar, if not quite as strong, effect as a wall being placed there.

6.4 Simulation 4

Simulation 4 is the simulation in two parts. The first part is much the same as Simulation 2, the first 1000 steps are exactly the same as Simulation 2 which expectantly produces similar results. In the second part of Simulation 4 the flow of people into the area stops, the three exits (stairwell, lighting mask, container) are open, the "mood" of the crowd changes to be more desperate (changed the meanWillFactor from 1 to 8), we also use the line-of-sight function to determine to which exit a person should go.

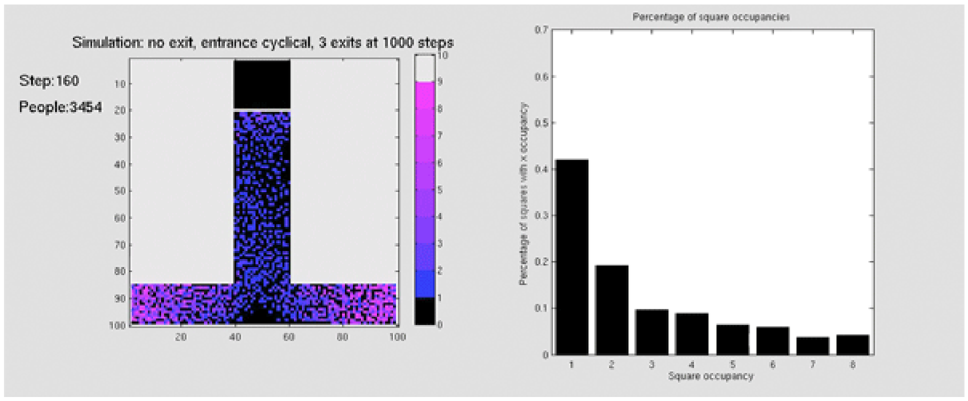


Image showing population after 160 steps

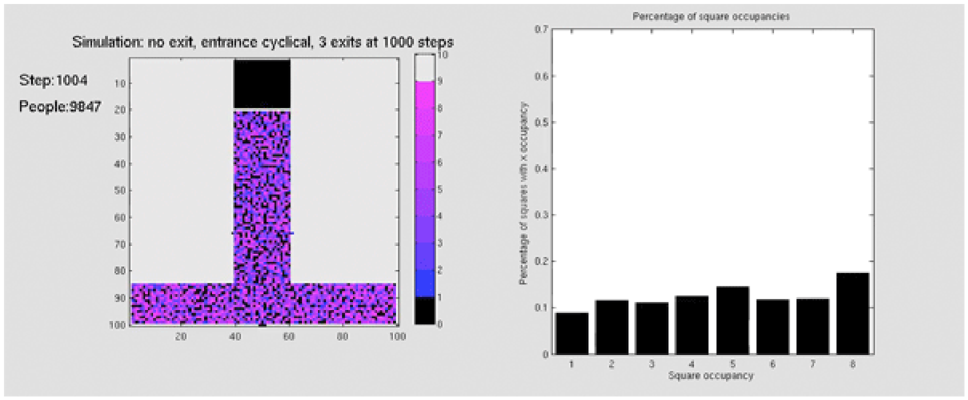


Image showing population and distribution at the time of the conditions switch

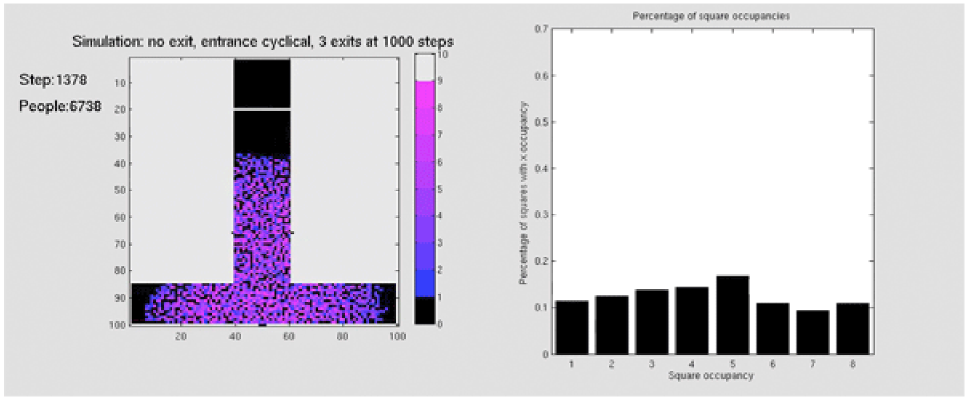


Image showing population and distribution at approximately halfway after the conditions switch

The results of this simulation show some interesting results. We expected that the density of people around the exits would be much higher after the conditions switch, we also anticipated that the density of people would overall increase quite a bit after the condition switch, mostly in part due to the change in the ”meanWillFactor” property.

We observe that at 1000 steps, the density distribution (distribution of numbers of people on each square) is approximately even (compare the to exponential distribution at 160 steps). This is to be expected as there density is generally quite high, and it is not possible for everybody to have their own square.

We observe that at 1378 steps, the density distribution is once again approximately even, but there are many empty squares. We can conclude that this is the effect of the ”meanWillFactor” property, as people are pushing to get towards the exits. The overall density at these exits is, however, not as high as we were expecting it might be. It doesn?t appear as though there is much difference in the density at the exits as compared the density at the wall in simulation 2. This is obviously a shortcoming of the model we have implemented and would be recommended to be improved before further testing is done.

7 Discussion

7.1 Limitations and Potential Improvements

The limitations to our model generally have to do with space discretization, ie. the resolution of our model was one square meter. This small resolution allowed us to create a simple pressure model, but made it difficult to observe the behavior of an individual. The pressure model abstracts groups of people into dots, this gives an averaged out overview and is thereby oblivious to the actions and locations of a single person.

Possibly the largest discrepancy between what happened in real life and our model is that one finds numerous cells occupied by no people next to ones occupied by up to 8 people. This really should not happen and it would be much more realistic to have a continuous distribution of people. This could have improved upon by analyzing the first Moore neighborhood in much more detail, but would require considerably more complex code and would result in much slower models

An additional shortcoming was that we had to artificially limit the number of people per square. The pressure buildup on one square should have limited the high number of people naturally, but it did not. This could have been resolved had we instead assigned a maximum of one person to on cell, the individual cell then in effect can be viewed as the minimum space that a person requires. This limit is much more natural. As mentioned in the results an effect that was unobservable, was the lack of a high person-density buildup around the exits. This is an outcome that, when logically considered, should have been present and is a side effect of the low resolution of the pressure model.

A topic that we struggled with for some time was how to model the psychology of an individual. We did add some random factors, but in essence people acted like robots. As an example they chose the exit closest to them even if there were fewer people around other exits. Additionally the fact that we only looked at one Moore neighborhood meant that people were indifferent to anything further away than a meter. In a real life situation people will try to optimize their route by looking into a distance to see where there are less people. An additional consideration is that if people do not have a line of sight to their goal, they should just continue along in a given direction. On our grounds we would have expected people to congeal in the center of the ”T” shape. Instead people headed straight for the goal even if they could not see it and therefore not possibly know about it. We essentially put blindfolds on peoples eyes and whispered "hotänd "coldïnto their ears, leading them to the goal. This limitation could potentially be improved upon if we looked at a larger Moore neighborhood. Again we are however looking at much longer simulation times.

7.2 Conclusion

Potentially the most frustrating aspect of our project is the fact that we have been trying to base our model on people, their behavior, and what is considered ”normal” behavior. At times it’s very difficult to imagine how people will behave and react to external stimuli until they are in the situation itself.

Watching videos of the Love Parade incident was a very eye-opening experience, as even by reading documentation of the event we could not begin to understand the conditions that the people present were under.

We feel that although our model produced some interesting and relevant results, we have after all been trying to quantify the actions and reactions of people, in situations which we ourselves can barely begin to comprehend. This in itself is a very difficult, if not almost impossible task.

This project has been an interesting learning experience, trying to understand the situation, all of the potential variables and under which conditions these variables change has taught us a lot about human behavior (or how little we know about it).

8 Appendix

8.1 Source Code

```
function [avgPplPerSq,pplPerSq]=testModel(runs)

%global variables
arenaSizex = 100;   %size of arena in x direction
arenaSizey = 100;   %size of arena in y direction
maxSize = max(arenaSizex,arenaSizey);   %maximum side length
maxPplPerSq = 8;    %maximum number of people allowed per square meter
pplPerSq = zeros(maxPplPerSq,runs); %A history of how many squares had n people
exitsPerRound = 3;  %number of people allowed to exit the grounds per iteration
meanWillFactor = 1;     %a persons average will factor
lineOfSight = 0;    %BOOL

%create figure
figure
hold off

[arena,exit,endZone,goal,titleText] = makeArenaNoExit(arenaSizex,arenaSizey);

aviobj = avifile('mymovie.avi','fps',25);
set(gcf,'position',[0 0 1200 500]);

%initialize List outside loop, this creates a slight speed improvement
initializeOutside = 0;
if initializeOutside == 1
    [~, idxSortedList] = sort(rand(1,arenaSizex*arenaSizey));   %generate a random list
from 1 to arenaSizex*arenaSizey
end

%RUN SIMULATION
for runNumber = 1:runs

    %add people to the arena every iteration
    if ((mod(runNumber,100) <= 50)&& runNumber <1000)
        for xadd = 85:99
            arena(xadd,2) = 5; %add people on the left
            arena(xadd,arenaSizex-1) = 5; %... right
        end
    end

    if (runNumber == 1000) % change arena at 1000 steps
        arena(100,49:51) = 0.001i;
        arena(66,39) = 0;
        arena(66,61) = 0;
        exit = [49 51 99 99;39 39 66 66;61 61 66 66]; %[xStart,xEnd,yStart,yEnd]
        endZone = [50 50 100 100;39 39 66 66;61 61 66 66];
        goal = [50 100;39 66;61 66];
        meanWillFactor = 8; %panic breaks out and people go crazy
        lineOfSight = 1;
    end

    %statistics
    for k = 1:maxPplPerSq
        pplPerSq(k,runNumber) = sum(sum(arena==k));
    end
    pplPerSq(maxPplPerSq,runNumber) = sum(sum(arena >= maxPplPerSq));

    %display figure
    clf     % Clear figure
    subplot(1,2,1);
    colormap([[0,linspace(32/255,221/255,8),0.85];[0,linspace(14/255,14/255,8),0.85];
[0,linspace(246/255,246/255,8),0.85]]');
    imagesc(abs(arena), [0 10])                   % Display grid
    axis image;
    colorbar
    title (titleText,'fontsize',15)
    text(-40,10,strcat('Step:',int2str(runNumber)),'fontsize',15)
    text(-40,20,strcat('People:',int2str(sum(sum(real(arena))))),'fontsize',15)
    h = subplot(1,2,2);
    hold off
    axis manual

    bar(h,pplPerSq(:,runNumber)/sum(pplPerSq(:,runNumber)));
    axis([0.5 8.5 0 0.7])
    title ('Percentage of square occupancies');
    xlabel('Square occupancy');
    ylabel('Percentage of squares with x occupancy');
    frame = getframe(gcf);
    if (mod(runNumber,2) == 0)
        aviobj = addframe(aviobj,frame);
    end

    %initialize Random Sequence
    if initializeOutside == 0
        [~, idxSortedList] = sort(rand(1,arenaSizex*arenaSizey));
    end
    %a random list of every square
    squaresList = [mod((idxSortedList-1),arenaSizex)+1; floor((idxSortedList-1)/
arenaSizey)+1]; %Generate a set of 100*100 randomly distributed coordinates

    %start moving people
    for randomSquare = squaresList   %iterate over all coordinates
        x = randomSquare(1);    %extract the current x coordinate
        y = randomSquare(2);    %... y coordinate
        if arena(y,x) <= 0
            continue    %skip cell if no one is there
        end

        %line of sight - ie. can people see the goals
        goalsInLineOfSight = zeros(0);

        if lineOfSight == 1
            for goalNumber = 1:size(goal,1);    %generate a list of what goals
individuals can see from their current location
                oneGoal = goal(goalNumber,:);
                inSight = 1;
                dir = [oneGoal(1)-x (oneGoal(2)-y)]';
                if norm(dir) == 0
                    continue
                end
                dir = dir/norm(dir);
                for k = 1:maxSize
                    if norm(oneGoal' - ([x y]'+floor([dir(1) dir(2)])'*k)) < 3
                        break
                     end
                    if ~(y+dir(2)*k < 1 || y+dir(2)*k > arenaSizey || x+dir(1)*k < 1 || x
+dir(1)*k > arenaSizex) ...
                            && imag(arena(y+floor(dir(2)*k),x+floor(dir(1)*k))) ~= 0
                        inSight = 0;
                        break
                    end
                end
                if inSight == 1 %if no wall is blocking the line of sight to a goal, then
add that goal to the list
                    goalsInLineOfSight = [goalsInLineOfSight; oneGoal];
                end
            end
        end
        if norm(goalsInLineOfSight) == 0    %if no goals can be seen, then assume a
person can see all goals
            goalsToTry = goal;
        else
            goalsToTry = goalsInLineOfSight;
        end

        bestDesiredDirection = [Inf Inf];
        for goalNumber = 1:size(goalsToTry,1)   %find the closest goal
            if (x >= endZone(goalNumber,1)) && (x <= endZone(goalNumber,2)) && (y >=
endZone(goalNumber,3)) && (y <= endZone(goalNumber,4))
                desiredDirection = [0;0];   %if we are at the end zone then we don't want
to move from our current position
                break
            end
            currentDesiredDirection = [(goalsToTry(goalNumber,1)-x) -(goalsToTry
(goalNumber,2)-y)]';
            if norm(currentDesiredDirection) < norm(bestDesiredDirection)  %is this goal
closer than the last goal
                bestDesiredDirection = currentDesiredDirection;
                desiredDirection = bestDesiredDirection/norm(bestDesiredDirection);
            end
        end

        for exitNumber = 1:size(exit,1)     %allow people to exit
             if x >= exit(exitNumber,1) && x <= exit(exitNumber,2) && y >= exit
(exitNumber,3) && y <= exit(exitNumber,4)
                 arena(y,x) = arena(y,x) - min(exitsPerRound, arena(y,x));
                 if arena(y,x) <= 0
                      continue    %skip cell if everyone has exited
                 end
             end
        end

        for person = 1:arena(y,x)   %iterate over every person in a cell
            arena(y,x) = arena(y,x)-1;  %remove person from position
            chosenSquare = intSquare(desiredDirection);

            if imag(arena(y-chosenSquare(2),x+chosenSquare(1))) ~= 0    %will we hit a
wall? In which case we need to find where go to get to our goal
                dirNintyDeg = [0 -1;1 0] * chosenSquare;    %rotate the chosen direction
90Deg, ie. parallel to wall
                for k = 1:maxSize
                    if ~(y-(chosenSquare(2)+dirNintyDeg(2)*k) < 1 || y-(chosenSquare
(2)+dirNintyDeg(2)*k) > arenaSizey ...    %make sure we are not testing outside of arena
                            || x+(chosenSquare(1)+dirNintyDeg(1)*k) < 1 || x+
(chosenSquare(1)+dirNintyDeg(1)*k) > arenaSizex) ...
                             && imag(arena(y-(chosenSquare(2)+dirNintyDeg(2)*k),x+
(chosenSquare(1)+dirNintyDeg(1)*k))) == 0     %have we found a non-wall position?
                        desiredDirection = dirNintyDeg/norm(dirNintyDeg);   %rotate
chosen direction 90Deg
                        break
                    end
                    if ~(y-(chosenSquare(2)-dirNintyDeg(2)*k) < 1 || y-(chosenSquare(2)-
dirNintyDeg(2)*k) > arenaSizey ...
                            || x+(chosenSquare(1)-dirNintyDeg(1)*k) < 1 || x+
(chosenSquare(1)-dirNintyDeg(1)*k) > arenaSizex) ...
                            && imag(arena(y-(chosenSquare(2)-dirNintyDeg(2)*k),x+
(chosenSquare(1)-dirNintyDeg(1)*k))) == 0
                        desiredDirection = -dirNintyDeg/norm(dirNintyDeg);  %rotate
chosen direction -90Deg
                        break
                    end
                    if k == maxSize
```

```
                        desiredDirection = [0;0];   %could not find a way to get to our
goal
                    end
                end
            end

            chosenDirection = movementFactor(arena(y-1:y+1,x-1:x+1),
desiredDirection,meanWillFactor);  %calculate where we want to go
            actualDir = intSquare(chosenDirection); %convert this to a Moore neighborhood

            if imag(arena(y-actualDir(2),x+actualDir(1))) == 0 && arena(y-actualDir(2),x
+actualDir(1)) < maxPplPerSq    %if our desired cell is not wall and the maxPpl has not
been reached
                arena(y-actualDir(2),x+actualDir(1)) = arena(y-actualDir(2),x+actualDir
(1)) + 1; %then place person in position
            else
                arena(y,x) = arena(y,x) + 1;   %else put person back where he came from
            end
        end
    end
end

aviobj = close(aviobj);

%average
avgPplPerSq = mean(pplPerSq,2);
end

function [pressure]=movementFactor(mobilitySquare, desiredDirection,meanWillFactor)
%calculate a persons ability to move in a given direction
%variables
wallPressure =2; %how much additional force do walls apply, resp. people are unlikely to
stay closer to walls
wallVariation = 0.5;  %vary the amount of "pressure" that walls applay
cornerFactor = 0.707;   %1/sqrt(2), People at a diagonal to you exert a smaller force
standardDevWillFactor = 1;  %how much does a persons will factor vary
claustrophobiaFactor = 1/5;     %how much is your movement inhibited by the presense of
others

%Newtons third law
for j = 1:3
    for k = 1:3
        if imag(mobilitySquare(j,k))
            mobilitySquare(j,k) = mobilitySquare(4-j,4-k)+wallPressure+wallVariation*abs
(randn(1));
        end
    end
end


peopleYplus = sum(sum(real(mobilitySquare).*[cornerFactor 1 cornerFactor;0 0 0;0 0 0]));
%number of people in positive y  direction
peopleYminus = sum(sum(real(mobilitySquare).*[0 0 0;0 0 0;cornerFactor 1 cornerFactor]));
%... negative y ..
peopleXplus = sum(sum(real(mobilitySquare).*[cornerFactor 1 cornerFactor;0 0 0;0 0 0]'));
%... positive x ..
peopleXminus = sum(sum(real(mobilitySquare).*[0 0 0;0 0 0;cornerFactor 1
cornerFactor]'));  %... negative x ..
willFactor = standardDevWillFactor*(meanWillFactor+randn(1))*[1/
(1+claustrophobiaFactor*min([peopleXplus,peopleXminus])); ...
    1/(1+claustrophobiaFactor*min([peopleYplus,peopleYminus]))]; %calculate a persons
will and ability to move to a given cell
pressure =  (diag([cornerFactor 1 cornerFactor;-1 0 1]*(real(mobilitySquare).^2)*[1
cornerFactor;0 1;-1 cornerFactor]) ...
```

```
    + willFactor.*desiredDirection);    %vector sum of pressure plus personal will
end

function [square]=intSquare(dir) %convert from a point in 2D cartesian space to a first 
Moore neighborhood
if abs(dir) < 3/2
    square = round(dir);
else
    angl = angle([1 1i]*dir)-pi/16;
    if angl < 0
         angl = angl + 2*pi;
    end
    correspondanceMatrix = [1 1;0 1;-1 1;-1 0;-1 -1;0 -1;1 -1;1 0]';
    square = correspondanceMatrix(:,floor(angl*8/(2*pi))+1);
end
end

function[arena,exit,endZone,goal,titleText]=makeArenaNoExit(arenaSizex,arenaSizey)
%Initialize Arena Borders
arena = 10i*ones(arenaSizey,arenaSizex);

%initialize Arena
arena(85:arenaSizey-1,2:arenaSizex-1) = 0;
arena(2:arenaSizey-1,40:60) = 0;
arena(20,40:60)=10i;

%small arena
exit = [45,55,2,10]; %[xStart,xEnd,yStart,yEnd]
endZone = [45 55 2 10];
goal = [50 4]; %[goalx, goaly] goal must be within endZone

titleText = 'Simulation: no exit, entrance cyclical, 3 exits at 1000 steps';
end

function[arena,exit,endZone,goal,titleText]=makeArenaOneExit(arenaSizex,arenaSizey)
%Initialize Arena Borders
arena = 10i*ones(arenaSizey,arenaSizex);

%initialize Arena
arena(85:arenaSizey-1,2:arenaSizex-1) = 0;
arena(2:arenaSizey-1,40:60) = 0;

exit = [45,55,2,10;45 55 90 93]; %[xStart,xEnd,yStart,yEnd]
endZone = [45 55 2 10];
goal = [50 4]; %[goalx, goaly] goal must be within endZone

titleText = 'Simulation of arena with one main exit, entrance cyclical';
end

function[arena,exit,endZone,goal,titleText]=makeArenaOneExitDawdle(arenaSizex,arenaSizey)
%Initialize Arena Borders
arena = 10i*ones(arenaSizey,arenaSizex);

%initialize Arena
arena(85:arenaSizey-1,2:arenaSizex-1) = 0;
arena(2:arenaSizey-1,40:60) = 0;

exit = [45,55,2,3]; %[xStart,xEnd,yStart,yEnd]
endZone = [45 55 4 10];
goal = [50 4]; %[goalx, goaly] goal must be within endZone

titleText = 'Simulation of arena with one main exit, entrance cyclical';
end
```

```
function[arena,exit,endZone,goal]=makeArenaLarge(arenaSizex,arenaSizey)
%Initialize Arena Borders
arena = 10i*ones(arenaSizey,arenaSizex);

%initialize Arena
arena(85:arenaSizey-1,2:arenaSizex-1) = 0;
arena(2:arenaSizey-1,40:60) = 0;
arena(20,40:60)=10i;

exit = [45,55,2,10]; %[xStart,xEnd,yStart,yEnd]
endZone = [45 55 2 10];
goal = [50 4]; %[goalx, goaly] goal must be within endZone
end
```