Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

Simulation of a hospital evacuation with the Social Force Model

Nemanja Andric, Gioele Balestra,
Duncan Betts and Elisa Fattorini

Zurich
May 2012

Agreement for free-download

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

Nemanja Andric

Gioele Balestra

Duncan Betts

Elisa Fattorini

Contents

1 Abstract

This report is about a simulation of a hospital evacuation based on the Social Force Model. Evacuation from buildings is an important issue for safety. In contrary to the normal evacuation scenarios, in a hospital people have different movement abilities. Different kinds of agents have to be treated differently, which makes using of the Social Force Model difficult, but more interesting. Several ratios between staff members, walking patients, and bedbounded patients are tested in order to find the best condition for evacuation. The main parameter we are interested in is the evacuation time. Futhermore, we study the influence of different locations of bedbounded patients.

2 Individual contributions

The idea of a Hospital Evacuation Simulation was a result of our different interests, mixing pedestrian dynamics with medical issues.
Our project was mainly a group work, where everyone had her/his own specialization. The main line of the project and the analysis of the result was done by all of us together.
Elisa Fattorini was involved in the hospital map creation and conversion into MATLAB data.
Nemanja Andric took care of the implementation of the Fast Marching Algorithm to create the potential field from the images.
Gioele Balestra and Nemanja Andric were mostly involved in the implementation of the Social Force Model and the main program.
Duncan Betts also worked on the main program and was in charge of all the visualizations and of the batch simulations to get the results.
Everybody was involved in trying to fix the errors that the other had in their programmes. Furthermore, more solutions were found to fix the problems, which also gave us two different ways of implementing the main program.
Our coding was a continuous evolution thanks to the help of all the members of the group.
Gioele Balestra converted the report into Latex.

3 Introduction and Motivations

3.1 General Introduction

Evacuation of buildings is an important issue for safety. Until now a lot of evacuation plans have been developed for different purposes. Here we want to study in more detail the evacuation of a hospital, which is particularly interesting and difficult because of the different movement capabilities of people within. Some studies were already done, for example Taaffe et al. [1] developed a model to understand, analyse, and improve hospital evacuation plans. They assumed three different acuity levels for the patients. First levels were the patients that were candidates to release from hospital, while third level patients were critical care patients. The second level was in between. The order of evacuation of the patients was not related to their acuity level. To evacuate a patient a nurse had to assist in transporting and remain with the patients till he reached a save zone. Based on feedback from hospitals they assumed that one nurse was required for the transport for every 5 patients of acuity level 1 and 2 and for 3 patients of acuity level 3. The time to relocate the patients was a function of the patient's acuity level. The correlation between patient counts and evacuation times was assessed. Taaffe et al. [1] did not vary the number of nurses for each type of patient and proposed this variation for future tests. By holding the patient count constant, the effect of the number of nurses assigned to each acuity level can be tested and additionally they proposed to assign different times required for the preparation of the different acuity levels. These additional tests could improve the evacuation system. In our model we assume there are some patients that are able to walk, and so they do not need any nurse for evacuation. Other patients have a high acuity level, i.e. they are bed bounded and they need a staff member who will evacuate them. We take into account the preparation time for the transport of the bed-rested patients as proposed by Taaffe et al. [1]. In addition we run the simulation for different staff-patients ratio in order to obtain the optimal staff number per bed bounded patient. In other studies, for example in the publication of C.W. Johnson, "Using Computer Simulations to Support A Risk-Based Approach For Hospital Evacuation" [2], the patients are divided in ambulant and non-ambulant patients. The number of staff and the total number of patients are then maintained constant and the evacuation time was computed.

3.2 Model

Our simulation of the hospital evacuation is based on the social force model. This kind of modeling has already been applied in several configurations, but in our project we will test it with agents that posses different characteristics and goals.

The agents types are: staff of the hospital (nurses, doctors, etc.), patients that can be either able to walk or only lying in the bed. The staff have to rescue the bedbound patients, while the walking patients can go out by themselves.

3.3 Variables of interest

The independent variables of our model are:

- number of staff: n_{staff}
- number of walking patients: $n_{patient}^{walking}$
- number of bedbound patients: $n_{patient}^{bed}$

The dependent variable which will be used to judge the evacuation conditions is:

- time needed to evacuate the hospital floor. Remark: this is not a physical time but rather the number of iterations needed for all the agents to get outside of the building.

Our goal is to find the best conditions (relation between independent variables) to evacuate the hospital in the shortest time possible. To achieve this we define the following variables that represent the simulation conditions:

- Total number of agents: $n_{agent} = n_{staff} + n_{patient}^{walking} + n_{patient}^{bed}$
- Percentage of staff from all agents: $\%_{staff/agent} = \frac{n_{staff}}{n_{agent}}$
- Percentage of patients that are bedbound from all the patients: $\%_{bed/patients} = \frac{n_{patient}^{bed}}{n_{patient}^{bed} + n_{patient}^{walking}}$

3.4 Fundamental Questions

With our project we would like to answer the following questions.

1. What is the optimal ratio between staff members and agents $\%_{staff/agent}$?
2. How the number of bedbounded patients $\%_{bed/patients}$ influence the evacuation time?
3. What is the best location for bedbounded patients at the floor?
4. How the total number of agents n_{agent} influence the evacuation time?
5. How does the social force model work for this simulation?

3.5 Expected Results

We expect to have the following answer to the above mentioned questions:

1. The number of staff needed is almost equal to the number of bedbounded patient so that there will be only one directional flow (the staff members do not have to go back into the hospital to rescue other people, no counterflow). Altough, this could maybe result in an unreasonable number of staff, which is not attainable for the hospital.

2. With an increasing number of bedbounded patients the evacuation time increases because the staff have to go back to rescue them.

3. The best location for bedbounded patients will be close to the exit so when the when the staff go back to rescue them they have to cross a shorter distance.

4. With an increasing number of agents in the hospital the interaction between the agents increases as well. This could result in pedestrian dynamic effects as bottle neck, etc. which can slow down the evacuation.

5. The social force model will have little influence when we have few agents in the building because there will be only few interactions. It will become more interesting when there are a lot of people (of different type) interacting with each other.

4 Description of the Model

As announced before the evacuation simulation is based on the Social Force Model where every type of agent have different properties and targets.

4.1 Simplifications

In order to answer our initial questions we have done the following simplifications for the simulation:

- We consider only one floor of the hospital.

- The staff of the hospital are initially in their office rooms or randomly distributed in the floor.

- The staff only help bedbound patients, walking patients have to go out by themselves. The staff and bedbound patient are considered toghether as a new agent with different properties.

- The walking patients are not allowed to rescue bedbound patients.
- At the beginning all the agents are not moving.

Despite these simplifications, this model is a good approximation because it represents a general situation in an hospital where we have patients with different mobility and staff.

4.2 Evacuation strategy

An important characteristic of our model is the fact that in contrast to other pedestrian dynamic simulation, the target of the agents (the staff) varies during time.
In order to rescue everybody the staff members follow here shortly described strategy:

1. The staff go to their closest bed.
2. When a staff reaches a bed, she/he takes the patient from the bed and goes towards the exit.
3. When they reach the exit they will leave the patient there and if there are still bedbounded patients to rescue, they will go to the one which is the most farthest away from the exit. This follows the assumption that places far away from the exit are more critical for evacuation.
4. Once that there are no more bedbound patients in the hospital floor, when the staff member reaches the exit she/he will stay outside.
5. At the same time the patients that are able to walk go towards the exits by themselves.

4.3 Social force model

We will describe now the Model which is used for the interaction between the people. The movement of all the agents is simulated by using the Social Force Model. This section is based on [3], [5], [4]. The *social force* is not a physical external force, it is rather a value which describes the *motivation to act* and make people moving in a direction with a given acceleration/deceleration. The dynamic of the pedestrian movement follows Newton's second law with those social forces acting on each person. There are different kinds of social forces.

Nomenclature

- actual position of agent i: $\vec{r}_i(t)$
- actual velocity of agent i: $\vec{v}_i(t)$
- desired velocity of agent i: $\vec{v}_i^0(t)$
- maximal velocity of agent i: $v_i^{max}(t)$

4.3.1 Destination force

Destination force can be described as a *static* force. Being independent of space and time, it will draw agents towards the designated locations. In a purely abstract form these locations can be presented by any geometrical shape. In our simulation the designated location is presented with the exit hospital doors. The exits can be placed anywhere, which is a matter of space allocation, and the desired outcome will be the same. For successful implementation of destination forces we need to know the position vector of the exit, $\vec{r}_\alpha^k$ and the position vector of an agent, $\vec{r}_\alpha^k(t)$. Since exit will have an orthogonal shape, its position vector will be presented by the position vector of the nearest point with respect to agent. Subtraction of the position vectors of agent and exit and corresponding normalization will give us a desired direction,

$$\vec{e}_\alpha(t) := \frac{\vec{r}_\alpha^k - \vec{r}_\alpha(t)}{||\vec{r}_\alpha^k - \vec{r}_\alpha(t)||} \tag{1}$$

It is important mentioning that this tedious procedure of vector subtraction for determination of the desired direction hasn't been implemented it its explicit form. Instead, an appropriate Fast Marching Algorithm was used for such a purpose (see section 5.3.1). If there are no disturbances the agents will move in the desired direction, $\vec{e}_\alpha(t)$ with desired velocity, $\vec{v}_\alpha^0(t) := v_\alpha(t)^0\vec{e}_\alpha(t)$. However, due to deceleration and avoidance processes, the agents actual velocity, $\vec{v}_\alpha(t)$ will be different from their desired velocity, $\vec{v}_\alpha^0(t)$. This deviation of the actual velocity will lead to a tendency to approach desired velocity within a certain relaxation time τ_α. This process can be described by the following acceleration term which correspond to the destination force,

$$\vec{F}_\alpha^0(\vec{v}_\alpha^0, v_\alpha^0\vec{e}_\alpha^0) := \frac{1}{\tau_\alpha}\left(v_\alpha^0\vec{e}_\alpha^0 - \vec{v}_\alpha^0\right) \tag{2}$$

4.3.2 Boundary force

Boundary forces can be classified as a second group of *static* forces. Unlike the destination forces, which are attractive, boundary forces are repulsive in their nature. In real life we cannot go through walls and of course we will try avoiding hitting them most of the time. In order to implement this in the simulation, we can set up boundary force field which will repel agents from the walls and other materialistic obstacles. The boundary force will be the highest very close to the wall and it will exponentially decrease as we go further away from it, i.e. boundary forces are exponential in their nature. If we label boundary as B, the repulsive force can modeled as

$$\vec{F}_{\alpha B}(\vec{r}_{\alpha B}) := -\nabla_{\vec{r}_{\alpha B}} U_{\alpha B}(||\vec{r}_{\alpha B}||) \tag{3}$$

with a boundary potential $U_{\alpha B}(||\vec{r}_{\alpha B}||)$. Here a new vector has been introduced, $\vec{r}_{\alpha B} := \vec{r}_{\alpha} - \vec{r}_{B}^{\alpha}$, where $\vec{r}_{B}^{\alpha}$ presents the location of the boundary which is closest to the agent α. As already mentioned, the repulsive potential has been modeled as an exponential function,

$$U_{\alpha B}(||\vec{r}_{\alpha B}||) = U_{\alpha B}^{0} \exp\left(-||\vec{r}_{\alpha B}||/R\right) \tag{4}$$

Note $U_{\alpha B}^{0}$ and R are parameters of boundary potential and they have constant values (see section 5.3.2).

4.3.3 Agents interaction force

Each pedestrian has its own *private sphere* (*territorial effect*) and wants to preserve it also during her/his movement. This implies that the motion of everybody is strongly influenced by the presence of other pedestrians in the environment, which creates a repulsive effect on each other. This force depends on the distance between two pedestrians and can be derived in a general form from a potential function.

$$\vec{f}_{\alpha\beta}(\vec{r}_{\alpha\beta}) := -\nabla_{\vec{r}_{\alpha\beta}} V_{\alpha\beta}\left[b(\vec{r}_{\alpha\beta})\right] \tag{5}$$

where

- $\vec{f}_{\alpha\beta}$: the repulsive force created by the presence of the pedestrian β on the motion of α
- $\vec{r}_{\alpha\beta} := \vec{r}_{\alpha} - \vec{r}_{\beta}$: the distance between the pedestrians α and β

- $V_{\alpha\beta}$: the potential function, which has an exponential decreasing dependence on the distance, e.g. $V_{\alpha\beta}(b) = V^0_{\alpha\beta} e^{-b/\sigma}$, with $V^0_{\alpha\beta} = 2.1\, m^2 s^{-2}$ and $\sigma = 0.3\, m$ are the value proposed by Helbing [3]

- $b := \frac{1}{2}\sqrt{(||\vec{r}_{\alpha\beta}|| + ||\vec{r}_{\alpha\beta} - v_\beta \Delta t \vec{e}_\beta||)^2 - (v_\beta \Delta t)^2}$ the semiminor axis of the ellipse of the equipotential lines which is centered on the agent β and is in the direction of its motion. It allows to model the space needed for a pedestrian for the next step; when β wants to move, it is going to influence α. $v_\beta \Delta t =: s_\beta$ is of the order of the step width of pedestrian β (Δt is the time step)

Furthermore, the perception of the private sphere and its influence on the movement depends on the direction of view. In fact, an agent would not care too much if another agent is just behind her/him because she/he doesn't see her/him. In order to model this effect of perception, direction dependent weights are introduced.

$$w(\vec{e}, \vec{f}) := \begin{cases} 1 & \text{if } \vec{e} \cdot \vec{f} \geq ||\vec{f}|| cos\varphi \\ c & \text{otherwise} \end{cases} \tag{6}$$

where

- $w(\vec{e}, \vec{f})$: the weight dependent on the direction of movement $\vec{e}$ and the one of the force $\vec{f}$

- φ: half of the effective angle of sight 2φ

- c: weaker influence of the force because out of the view ($0 < c < 1$)

The agent interaction force becomes

$$\vec{F}_{\alpha\beta}(\vec{e}_\alpha, \vec{r}_{\alpha\beta}) := w(\vec{e}_\alpha, -\vec{f}_{\alpha\beta}) \vec{f}_{\alpha\beta}(\vec{r}_{\alpha\beta}) \tag{7}$$

A simplified model for the interaction forces is proposed in [4]. This model distinguish the *social* forces from the *physical* forces.

$$\vec{F}_{\alpha\beta} := \vec{F}^{social}_{\alpha\beta} + \vec{F}^{physical}_{\alpha\beta} \tag{8}$$

Social forces The social forces reflect the tendency to preserve the private sphere and, as seen before, depend on the view of the pedestrian (her/his perception of the environment).

$$\vec{F}^{social}_{\alpha\beta}(\varphi_{\alpha\beta}, \vec{r}_{\alpha\beta}) = A^{social}_{\alpha} \cdot \left(\lambda_\alpha + (1-\lambda_\alpha)\frac{1+\cos(\varphi_{\alpha\beta})}{2}\right) \cdot \exp\left[\frac{l_{\alpha\beta} - ||\vec{r}_{\alpha\beta}||}{B^{social}_{\alpha}}\right] \vec{n}_{\alpha\beta} \quad (9)$$

where

- $\vec{F}^{social}_{\alpha\beta}$: the social force on agent α because of presence of β
- A^{social}_{α}: interaction strength of this force
- B^{social}_{α}: the range of the repulsive interaction
- λ_α: parameter of the anisotropy of the interaction force (e.g. $\lambda_\alpha = 0.75$ [4])
- $\varphi_{\alpha\beta}$: the angle between the direction of motion $\vec{e}_\alpha$ and the direction of the force $-\vec{n}_{\alpha\beta}$
- $\vec{n}_{\alpha\beta} := \frac{\vec{r}_{\alpha\beta}}{||\vec{r}_{\alpha\beta}||}$: normalized vector pointing from podestrian β to α
- $l_{\alpha\beta} = l_\alpha + l_\beta$: the sum of the dimension (radii) of each agent (e.g. $l_{\alpha\beta} = 0.6\,m$ [4])

Remark With this notation, the condition 6 for the anisotropy in the previous model can be written as

$$w(\vec{e}, \vec{f}) := \begin{cases} 1 & \text{if } \varphi_{\alpha\beta} \leq \varphi \\ c & \text{otherwise} \end{cases} \quad (10)$$

Physical force The physical forces instead, take account of the physical action that occurs when people are very close and (almost) touch each other. This model neglects frictional effects. Because the physical contact is independent of the view of the agent, these forces are isotropic.

$$\vec{F}^{physical}_{\alpha\beta}(\vec{r}_{\alpha\beta}) = A^{physical}_{\alpha} \cdot \exp\left[\frac{l_{\alpha\beta} - ||\vec{r}_{\alpha\beta}||}{B^{physical}_{\alpha}}\right] \vec{n}_{\alpha\beta} \quad (11)$$

where

- $\vec{F}^{physical}_{\alpha\beta}$: the physical force on agent α because of contact with β
- $A^{physical}_{\alpha}$: interaction strength of this force (e.g. $A^{physical}_{\alpha} = 3\,ms^{-2}$ [4])
- $B^{physical}_{\alpha}$: the range of the repulsive interaction (e.g. $B^{physical}_{\alpha} = 0.2\,m$ [4])

- $\vec{n}_{\alpha\beta} := \frac{\vec{r}_{\alpha\beta}}{||\vec{r}_{\alpha\beta}||}$: normalized vector pointing from podestrian β to α
- $l_{\alpha\beta} = l_\alpha + l_\beta$: the sum of the dimension (radii) of each agent (e.g. $l_{\alpha\beta} = 0.6\,m$ [4])

Remark The parameters $A_\alpha, B_\alpha, \lambda_\alpha, l_\alpha$ depend on the agent α (age, size, culture, etc.)

4.3.4 Total force

The total force at a given time is just given by:

$$\vec{F}_\alpha^{total}(t) = \vec{F}_\alpha^0(\vec{v}_\alpha^0(t)) + \vec{F}_{\alpha B}(\vec{r}_\alpha(t)) + \sum_{\beta\neq\alpha}\left[\vec{F}_{\alpha\beta}^{physical}(\vec{r}_{\alpha\beta}(t)) + \vec{F}_{\alpha\beta}^{social}(\varphi_{\alpha\beta}(t), \vec{r}_{\alpha\beta}(t))\right] \tag{12}$$

4.3.5 Movement equation

The movement of each agent is then governed by the following law:

$$\frac{d\vec{v}_\alpha(t)}{dt} = \vec{F}_\alpha^{total}(t) + \vec{\xi}_\alpha(t) \tag{13}$$

where $\vec{\xi}_\alpha$ is a fluctuation term. However we will neglect this term in our implementation.

Remark This law is just Newton's second law with $m_\alpha = 1$. In fact we do not consider the inertia of the agent but instead we define their desired and maximal velocities.

If the velocity excess the maximal one we have to rescale it by doing:

$$\vec{v}_\alpha(t) := \vec{v}_\alpha^0(t) g(\frac{v_\alpha^{max}(t)}{||\vec{v}_\alpha^0(t)||}) \tag{14}$$

where

$$g(\frac{v_\alpha^{max}(t)}{||\vec{v}_\alpha^0(t)||}) := \begin{cases} 1 & \text{if } ||\vec{v}_\alpha(t)|| < v_\alpha^{max} \\ v_\alpha^{max}/||\vec{v}_\alpha^0|| & \text{otherwise} \end{cases} \tag{15}$$

The movement of an agent α is then:

$$\vec{r}_\alpha(t) = \int \vec{v}_\alpha(t)dt \tag{16}$$

5 Implementation

In this section we will describe how we have implemented our model and the main parts of the code. However, we do not comment here the code. It can be found in the appendix.

5.1 Image acquisition

Research of a hospital ground plan was performed. A ground plan of a project of a "RSA - Residenza Sanitaria Assistita" was chosen. The original ground plan, shown in figure 1, was then modified using the image-editing program Photoshop.

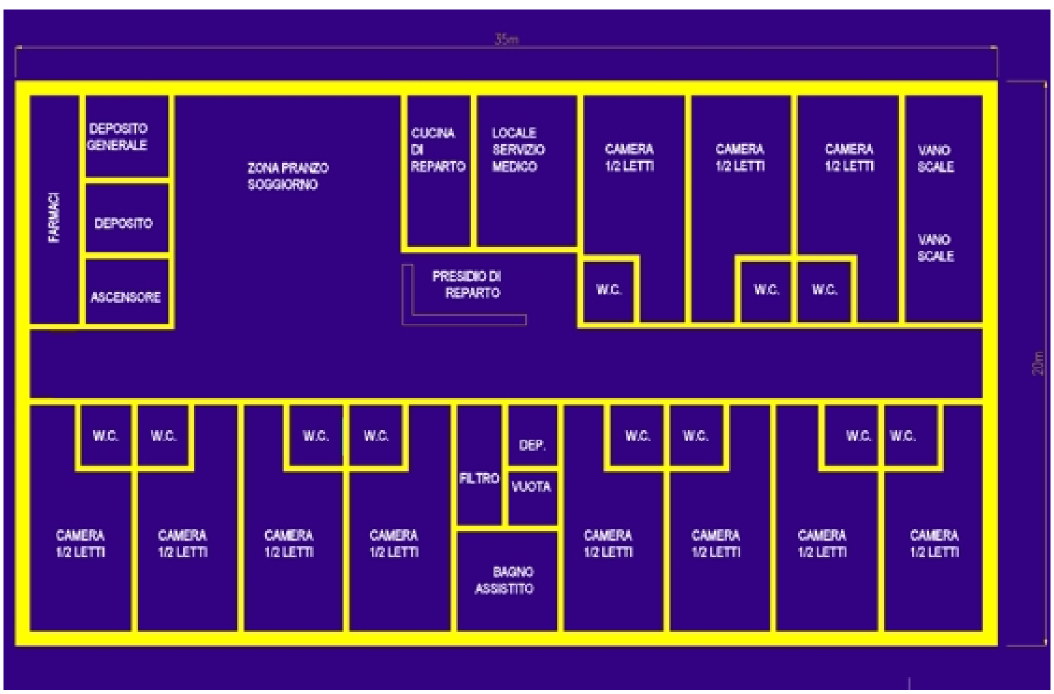


Figure 1: Original ground plan of the chosen hospital

The image was edited so that walls were black, the exit red. After the nude ground plan was created, staff members and patient were included in the image using green and blue respectively. Staff members were placed according to the original position of staff office and pharmacy. Two patients were placed in each room and then divided into ambulant patients and bed patients (light blue). Yellow-colored areas were the areas where people should walk slowly, but we neglected this effect. The resulting image is shown in figure 2. The colors used in image editing where converted into values by saving the image as 4 bit bitmap file (.bmp). The values of the Bitmap file were 4 for free space, 3 for the exit, 0 for the patients, 6 for the bed patients, 1 for the staff and 5 for the walls.

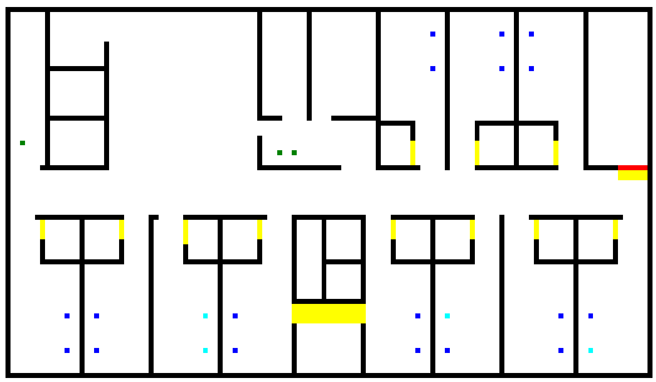


Figure 2: Modified image of ground plan of the chosen hospital

Because of the use of a social force model the creation of a bigger ground plan was needed, permitting the increase of agents number. The bigger hospital plan was made using the real plan as model. The agents were then included in the image using the same table of colors. In the new image slow motion areas were neglected, the closed spaces were opened up and all patients were displayed as ambulant patients. The final results is shown in figure 3. The number of bed-rested patients and their location were simulated using a distributing algorithm.

5.2 Storage of information of agents

The agent informations are stored in a matrix of the form showed in table 1.

Agent initialization The positions are directly stored in the first two rows of the matrix. The number of columns correspond to the number of agents in the simulation. The initial position of the staff is the one of the location of the staff in the provided image. If we choose more staff than on the image, the remaining ones

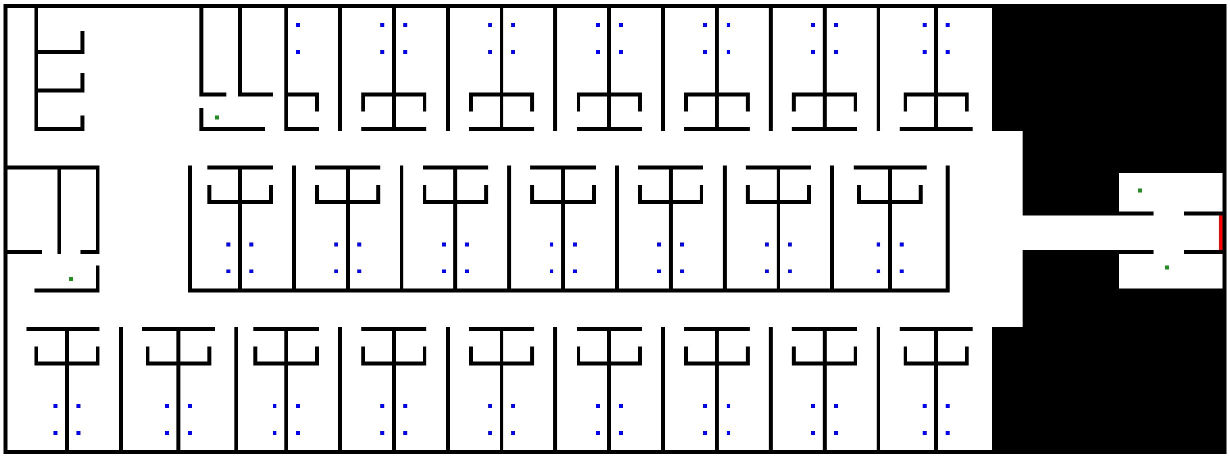


Figure 3: Final plan of the chosen hospital

		ID
1	position x	...
2	position y	...
3	velocity direction x	...
4	velocity direction y	...
5	maximal velocity	...
6	state	...

Table 1: Matrix containing the information of the agents

are attribuated to a random location. The initial positions of walking patients and bedbounded patients are also attribuated randomly from a list of possible locations (rooms).
The initial velocities are setted to zero.
The maximum velocity of each agent follows a random normal distribution, with mean value 1.25 for the staff and 0.95 for the patients; the standard deviation is of 0.05 for all kind of agents. We could have taken also a larger standard deviation but we didn't want to have agents going to slow which could have influenced the results of the simulation in a wrong way (time for evacuation could be huge if somebody is going to slow). When a staff is carrying a bedbounded patient, her/his maximal velocity is reduced to 2/3.

The possible states for the agents are:

- 0: patient is going to the exit directly
- 1: staff is going to a bedbounded patient
- 2: staff is carrying a bedbounded patient toward the exit
- 3: staff reaches the exit
- 4: staff is going back to a bedbounded patient
- 5: staff is going directly to the exit without a bedbounded patient

This repartition into different states is needed for the computation of the forces on all the agents because the parameters depend on the agent type (see 5.3.2, table 4).

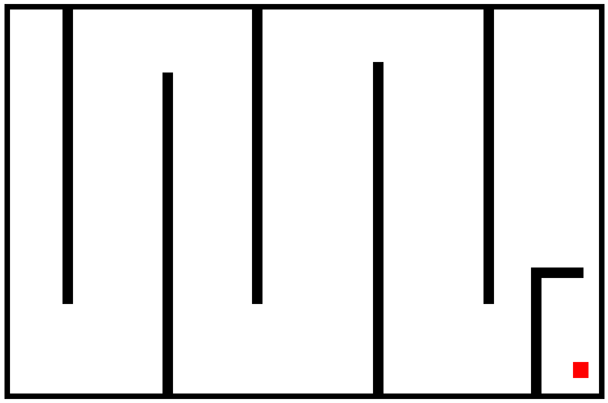
5.3 Social Force Model

5.3.1 Fast Marching Algorithm

As it was mentioned in the section for destination forces (section 4.3.1), the Fast Marching Algorithm (FMA) has been implemented in order to determine the desired direction of motion. Calling the MATLAB function `perform_fast_marching.m` and specifying boundary and exit coordinates as input parameters we will obtain as an output parameter a potential field in which every number will correspond to distance towards designated exit with respect to specified boundaries. At the very exit the value of potential field will be zero, while at the locations which correspond to walls its value will be infinity. A simple example illustrating the results obtained using FMA is presented in figure 4.

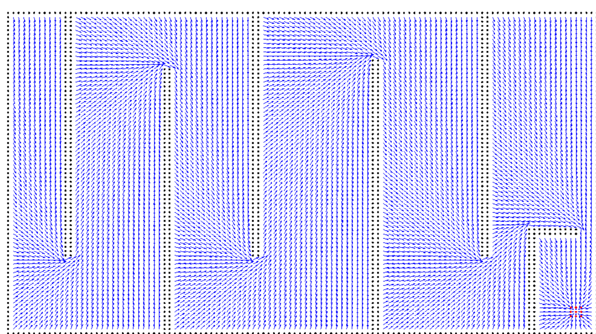
The image 5.3.1 is the one used for generation of the potential field. Exit has been labeled with red color, while boundaries are labeled as black. The results have been visualized in the image 5.3.1 using *hot* color format. At the image 5.3.1 one can see the corresponding vector field which has been obtained by finding the gradient of potential field. This vector field is giving us the desired direction of motion. The arrows are pointing in the direction of the closest path from the free points towards the exit. It is important to underline that this vector field has not yet been normalized and the normalization is implemented inside MATLAB function `destination.m`. The potential and corresponding vector fields of image relevant for modeling are presented in the figure 5.

Finding the gradient of potential field wasn't trivial task. Since the value of the potential field at the positions which correspond to boundaries will be infinity, implementing simple MATLAB function `gradient.m` will not give us correct results



(a) Geometry

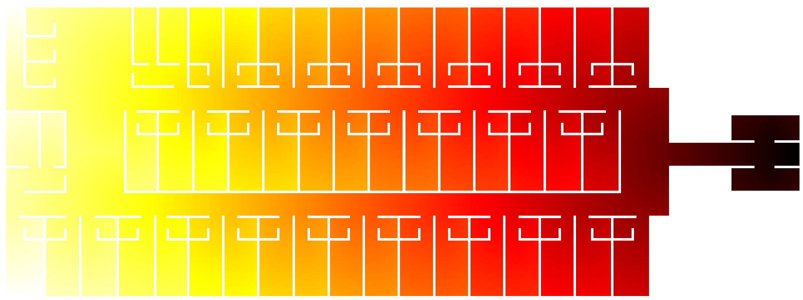
(b) Potential field



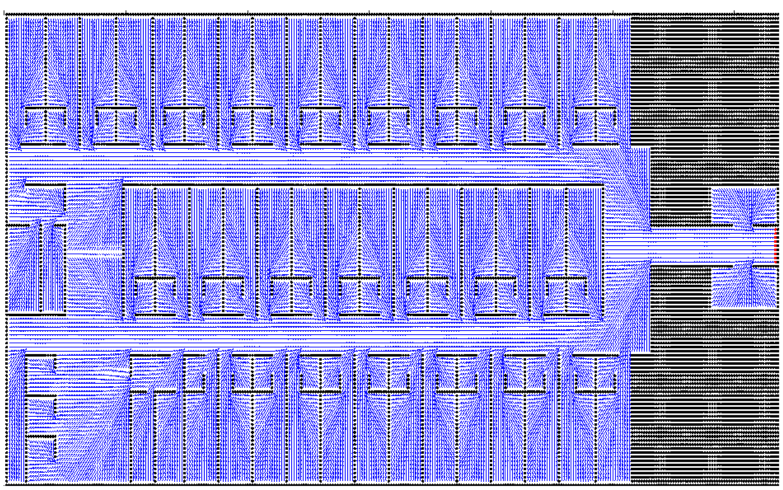
(c) Vector field

Figure 4: Example of computing vector field with Fast Marching Algorithm

near walls. For this reason a new function `grad.m` has been written which successfully copes with this problem.



(a) Potential field



(b) Vector field

Figure 5: Computation of the vector field with Fast Marching Algorithm

5.3.2 Parameters of the model

As we have seen in section 4.3 there are several parameters of the social force model that have to be tuned to fit our case. Furthermore, we have three types of agents

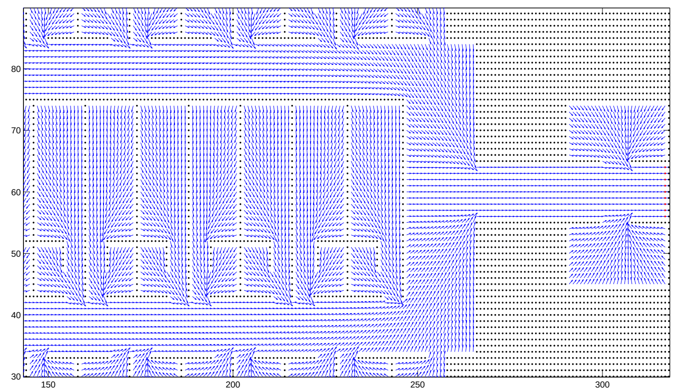


Figure 6: Zoom of the figure 5

that have different characteristics:

- staff
- walking patient
- staff carrying a bedbounded patient

Remark However for the destination and boundary forces we consider as if all the agents behave in the same manner.

Destination force The parameters for the destination forces that we have chosen after tuning are reported in table 2.

τ_{alpha}	0.5
v_α^0	1.34

Table 2: Destination force parameters

Boundary force The parameters for the boundary forces that we have chosen after tuning are reported in table 3.

The field of the boundary forces is shown in figure 7.

$U^0_{\alpha B}$	100
R	0.4

Table 3: Boundary force parameters

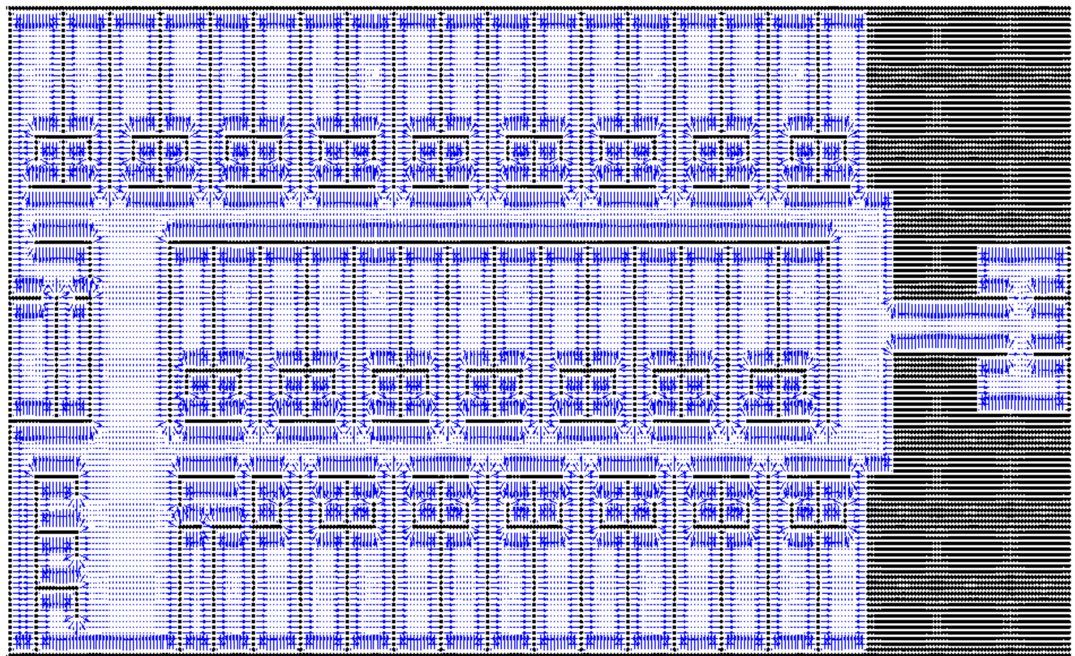


Figure 7: Boundary force field

Agents interaction force We implemented both simplified and non-simplified model for the interaction force between agents. However, we have used only the simplified model in order to distinguish the physical force from the social one.

In table 4 are shown the parameters that we have chosen for the staff, for the patient and for the staff carrying a bedbound patient.

These values are tuned such that they represent the type of agent, e.g. the agents carrying a bedbounded patient are much more larger than the others or the patients need also some more space because of their limited mobility.

In figure 8 are shown the isolines of the intensities of the physical and social forces of an agent α moving in direction $(0, 1)^T$ for all possible position of the interacting agent β.

We can remark that the social force is anisotropic. The anisotropy will increase

	Staff	Patient	Staff with bedbound
A^{social}_{alpha}	5	6	5
B^{social}_{alpha}	0.3	0.6	0.7
$A^{physical}_{alpha}$	2	3	5
$B^{physical}_{alpha}$	0.2	0.3	0.5
λ_{alpha}	0.3	0.3	0.3
l_{alpha}	0.3	0.4	1

Table 4: Agents interaction force parameters

with a decreasing parameter λ_{alpha}.

Remark Actually in the code we implement a mix between a Cellular Automata Model and the Social Force Model. That is, in order to make the simulation less expensive, we consider only the interaction forces if the agents α and β are within a distance of $3\,m$.

Test simulation In order to be able to tune the parameters we have created a basic simulation where we have the agents on one side of a wall and the exit on the other side.

5.3.3 Time discretization

The governing equation 13 is discretized with an explicit Euler scheme:

$$\vec{F}^{total}_{\alpha}(t) = \frac{d\vec{v}_{\alpha}(t)}{dt} \simeq \frac{\vec{v}_{\alpha}(t) - \vec{v}_{\alpha}(t-\Delta t)}{\Delta t} \tag{17}$$

so that

$$\vec{v}_{\alpha}(t) \simeq \vec{v}_{\alpha}(t-\Delta t) + \vec{F}^{total}_{\alpha}(t) \cdot \Delta t \tag{18}$$

and

$$\vec{r}_{\alpha}(t) \simeq \vec{r}_{\alpha}(t-\Delta t) + \vec{v}_{\alpha}(t) \cdot \Delta t \tag{19}$$

Remark Other schemes which are more stable could have been used. Instead, to ensure stability we choose a small $\Delta t = 0.1$.

(a) Staff

(b) Patient (walking)

(c) Staff with bedbound patient

Figure 8: Physical and social force intensities with the parameters given in table 4

5.4 Code structure

We describe briefly the structure of the main programs.

```
initialization
        number of agent per type for the simulation
        get the image
                initialization of agent position
                initialization of agent parameter
                bed coordinates
                wall coordinates
                exit coordinates
        assign the bed to the staff
        compute the gradient maps
                of the exits
                of the beds
        compute the corresponding initial destination force
        compute the static boundary force
        compute the initial interaction force between agent
        initialize counters for statistics
loop in time
        update the velocity
        update the position
        loop over agents
                look if staff close to a bed
                look if agent close to a exit
                update agent state
                depending on the state compute the total
                   force
                update counters
                storage of position for further
                   visualization
        end loop
        plot the situation/statistics
end loop
```

5.5 Main code versions

As it can be seen in the appendix we did two main programmes. The former one, `evacuation_main.m`, was first created. In this code, all the information concerning an agent are deleted when she/he reaches the exit. This created some problems

in the attribution of targets because of the continuous changing of corresponding matrix size. To solve this, part of the group implemented another way where the informations are linked with connectivity arrays in order to have a solid connection between agents and their goals (`evacuation.m`). At the same time, the other part of the group was able to fix the problem in the former code.
The structure of both codes is the same and is the one presented in the previous section 4.3.1.

However, from now on, the discussion will be based on the later version of the main code.

5.6 Visualization

The need for visualisation of the simulation occurred at two stages. Primarily when creating the program and bug fixing it was necessary to be able to visualise in real time the result at each timestep. The second was required to create videos of completed simulations for presenting the results, and demonstrating interesting phenomenon.

In section 5.4 the structure of the program was introduced, to explain the methods used for visualisation it's important to refer to this and discuss in more detail the data structures used for the program. Within the program the active agents are stored in a linked list, which can dynamically change size as agents leave the simulation. To view in real time the location of the agents, the linked list can be passed to the visualisation program, and agents who have left the simulation will not be rendered. However when saving a time history for a simulation the problem arises that when using a single array to store the state this cannot change size as the simulation progresses. While it is possible to store each state as an array in an array of structures which store the data at each time point, this method would not allow pre-allocation of memory which is a larger pitfall. The data structures in which historical data are stored are illustrated below in table 5.

The agent history file contains 'stacks' of the agent (table 1) states from the program, their locations, velocities, and also the state of each agent. Any agents who were no longer in the linked list would have their attributes stored as zero. The number of agents array (see table 6) contained purely the number of those agents still in the simulation.

The first category as mentioned was used as a debugging tool, allowing group members to visualise the simulation in real time. This also allowed them to quickly

$t = 1$	1	2	3	4...
X				
Y				
V_x				
V_y				
$state$				
$t = 2$	1	2	3	4...
X				
Y				
V_x				
V_y				
$state$				
$t = 3$	1	2	3	4...
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 5: Matrix containing the information of the agents in time

time	number of a agent type
1	17
2	17
3	17
4	16
$\vdots$	$\vdots$

Table 6: Array containing the number of a certain agent type during time

cancel simulations when it was seen that the desired effect was not achieved. It also had functionality to display frames from the history matrix, which also allowed the programmer to march backwards through time to see exactly when erratic/wrong behaviour in the agent began.

This visualisation function was designed to work with all the existing structures in the simulation, so the code could be inserted at the end of a simulation loop, to generate the image. The use of existing data structures allowed the code to run quite rapidly, so reducing the strain on memory usage and CPU usage. It also allowed it to be compatible with all code revisions which maintained consistency with these standards used within the group. To run it the function was passed the agent state matrix, and the linked list, this allowed the function to display the agents still in the simulation. One drawback for de-bugging is that the linked list was not stored historically, so when visualising using the historical records, the agents who had already exited where put at position zero. However this did not affect the functionality, and at the sacrifice of some speed it would be possible to eliminate these with some logical checks. The second category was created to visualise the results of a batch of simulations. These visualisations where required for the presentation and the report. The figure 9 shows a frame from such a render.

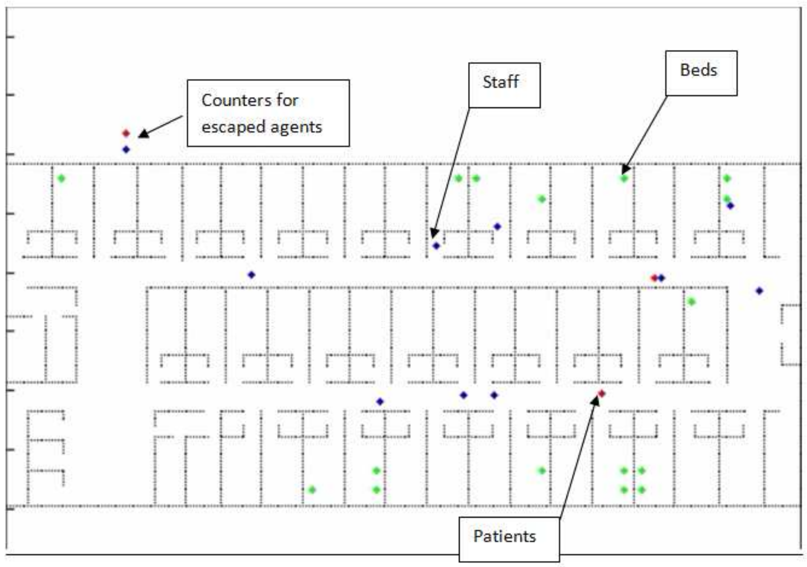


Figure 9: Frame of a simulation

Within this figure it's possible to see a variety of information. Clearly it shows the geometry of the hospital floor, and the location of the agents within. The agents are also identified by colour, with staff blue, walking patients red and bed bound patients in green. There are also three counters in the top left, these counters represent the agents who have exited the simulation. A typical simulation would have in excess of 5000 time steps, As mentioned above the agents who had left the simulation have a value in the history array of zero, the code was written in such a way that it could catch these zero locations, and use this for creating a graphical counter of the moving agents. For the animation of these results, 4 arrays where used.

- The staff/patient history array
- The bed coordinate array

- The wall coordinate array
- The bed patient history array

The staff patient array contained all the x-y coordinates of all the staff and patients. As the number of staff where known, it was possible to separate them due to staff and patients occupying separate sections of the array. The bed coordinates changed randomly between simulations, so these where stored with the other results. The wall coordinates were something global to a family of simulations and so where stored separate to the results. The bed patient history array was used for the counter which displays the rescued patients.

6 Simulation Results and Discussion

6.1 Batch Simulations

The parameter study we had planned required running multiple simulations, initial 10 different ratios of staff to patients, 10 ratios of mobile and immobile patients and 8 different numbers of agents, this came to a total of 800 simulations. This was an impossible amount to run, on the computing power available to us, however we attempted to split the work up between different group member's computers.
To do this the main simulation was converted into a function, which could be passed these different parameters, and would return all of the historical data, which would then be stored in a structure with all the details of the simulation parameters.
It was seen as a beneficial to try and run multiple simulations in parallel, so within the driver function we changed one of the loops to a parfor loop.
The simulations with varying numbers of staff to patients was chosen to be ran in parallel, while the total number of agents and number of bedded patients was changed outside of the parallel loop. The logic was that the complexity of the simulation depended on the total number of agents, and so the simulations ran in parallel would have similar running times. Also the number to or parameters searched with the parallel loop was always a multiple of the number of threads available, this was to maximise the amount the whole CPU of a machine was used.
Several problems where encountered, first was that if an error occurred during a simulation the whole program stopped. An error catcher was implemented, but this was not compatible with the parfor, and while the parfor still ran, it did not prevent errors from cancelling the batch. The second was that the indices of the structure of results could not combine the parfor looped variable with other looped variables. This was a safety feature in MATLAB, which was in this case unnecessary. This meant that the results structures ended up by 3 dimensional arrays of structures, which was as easy to use as it is to say aloud. A final problem was encountered, a bug in the forces meant that if a person was positioned close enough to a wall they ended up trapped by the boundary. This caused several results to run until the end of the pre-allocated memory for the results, meaning the total evacuation time was not immediately available. To correct this a function was made that would find the total number of agents stuck in the wall, and then find the time step where everyone else had exited. This was saved then in the structure as the total time.

Remark We are running our simulation with random initial location of the agents. Because of time reasons we can not afford to run many simulations with the same parameters and average them. We will assume that the mesh is fine enough to smear out those random effects.

6.2 Varying the ratio of agents type

Once enough simulations had been run, it was necessary to create a meaningful interpretation of them. The script `iso_surface` was made. This took the results structure and the vectors of runtime parameters used, and created a stack of arrays based on the number of agents as the z-parameter, and the other two parameters as x and y. It was then possible to create 3d surfaces and colour plots of how these parameters effect the run time. The term iso-surface comes from how each variation of the number of agents created an iso-surface.
It then became important to visualise the evacuations, and understand what phenomenon occurred. The total number of simulations was something close to 128, which would have taken too much time to generate and watch videos of them all.

Two iso-surfaces for 50 agents in total are shown in figures 10 and 11. The first one is obtained by using the new code (`evacuation.m`) and the second is obtained by using the former code (`evacuation_main.m`). Unfortunately we couldn't save the images as .eps with MATLAB because it would have smeared out the result.

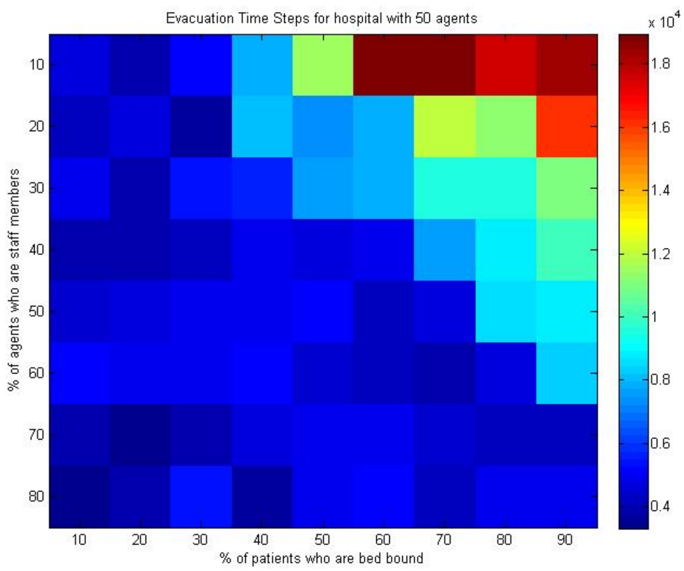


Figure 10: Evacuation time for different $\%_{staff/agent}$ and $\%_{bed/patients}$ for a total of 50 agents (with `evacuation.m`)

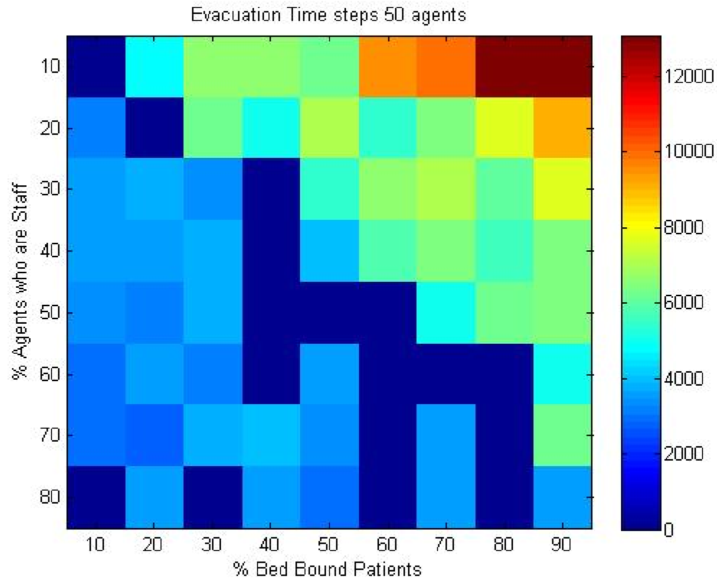


Figure 11: Evacuation time for different $\%_{staff/agent}$ and $\%_{bed/patients}$ for a total of 45 agents (with (`evacuation_main.m`))

Remark In figure 11 we can see that time $t = 0$ is obtained for some simulations. This is due to the fact that for these simulations we have encountered an error. This error is caused by using the function for random distribution of bed positions with old code.

The obtained results confirm our hypothesis. If we increase the percentage of bedbounded patients and decrease the ratio of staff, the time for the evacuation increases drastically.
Secondly, for a fixed ratio of staff to agents, we can see an increase of evacuation time when the ratio of bedbounded patients increases. The reason for this is the fact that the staff have to go back to rescue the other patients. This is even more significant when the number of staff is small, where we have a kind of exponential increase.
Furthermore, by keeping the ratio of bed patients constant, we can observe that if we decrease the number of staff the evacuation time increases too, but not so dramatically as for an increase of bedbounded patients for a given staff ratio.
Another observation is that there is often a minimum of the evacuation time for a given number of bedbounded patients. If we have a high number of bed patients it is necessary to have a high staff to agent ratio in order to have reasonable evacuation time. On the other side, for not too many bed bounded patients, it is not strictly

necessary to have a larger percentage of staff.
We can argue that if we had too many staff in comparison to the bed patients, the evacuation time will be larger. This will also be the case when we increase the number of total agents. In this case the staff that have to go back will interact mutually and with other patients which would result in prolonged evacuation time. In this case, the social force model will become even more important and we could have some interesting effects as bottle necks, strong counterflow, etc.

Remark We were able to observe these effects, but because of the lack of time and duration of the simulation we weren't able to perform more detailed analysis of the mentioned effects. To detect group interaction, a function `find_groups.m` was made, which took the results structure and searched through it to determine within a set of radii, when the maximum number of people are within them, and at which time. The different sets of radii allowed for a 'filtering' of the pile-ups, as if the maximum number of people within the larger radii occurred before the maximum within a smaller, it is likely that there is a pile-up which started with a sparse group of agents, and who due to a blockage ended up becoming a denser crowd.

Here below in figure 12 we show some interesting captured images of groups of agents, where clearly the social forces between the people were important.

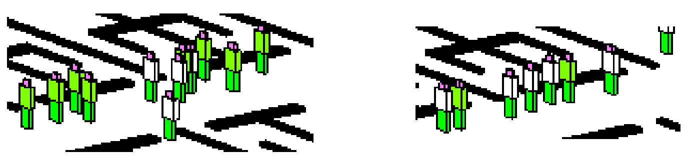
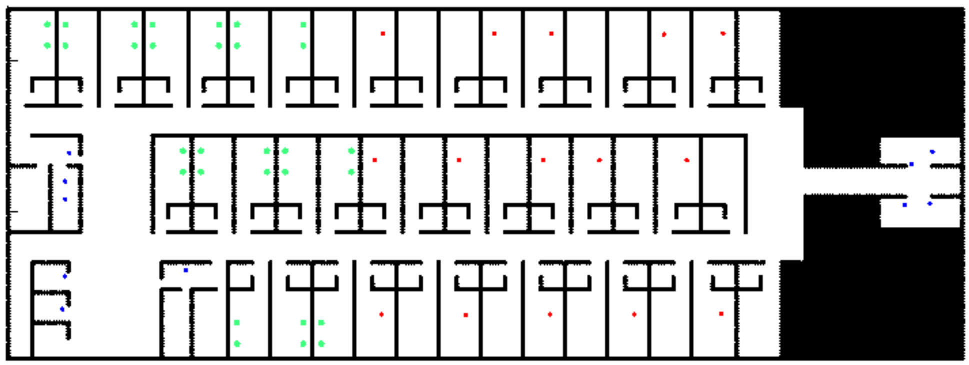


Figure 12: Example of interaction between agent

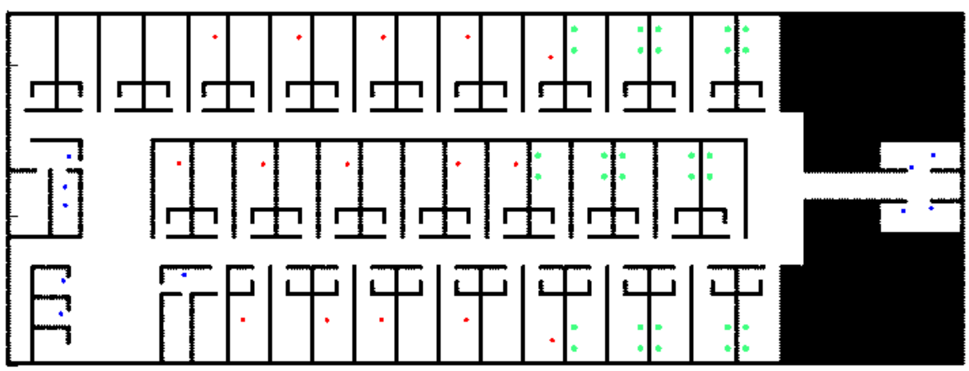
6.3 Varying location of bedbound patients

In order to answer the question about the best location of beds, we run three simulations where in the first one we have a homogeneous distribution of bedded patients in the hospital, in the second we assumed that all the beds are close to the exit and in the last we put all bedded patient far away from the exit. In each simulation we have 30 beds, 15 walking patients and 10 staffs members (see figure 13).
In order to compare the evacuations, we plotted in the figure 14 the number of all

the patients in the building during time. The evacuation time for every configuration is summarized in table 7.



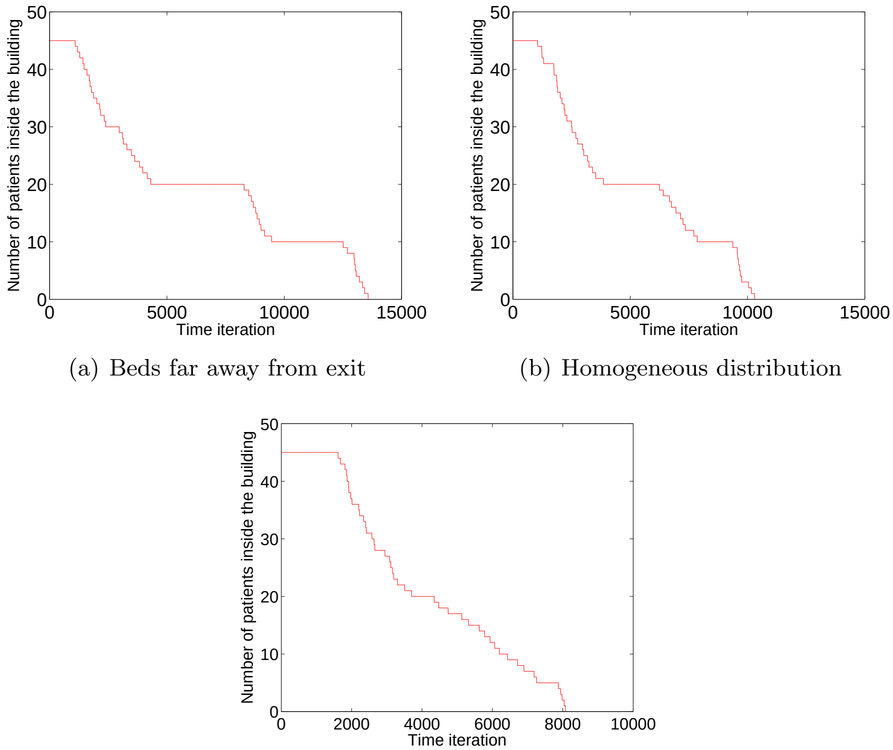
(a) Beds far away from exit



(b) Beds close to exit

Figure 13: Locations of bed

We can observe how the location of the beds influence the time for the evacuation. The time laps during which no staff reach the exit correspond to the time that the



(a) Beds far away from exit

(b) Homogeneous distribution

(c) Beds close to exit

Figure 14: Number of agents during time for different locations of bed

staff need to go back to rescue the bedbounded patients. In the case where the beds are close to the exit, this lapse of time is very short and the flow of patients being rescued is almost constant. Furthermore we can see that the number of patients decreases rapidly at the beginning because of the walking patients reach the exit by themselves.

This analysis shows us that it is better to put the patients with the less ability to move close to the exit. The evacuation time will be lower, as well as the risk that the staff encounter when they go back in the building is decreased.

Location of beds	Time for evacuation
homogeneous	10301
far from exit	13575
close to exit	8075

Table 7: Time needed for evacuation for different location of bed

7 Summary and Outlook

7.1 Summary

The modelling of complex social situations with multiple agent types, and individual agent objectives is a large extension of a simple social force model. In our case we had just two agent types with 3 agent states (going to exit, going to bed, taking bed to exit), however as each bed is only visited by a single agent, this translated to every bed being a unique objective. The linking of the agents to different objectives was a difficult process, as the list of uncompleted objectives was completely changing as were too the list of agents. We found that using linked lists was a flexible method, which allowed for a transparent allocation of agents and objectives, which was easily customised. As it was mentioned before this method was implemented in the new code. However this came with erosion in the computational speed, compared to the old code where matched indexes where the arrays of the objectives and agents were arranged so the indexes correspond. However this method was extremely inflexible, and required the matrices of objectives to be resized, which increased the likelihood of bugs occurring when for example other corresponding matrices (e.g. forces) where not resized.

We also saw quite interestingly that the ratios of staff to agents, and mobile to immobile patients, both effected the evacuation time differently. The staffs to agents have an exponential decay, while the mobile to immobile is a parabolic decay (see figures 10 and 11). It can be concluded that the appropriate level of staff in a hospital is critical, as this is a controllable factor and has as stated a exponential decay on the evacuation time, whereas the number of bed bound patients is not something which can be easily determined, and the condition of patients in a hospital is constantly changing, which makes also difficult putting them to the most convenient location.

7.2 Outlook

There are several interesting improvements that can be done in the future about our project. Here are some proposals:

- correct the errors that we have encountered during the parametric study
- run more simulations with different parameters to obtain smoother results
- average over the simulations to get the mean values
- implement more floors of the hospital
- study more in detail the influence of the Social Force Model and its parameters
- test new geometries of buildings
- test different strategies for the evacuation
- implement the real form of bed (this could be done e.g. by using elliptic physical forces)
- introduce the relatives of the patients in the hospital

8 Reference

1. K. Taaffe, M. Johnson and D. Steinmann: *Improving hospital evacuation planning using simulation.* Winter Simulation Conference (2006)

2. C. Johnson: *Using computer simulations to support a risk-based approach for hospital evacuation.* University of Glasgow, Department of Computing Science Briefing (2006)

3. Dirk Helbing and Peter Molnar: *Social force model for pedestrian dynamics.* Physical Review 51,5, 4282-4286 (May 1995)

4. Dirk Helbing et al.: *Self-Organized Pedestrian Crow Dynamics: Experiments, Simulations, and Design Solutions.* Transportation Science 39,1, 1-24 (February 2005)

5. Dirk Helbing et al.: *Simulating dynamical features of escape panic.* Nature, 407, 487-490 (July 2000)

6. Philipp Heer and Lukas Buhler: *Pedestrian Dynamics: Airplane Evacuation Simulation* (May 2011)

7. Moritz Vifian, Matthias Roggo and Michael Aebli: *Modelling Crowd Behaviour in the Polymensa using the Social Force Model* (December 2011)

A getfile | getfile.m

```
function [agent, I, wall, exits, beds] = getfile()
% This function converts a bmp file to a matrix.
% Initialization of the agents as output

%clear all
%clc
av_staff_vel=1.25;
stdev_staff_vel=0.05;

av_patient_vel=1;
stdev_patient_vel=0.05;

exit=0;
while exit==0
    [FileName,PathName]=uigetfile('*.bmp', 'Select a Bitmap File');
    I=imread(strcat(PathName,FileName));

    if (find(I>6))
        exit=0;
        uiwait(msgbox('Wrong file'));
    end

    space=find(I==4);
    [exits]=find(I==3);
    [exit_x exit_y]=ind2sub(size(I),exits);
    exits=[exit_x';exit_y'];
    patient=find(I==0);
    beds=find(I==6);
    [bed_x bed_y]=ind2sub(size(I),beds);
    beds = [bed_x';bed_y'];
    staff=find(I==1);
    wall=abs((I==5)-1);

    %imshow(I,[])
    %Create staff member and patients "matrix"
    n=1;
    m=1;
    %Is im(x,y) a staff member?
    for x=1:size(I,2)
        for y=1:size(I,1)
            if (I(y,x)==1)
                agent.staff(1,n)=x;
                agent.staff(2,n)=y;
                agent.staff(3,n)=0;
                agent.staff(4,n)=0;
                agent.staff(5,n)=normrnd(av_staff_vel,stdev_staff_vel); % desired velocity
                agent.staff(6,n)=1; % 1 = free (going to a patient); 2 = with a bed (going to the exit); at the beginning all the staff are going
                n=n+1;
            end
            %Is im(x,y) a patient?
            if (I(y,x)==0)
                agent.patient(1,m)= x;
                agent.patient(2,m)=y;
                agent.patient(3,m)=0;
                agent.patient(4,m)=0;
                agent.patient(5,m)=normrnd(av_patient_vel,stdev_patient_vel);
                agent.patient(6,m)=0; % going to a exit (patient is free, it will always be the case, he is always going to the exit only)
                m=m+1;
            end
        end
    end


    if (m==1)
        agent.patient(1,m)=0;
        agent.patient(2,m)=0;
        agent.patient(3,m)=0;
        agent.patient(4,m)=0;
        agent.patient(5,m)=0;
        agent.patient(6,m)=0; % going to a exit (patient is free, it will always be the case, he is always going to the exit only)
    end



    exit=1;
```

```
end
```

B getfile_rand_staff | getfile_rand_staff.m

```
function [agent, I, wall, exits, beds] = getfile()
% This function converts a bmp file to a matrix.
% Initialization of the agents as output

%clear all
%clc
av_staff_vel=1.25;
stdev_staff_vel=0.05;

av_patient_vel=1;
stdev_patient_vel=0.05;

exit=0;
while exit==0
    [FileName,PathName]=uigetfile('*.bmp', 'Select a Bitmap File');
    I=imread(strcat(PathName,FileName));

    if (find(I>6))
        exit=0;
        uiwait(msgbox('Wrong file'));
    end

    space=find(I==4);
    [exits]=find(I==3);
    [exit_x exit_y]=ind2sub(size(I),exits);
    exits=[exit_x';exit_y'];
    patient=find(I==0);
    beds=find(I==6);
    [bed_x bed_y]=ind2sub(size(I),beds);
    beds = [bed_x';bed_y'];
    staff=find(I==1);
    wall=abs((I==5)-1);

    %imshow(I,[])
    %Create staff member and patients "matrix"
    n=1;
    m=1;
    %Is im(x,y) a staff member?
    for x=1:size(I,2)
        for y=1:size(I,1)
            if (I(y,x)==1)
                agent.staff(1,n)=x;
                agent.staff(2,n)=y;
                agent.staff(3,n)=0;
                agent.staff(4,n)=0;
                agent.staff(5,n)=normrnd(av_staff_vel,stdev_staff_vel); % desired velocity
                agent.staff(6,n)=1; % 1 = free (going to a patient); 2 = with a bed (going to the exit); at the beginning all the staff are going
                n=n+1;
            end
            %Is im(x,y) a patient?
            if (I(y,x)==0)
                agent.patient(1,m)= x;
                agent.patient(2,m)=y;
                agent.patient(3,m)=0;
                agent.patient(4,m)=0;
                agent.patient(5,m)=normrnd(av_patient_vel,stdev_patient_vel);
                agent.patient(6,m)=0; % going to a exit (patient is free, it will always be the case, he is always going to the exit only)
                m=m+1;
            end
        end
    end


    if (m==1)
        agent.patient(1,m)=0;
        agent.patient(2,m)=0;
        agent.patient(3,m)=0;
        agent.patient(4,m)=0;
        agent.patient(5,m)=0;
        agent.patient(6,m)=0; % going to a exit (patient is free, it will always be the case, he is always going to the exit only)
    end
```

```
    exit=1;
end
```

C destination | destination.m

```
function [fx_dest,fy_dest]=destination(e_x,e_y,v_x,v_y)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function is used for determination of destination    %
% forces. Force has a form of an acceleration term and for  %
% its determination one needs components of desired dire-   %
% ction vector (e_x,e_y), which are obtained using FMA, and %
% actual velocity vector (v_x,v_y).                         %
%                                                           %
% INPUT arguments:                                          %
% e_x - desired x-direction, obtained from gradientmap.f     %
% e_y - desired y-direction, obtained from gradientmap.f     %
% v_x - actual velocity in x-direction, from agent.struct    %
% v_y - actual velocity in y-direction, from agent.struct    %
% OUTPUT arguments:                                          %
% fx_dest - destination force in x-direction                 %
% fy_dest - destination force in y-direction                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%parameters%%%%%%%%%%%%%%%%
  %relaxation time        %
     tau_alpha=0.5;       %
  %mean desired velocity  %
     v0_mean=1.34;        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%

%normalization of desired direction%
norm=sqrt(e_x.^2+e_y.^2);          %
e_x=e_x./norm;                     %
e_y=e_y./norm;                     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Cooping with possible deviations%%%
deviation1=find(e_x==inf);         %
deviation2=find(e_y==NaN);         %
                                   %
e_x(deviation1)=0;                 %
e_x(deviation2)=0;                 %
                                   %
e_y(deviation1)=0;                 %
e_y(deviation2)=0;                 %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%destination forces (see Report-Destination forces)%
fx_dest=(1/tau_alpha).*(v0_mean.*e_x-v_x);         %
fy_dest=(1/tau_alpha).*(v0_mean.*e_y-v_y);         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

D grad | grad.m

```
function [e_x,e_y]=grad(A)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Using Fast Marching Algorithm (FMA) one can obtain potential filed.  %
% In order to obtain vector field of desired directions one must find  %
% gradient of previously obtained potenial field. Since wall positions %
% in potential field have value of inf, it is not possible to use      %
% MATLAB function gradient. For this reason the following function was %
% written and it allows to cope with the problem in effective way.     %
%                                                                      %
% INPUT arguments:                                                      %
% A-input matrix                                                        %
% OUTPUT arguments:                                                     %
% e_x - gradient in x-direction                                         %
% e_y - gradient in y-direction                                         %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
[a b]=size(A);
e_x=zeros(a,b);
e_y=zeros(a,b);

for m=1:a
    for n=1:b

        %X-direction
        if (A(m,n)~=inf && n>1 && n<b)
           if (A(m,n-1))==inf
               e_x(m,n)=A(m,n)-A(m,n+1);
           elseif (A(m,n+1)==inf)
               e_x(m,n)=A(m,n-1)-A(m,n);
           else
               e_x(m,n)=(A(m,n-1)-A(m,n+1))/2;
           end
        end

        if (A(m,n)==inf && n>1 && n<b)
           if (A(m,n-1)==inf || A(m,n+1)==inf)
               e_x(m,n)=0;
           elseif (A(m,n-1)==inf && A(m,n+1)==inf)
               e_x(m,n)=0;
           else
               e_x(m,n)=0;
           end
        end

        %Y-direction
        if (A(m,n)~=inf && m>1 && m<a)
           if (A(m-1,n))==inf
               e_y(m,n)=A(m,n)-A(m+1,n);
           elseif (A(m+1,n)==inf)
               e_y(m,n)=A(m-1,n)-A(m,n);
           else
               e_y(m,n)=(A(m-1,n)-A(m+1,n))/2;
           end
        end

        if (A(m,n)==inf && m>1 && m<a)
           if (A(m-1,n)==inf || A(m+1,n)==inf)
               e_y(m,n)=0;
           elseif (A(m-1,n)==inf && A(m+1,n)==inf)
               e_y(m,n)=0;
           else
               e_y(m,n)=0;
           end
        end

        if (n==1 || n==b)
            e_x(m,n)=0;
        end
        if (m==1 || m==a)
            e_y(m,n)=0;
        end

    end
end
```

E gradientmap | gradientmap.m

```
function [e_x e_y]=gradientmap(wall, sink)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% This function is used for formation of gradient map of desired   %
% direction. For the creation of potential filed appropriate tool- %
% box for FastMarchingAlgorithm (FMA) has been used.               %
%                                                                  %
% INPUT arguments:                                                 %
% wall - wall map obtained from getfile.m                          %
% sinn - exit coordinates obtained from getfile.m                  %
```

```
10 % OUTPUT arguments:                                                %
11 % e_x - desired x-direction (unnormalized!)                        %
12 % e_y - desired y-direction (unnormalized!)                        %
13 % Normalization is done in destination.f                           %
14 %                                                                  %
15 % Function perform_fast_marching().m can be founf in toolbox_fast_ %
16 % _marching folder.                                                %
17 %                                                                  %
18 % Note: For finding the gradients of potential field, MATLAB       %
19 % function gradient should not be used (see Report - FMA. For this %
20 % purpose a new function grad.f has been written.                  %
21 %                                                                  %
22 % To obtain plots and visualize the destination force direction    %
23 % uncomment lines 33-45.                                           %
24 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
25
26 options.nb_iter_max=inf;
27
28 [D S]=perform_fast_marching(wall,sink,options);
29 %figure,imshow(D,[])
30
31 [e_x,e_y]=grad(D);
32
33 % [y_wall x_wall]=find(wall==0);
34 %
35 %  figure
36 %  quiver(e_x,e_y,1),hold on
37 %
38 %  for i=1:size(x_wall)
39 %      plot(x_wall(i),y_wall(i),'.k','LineWidth',1.5),hold on
40 %  end
41 %
42 %  for i=1:size(sink,2)
43 %      plot(sink(2,i),sink(1,i),'.r','LineWidth',1.5),hold on
44 %      end
45 %  end
46
47
48
49
50
```

F boundary | boundary.m

```
1 function [fx_bound fy_bound]=boundary(A)
2 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
3 % Function for computation of boundary forces.             %
4 %                                                          %
5 % INPUT arguments:                                         %
6 % A - input matrix (wall matrix obtained from getfile.m)   %
7 % OUTPUT arguments:                                        %
8 % fx_bound - boundary force in x-direction                 %
9 % fy_bound - boundary force in y-direction                 %
10 %                                                         %
11 % To obtain plots and visualize the boundary forces       %
12 % uncomment lines 51-56.                                  %
13 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
14
15 %Parameters of boundary potential%
16 u0_alphaB=150;                   %
17 R=0.4;                           %
18 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
19
20 %Determination of boundary potential%%
21 [a b]=size(A);                       %
22 r=zeros(a,b);                        %
23 [y x]=find(A==0);                    %
24 for m=1:a                            %
25     for n=1:b                        %
26         if (A(m,n)~=0)               %
27             rx=n-x;                  %
28             ry=m-y;                  %
29             rr=sqrt(rx.^2+ry.^2);    %
30             r(m,n)=min(rr);          %
31         end                          %
```

```
        if (A(m,n)==0)              %
            r(m,n)=0;               %
        end                         %
    end                             %
end                                 %
                                    %
r(r==0)=-inf; % Explanation:        %
% value of r=0 is obtained for wall  %
% point. By making value of r at the %
% wall r=-inf we'll get that bounda- %
% ry potential is equal to inf at    %
% this point and now gradient of bo- %
% undaru potential can be found      %
% using grad.m                       %
bound_pot=u0_alphaB.*exp(-r./R);     %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[fx_bound fy_bound]=grad(bound_pot);

% figure
% quiver(fx_bound,fy_bound,2),hold on;
% [y_wall x_wall]=find(A==0);
%  for i=1:size(x_wall)
%      plot(x_wall(i),y_wall(i),'.k','LineWidth',1.5),hold on
%  end



```

G attribuate_bed | attribuate_bed.m

```
function [agent_staff,residual_staff,init_bed_coords,rest_bed_coords,final_position,d_staff_bed,d_staff_bed_backup1]=attribuate_bed(agent,bed_coo
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

agent_staff=agent.staff;
nb_staff=size(agent_staff,2);
nb_beds=size(bed_coords,2);
d_staff_bed=zeros(nb_staff,nb_beds);

for m=1:nb_staff
    for l=1:nb_beds
        d_staff_bed(m,l)=sqrt((agent_staff(1,m)-bed_coords(2,l))^2 + (agent_staff(2,m)-bed_coords(1,l))^2);
    end
end

final_position=zeros(1,nb_staff);
d_staff_bed_backup=d_staff_bed;
d_staff_bed_backup1=d_staff_bed;

% %more primitive
% for m=1:nb_staff
%     [temp_min,temp_min_position]=min(d_staff_bed(m,:));
%     final_position(m)=temp_min_position;
%     d_staff_bed(:,temp_min_position)=inf;
% end

%more advanced
% for m=1:nb_staff
%     counter=0;
%     while (counter==0)
%     [temp_min,temp_min_position]=min(d_staff_bed(m,:));
%     validation=temp_min>d_staff_bed(:,temp_min_position);
%     validation=double(validation);
%        if (sum(validation)>0)
%           d_staff_bed(m,temp_min_position)=inf;
%        else
%           final_position(m)=temp_min_position;
%           counter=counter+1;
%        end
%        saturation=d_staff_bed(m,:)==inf;
%        if (sum(saturation)==nb_beds)
%            counter=counter+1;
%            final_position(m)=0;
%     end
%     end
% end
```

```
%
% for m=1:nb_staff
%     if (d_staff_bed(m,1)~=inf)
%         d_staff_bed_backup(m,:)=inf;
%         d_staff_bed_backup(:,final_position(m))=inf;
%     end
% end

 counter=0;
 while (counter==0)
     kernel=min(min(d_staff_bed_backup));
     [staff bed]=find(d_staff_bed_backup==kernel);
     final_position(staff)=bed;
     d_staff_bed_backup(staff,:)=inf;
     d_staff_bed_backup(:,bed)=inf;
     if (min(d_staff_bed_backup)==inf)
         counter=counter+1;
     end
 end

residual=find(final_position==0);
rest_bed_coords=[];
residual_staff=[];
temp1=find(final_position==0);
if (temp1>0)
    agent_staff(:,temp1)=[];
    final_position(temp1)=[];

end

nb_staff=size(agent_staff,2);
for m=1:nb_staff
    init_bed_coords(:,m)=bed_coords(:,final_position(m));
end

if (residual>0)
residual_staff=agent.staff(:,residual);
residual_staff(6,:)=5;
end

bed_coords(:,final_position')=[];
rest_bed_coords=bed_coords;
```

H sort_bed | sort_bed.m

```
function [bed_coords1]=sort_bed(bed_coords,exit_coords)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

a=size(bed_coords,2);
for m=1:a
    distance(m)=sqrt((exit_coords(1,1)-bed_coords(1,m)).^2+(exit_coords(2,1)-bed_coords(2,m))^2);
end
%distance=distance';
bed=[bed_coords;distance];
 bed(:,(a+1):(a+100))=-1000;

 for m=1:a
    for n=1:a
      if (bed(3,m+n)>bed(3,m))
          q=bed(:,m+n);
          p=bed(:,m);
          bed(:,m+n)=p;
          bed(:,m)=q;
     end
    end
 end

 bed(:,(a+1):(a+100))=[];
 bed(3,:)=[];
 bed_coords1=bed;
```

I r_alpha_beta | r_alpha_beta.m

```
function [rx_alpha_beta,ry_alpha_beta] = r_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Vector distance between agent beta and alpha (going from beta to alpha)
% INPUT:
%     rx_alpha: position x of agent alpha
%     ry_alpha: position y of agent alpha
%     rx_beta: position x of agent beta
%     ry_beta: position y of agent beta
% OUTPUT:
%     rx_alpha_beta: distance between beta and alpha in direction x
%     ry_alpha_beta: distance between beta and alpha in direction y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

rx_alpha_beta = rx_alpha - rx_beta;
ry_alpha_beta = ry_alpha - ry_beta;

end
```

J n_alpha_beta | n_alpha_beta.m

```
function [nx_alpha_beta,ny_alpha_beta] = n_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Unity directional vector from agent beta to alpha
% INPUT:
%     rx_alpha: position x of agent alpha
%     ry_alpha: position y of agent alpha
%     rx_beta: position x of agent beta
%     ry_beta: position y of agent beta
% OUTPUT:
%     nx_alpha_beta: x component of unity directional vector
%     ny_alpha_beta: y component of unity directional vector
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[rx_alpha_beta,ry_alpha_beta] = r_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta); % Directional vector from beta to alpha
d_alpha_beta = distance_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta); % Distance between alpha and beta
nx_alpha_beta = rx_alpha_beta./d_alpha_beta; % Normalization of the directional vector
ny_alpha_beta = ry_alpha_beta./d_alpha_beta; % Normalization of the directional vector

end
```

K phi_alpha_beta | phi_alpha_beta.m

```
function phi_alpha_beta = phi_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta,ex_alpha,ey_alpha)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Angle between direction of motion e and direction of force -n
% INPUT:
%     rx_alpha: position x of agent alpha
%     ry_alpha: position y of agent alpha
%     rx_beta: position x of agent beta
%     ry_beta: position y of agent beta
%     ex_alpha: x component of direction of motion of agent alpha
%     ey_alpha: y component of direction of motion of agent alpha
% OUTPUT:
%     phi_alpha_beta: norm of the vector r_alpha_beta:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[nx_alpha_beta,ny_alpha_beta] = n_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta);% Unity directional vector from beta to alpha

%phi_alpha_beta = acosd(dot(-[nx_alpha_beta,ny_alpha_beta],[ex_alpha,ey_alpha]));
phi_alpha_beta = acosd(-(nx_alpha_beta.*ex_alpha+ny_alpha_beta.*ey_alpha));

end
```

L distance_alpha_beta | distance_alpha_beta.m

```
function d_alpha_beta = distance_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```matlab
% Distance between agent beta and alpha
% INPUT:
%     rx_alpha: position x of agent alpha
%     ry_alpha: position y of agent alpha
%     rx_beta: position x of agent beta
%     ry_beta: position y of agent beta
% OUTPUT:
%     d_alpha_beta: norm of the vector r_alpha_beta:
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[rx_alpha_beta,ry_alpha_beta] = r_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta);

d_alpha_beta = sqrt(rx_alpha_beta.^2+ry_alpha_beta.^2);


end
```

M force_social | force_social.m

```matlab
function [fsx,fsy] = force_social(rx_alpha,ry_alpha,rx_beta,ry_beta,ex_alpha,ey_alpha,A_social,B_social,lambda,l_alpha,l_beta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Social force between agent alpha and beta
% INPUT:
%     rx_alpha: position x of agent alpha
%     ry_alpha: position y of agent alpha
%     rx_beta: position x of agent beta
%     ry_beta: position y of agent beta
%     ex_alpha: x component of direction of motion of agent alpha
%     ey_alpha: y component of direction of motion of agent alpha
%     A_social: interaction strength of social force
%     B_social: ragne of this repulsive force
%     lambda: parameter of the anisotropy of the interaction force
%     l_alpha: radii of agent alpha
%     l_beta: radii of agent beta
% OUTPUT:
%     fsx: social force in direction x
%     fsy: social force in direction y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

l_alpha_beta = l_alpha + l_beta; % Total size of agent alpha and beta (sum of their raiis)

d_alpha_beta = distance_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta); % Distance between alpha and beta

[nx_alpha_beta,ny_alpha_beta] = n_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta); % Unity directional vector from beta to alpha

phi = phi_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta,ex_alpha,ey_alpha); % Angle between direction of motion e and direction of force -n

fs = A_social.*(lambda+(1-lambda).*0.5.*(1+cosd(phi))).*exp((l_alpha_beta-d_alpha_beta)./B_social); % Magnitude of the social force
fsx = fs.*nx_alpha_beta; % x component of social force
fsy = fs.*ny_alpha_beta; % y component of social force

end
```

N force_physical | force_physical.m

```matlab
function [fpx,fpy] = force_physical(rx_alpha,ry_alpha,rx_beta,ry_beta,A_physical,B_physical,l_alpha,l_beta)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Physical force between agent alpha and beta
% INPUT:
%     rx_alpha: position x of agent alpha
%     ry_alpha: position y of agent alpha
%     rx_beta: position x of agent beta
%     ry_beta: position y of agent beta
%     A_physical: interaction strength of physical force
%     B_physical: ragne of this repulsive force
%     l_alpha: radii of agent alpha
%     l_beta: radii of agent beta
% OUTPUT:
%     fpx: physical force in direction x
%     fpy: physical force in direction y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

l_alpha_beta = l_alpha + l_beta; % Total size of agent alpha and beta (sum of their raiis)
```

```
d_alpha_beta = distance_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta); % Distance between alpha and beta

[nx_alpha_beta,ny_alpha_beta] = n_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta); % Unity directional vector from beta to alpha

fp = A_physical.*exp((l_alpha_beta-d_alpha_beta)./B_physical); % Magnitude of the physical force
fpx = fp.*nx_alpha_beta; % x component of physical force
fpy = fp.*ny_alpha_beta; % y component of physical force

end
```

O force_tot_agent_interaction | force_tot_agent_interaction.m

```
function [ftaix,ftaiy] = force_tot_agent_interaction(rx_alpha,ry_alpha,rx_beta,ry_beta,ex_alpha,ey_alpha,A_physical,B_physical,A_social,B_social,
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Total of agents interaction forces between agent alpha and beta
% INPUT:
%     rx_alpha: position x of agent alpha
%     ry_alpha: position y of agent alpha
%     rx_beta: position x of agent beta
%     ry_beta: position y of agent beta
%     ex_alpha: x component of direction of motion of agent alpha
%     ey_alpha: y component of direction of motion of agent alpha
%     A_physical: interaction strength of physical force
%     B_physical: ragne of this repulsive force
%     A_social: interaction strength of social force
%     B_social: ragne of this repulsive force
%     lambda: parameter of the anisotropy of the interaction force
%     l_alpha: radii of agent alpha
%     l_beta: radii of agent beta
% OUTPUT:
%     ftaix: total agent interaction force in direction x
%     ftaiy: total agent interaction force in direction y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[fpx,fpy] = force_physical(rx_alpha,ry_alpha,rx_beta,ry_beta,A_physical,B_physical,l_alpha,l_beta); % Physical agent interaction force between ag
[fsx,fsy] = force_social(rx_alpha,ry_alpha,rx_beta,ry_beta,ex_alpha,ey_alpha,A_social,B_social,lambda,l_alpha,l_beta); % Social agent interaction

ftaix = fpx + fsx;
ftaiy = fpy + fsy;
end
```

P force_tot_agent_interaction_elliptic | force_tot_agent_interaction_ellip

```
function [ftaiex,ftaiey] = force_tot_agent_interaction_elliptic(rx_alpha,ry_alpha,rx_beta,ry_beta,ex_alpha,ey_alpha,ex_beta,ey_beta,V0_alpha_beta
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Total of agents interaction forces between agent alpha and beta by using
% the elliptic potential
% INPUT:
%     rx_alpha: position x of agent alpha
%     ry_alpha: position y of agent alpha
%     rx_beta: position x of agent beta
%     ry_beta: position y of agent beta
%     ex_beta: x component of direction of motion of agent beta
%     ey_beta: y component of direction of motion of agent beta
%     V0_alpha_beta: magnitude of the potential
%     dt: time step
%     vx_beta: velocity of agent beta in direction x
%     vy_beta: velocity of agent beta in direction y
%     sigma: range of interaction force
%     phi_view: half of the effective angle of sight
%     view_weight: weight of the force if beta out of view
% OUTPUT:
%     ftaiex: total agent interaction force in direction x
%     ftaiey: total agent interaction force in direction y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

v_beta = sqrt(vx_beta.^2+vy_beta.^2);

b = 0.5.*sqrt((distance_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta)+distance_alpha_beta(rx_alpha,ry_alpha,rx_beta+v_beta*dt.*ex_beta,ry_beta+v_

V_alpha_beta = V0_alpha_beta.*exp(-b/sigma); % Interaction potential
```

```
[nx_alpha_beta,ny_alpha_beta] = n_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta); % Unity directional vector from beta to alpha

phi = phi_alpha_beta(rx_alpha,ry_alpha,rx_beta,ry_beta,ex_alpha,ey_alpha);

if (phi < phi_view) w = 1; % Is beta in the view field of alpha or not?
else w = view_weight;
end


ftaiex = w.*V_alpha_beta.*nx_alpha_beta;
ftaiey = w.*V_alpha_beta.*ny_alpha_beta;

end
```

Q force_on_alpha | `force_on_alpha.m`

```
function [fx_tot_alpha,fy_tot_alpha] = force_on_alpha(agent,alpha,e_x,e_y)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% total force on agent alpha
% input:
%   agent: matrix of all the agent
%   alpha: agent on which we want to calculate the forces
%   e_x: x component of direction of the gradient
%   e_y: y component of direction of the gradient
% output:
%     fx_tot: total force in direction x
%     fy_tot: total force in direction y
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% constants

% to be tuned:
if (agent(6,alpha)==0) % this patient is free to move
    influence_area = 3; % [m] area of influence for the interaction force between the agents
    a_physical = 3;
    b_physical = .3;
    a_social = 6;
    b_social = .6;
    lambda = 0.3;
    l_alpha = 0.4*2;
    l_beta = 0.3*2;
    v0_alpha_beta = 1;
    dt = 0.1;
    sigma = 0.3;
    phi_view = 30;
    view_weight = 0.5;
elseif (agent(6,alpha)==1||agent(6,alpha)==3||agent(6,alpha)==4||agent(6,alpha)==5)
    % this staff member is going to a bed or to a exit without bed (still free to move without bed)
    influence_area = 3; % [m] area of influence for the interaction force between the agents
    a_physical = 2;
    b_physical = 0.2;
    a_social = 5;
    b_social = 0.3;
    lambda = 0.3;
    l_alpha = 0.3*2;
    l_beta = 0.3*2;
    v0_alpha_beta = 2.1;
    dt = 0.1;
    sigma = 0.3;
    phi_view = 30;
    view_weight = 0.5;
elseif (agent(6,alpha)==2) % this staff memeber is going to the exit with a bed
    influence_area = 6; % [m] area of influence for the interaction force between the agents
    a_physical = 5;
    b_physical = 0.5;
    a_social = 5;
    b_social = 0.7;
    lambda = 0.3;
    l_alpha = 1*2;
    l_beta = 0.3*2;
    v0_alpha_beta = 2.1;
    dt = 0.1;
    sigma = 0.3;
    phi_view = 30;
```

```
    view_weight = 0.5;
end
%% initialization

% agenent alpha infos (rx,ry,vx,vy,v0)
agent_alpha = agent(:,alpha);

% distance between alpha and all the other agents (vector 1xnagent)
dist_alpha_beta = distance_alpha_beta(agent_alpha(1),agent_alpha(2),agent(1,:),agent(2,:));

% influent agents bool(into the influence region)
agent_influent_bool = dist_alpha_beta < influence_area;

% agent alpha does't influence himself
agent_influent_bool(alpha) = 0;
agent_influent_bool(dist_alpha_beta==0) = 0;

% influent agents infos
agent_influent = agent(:,agent_influent_bool);

% desired direction following gradient
ex_alpha=e_x(round(agent_alpha(2)),round(agent_alpha(1)));
ey_alpha=e_y(round(agent_alpha(2)),round(agent_alpha(1)));
%ex_agent_influent=e_x(agent_influent(1,:),agent_influent(2,:)); % only needed for
%elliptic potential
%ey_agent_influent=e_y(agent_influent(1,:),agent_influent(2,:));

%% destination force
[fx_dest,fy_dest]=destination(ex_alpha,ey_alpha,agent_alpha(3),agent_alpha(4));

%% boundary force
% not added here but in the main directly (is a static force component)

%% agent interaction forces
[ftaix,ftaiy] = force_tot_agent_interaction(agent_alpha(1),agent_alpha(2),...
    agent_influent(1,:),agent_influent(2,:),ex_alpha,ey_alpha,a_physical,...
    b_physical,a_social,b_social,lambda,l_alpha,l_beta);
%[ftaiex,ftaiey] = force_tot_agent_interaction_elliptic(agent_alpha(1),...
%agent_alpha(2),agent_influent(1,:),agent_influent(2,:),ex_alpha,ey_alpha,...
%ex_agent_influent,ey_agent_influent,v0_alpha_beta,dt,agent_influent(3,:),agent_influent(4,:),sigma,phi_view,view_weight);

ftaix_on_alpha = sum(ftaix);
ftaiy_on_alpha = sum(ftaiy);

%ftaiex_on_alpha = sum(ftaiex);
%ftaiey_on_alpha = sum(ftaiey);

%% total forces on agent alpha (without boundary forces)
fx_tot_alpha = fx_dest + ftaix_on_alpha;
fy_tot_alpha = fy_dest + ftaiy_on_alpha;

%fex_tot_alpha = fx_dest + ftaiex_on_alpha;
%fey_tot_alpha = fy_dest + ftaiey_on_alpha;

end
```

R in_wall | in_wall.m

```
function [x_coord y_coord] = in_wall(x_coord,y_coord,x_vel,y_vel,wall_map)
% function which verifies that the position of an agent is not in the wall
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% INPUT:
%   x_coord: x coordinate to be tested
%   y_coord: y coordinate to be tested
%   x_vel: velocity in direction x
%   y_vel: velocity in direction y
%   wall_map: map of the situation (0==wall, 1==empty space)
% OUTPUT:
%   x_coord: corrected x coordinate
%   y_coord: corrected y coordinate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[y_size_map x_size_map] = size(wall_map);

x_coord_new = x_coord;
y_coord_new = y_coord;
```

```

if (x_vel>0)
    for i = 1:round(x_coord)-1
        if(wall_map(round(y_coord),round(x_coord)-i)==1)
            x_coord_new = round(x_coord)-i;
            break;
        end
    end
end

if (x_vel<0)
    for i = 1:(x_size_map-round(x_coord)-1)
        if(wall_map(round(y_coord),round(x_coord)+i)==1)
            x_coord_new = round(x_coord)+i;
            break;
        end
    end
end

if (x_vel==0)
    x_coord_new = round(x_coord);
end

if (y_vel>0)
    for j = 1:round(y_coord)-1
        if(wall_map(round(y_coord)-j,round(x_coord))==1)
            y_coord_new = round(y_coord)-j;
            break;
        end
    end
end

if (y_vel<0)
    for j = 1:(y_size_map-round(y_coord)-1)
        if(wall_map(round(y_coord)+j,round(x_coord))==1)
            y_coord_new = round(y_coord)+j;
            break;
        end
    end
end


if (y_vel==0)
    y_coord_new = round(y_coord);
end

if(abs(round(x_coord)-x_coord_new)~=0)
    x_coord = x_coord_new;
    y_coord = round(y_coord);
elseif(abs(round(y_coord)-y_coord_new)~=0)
    x_coord = round(x_coord);
    y_coord = y_coord_new;
elseif((abs(round(x_coord)-x_coord_new)~=0)&&(abs(round(y_coord)-y_coord_new)~=0))
    if(abs(round(x_coord)-x_coord_new)<abs(round(y_coord)-y_coord_new))
        x_coord = x_coord_new;
        y_coord = round(y_coord);
    else
        x_coord = round(x_coord);
        y_coord = y_coord_new;
    end
end
end
```

S evacuation_main | evacuation_main.m

```
%% HOSPITAL EVACUATION MATLAB SIMULATION

close all;
clc;

%% INITIALIZATION
% Read the image and store informations
%[agent, raw_map, wall_map, exit_coords, bed_coords_initial] = getfile_rand_staff(num_staff,num_patients,num_beds, image_name);
[agent, raw_map, wall_map, exit_coords, bed_coords_initial] = getfile();


```

```
% Number of staffs
nb_staff = size(agent.staff,2);
% Number of patients (WALKING PATIENTS)
nb_patient = size(agent.patient,2);
% Number of people (agents)
nb_agent = nb_staff + nb_patient;
% Number of beds
nb_bed = size(bed_coords_initial,2);
% Number of exits
nb_exit = size(exit_coords,2);
% Number of total patients (walking and bedded)
nb_patient_tot = nb_patient+nb_bed;

% Look for the coordinates of the walls (just needed for visualization)
[wall_coord(:,1) wall_coord(:,2)]=find(wall_map==0);

% Compute the gradient map for the exit
[exit_e_x exit_e_y]=gradientmap(wall_map,exit_coords);

% Reorder beds so that the first one is the farthest away from the exit
[bed_coords]=sort_bed(bed_coords_initial,exit_coords);

% Attribuate bed to each staff member (closest bed to each staff) and create coordinates of bed which
% are not taken yet by a staff member
[staff_ordered_to_bed,residual_staff,init_bed_coords,bed_free_coords]=attribuate_bed(agent,bed_coords);
bed_coords=[init_bed_coords,bed_free_coords];
control1=nb_bed-nb_staff;

if (control1>size(bed_free_coords,2))
    control2=control1-size(bed_free_coords,2)
    bed_free_coords(:,1:control2)=[];
end

control1=size(bed_free_coords,2)-nb_staff;
if (control1 > 0)
    for i=1:control1
        bed_e_x_attribuated(:,:,nb_staff+i)=0;
        bed_e_y_attribuated(:,:,nb_staff+i)=0;
    end
end

% Compute the gradient map for the different beds
bed_e_x = zeros(size(raw_map,1),size(raw_map,2),nb_bed);
bed_e_y = zeros(size(raw_map,1),size(raw_map,2),nb_bed);
for i=1:size(init_bed_coords,2)
    [bed_e_x_temp bed_e_y_temp]=gradientmap(wall_map,init_bed_coords(:,i));
    bed_e_x(:,:,i)=bed_e_x_temp;
    bed_e_y(:,:,i)=bed_e_y_temp;
end

% Create a matrix containing both staff (ordered) and patients
agents=horzcat(staff_ordered_to_bed,residual_staff,agent.patient);
%staff_filter=1:size(temp,2);

% Initialization for the plotting
firstplot=1;

% The boudnary force on every agent is added after because it is not
% dependent on many agent, it is just dependent on the location of the
% agent
[fx_bound fy_bound]=boundary(wall_map);
Fx=zeros(nb_agent,1);
Fy=zeros(nb_agent,1);
for i=size(staff_ordered_to_bed,2)+1:nb_staff
    if (agents(6,i)==5) % is going to the exit
        [Fx(i) Fy(i)]=force_on_alpha(agents,i,exit_e_x,exit_e_y);
    end
    Fx(i) = Fx(i) + fx_bound(agents(2,i),agents(1,i));
    Fy(i) = Fy(i) + fy_bound(agents(2,i),agents(1,i));
end

for i=1:size(staff_ordered_to_bed,2)
    if (agents(6,i)==1) % is going to his assigned bed (ONLY FOR THE BEGINNING, THEN IT IS CHANGED)
        [Fx(i) Fy(i)]=force_on_alpha(agents,i,bed_e_x(:,:,i),bed_e_y(:,:,i));
    end
    Fx(i) = Fx(i) + fx_bound(agents(2,i),agents(1,i));
    Fy(i) = Fy(i) + fy_bound(agents(2,i),agents(1,i));
end
```

```
for i=nb_staff+1:nb_agent
    if (agents(6,i)==0) % is going to the exit
        [Fx(i) Fy(i)]=force_on_alpha(agents,i,exit_e_x,exit_e_y);
    end
    Fx(i) = Fx(i) + fx_bound(agents(2,i),agents(1,i));
    Fy(i) = Fy(i) + fy_bound(agents(2,i),agents(1,i));
end


% Time iterations
tf = 20000;
% Time step
Dt=0.1;

% Storage of time position of all the agents and attribuated bed for each staff (x coord, y coord, x bed coord, y bed coord; ID; time)
agents_info=zeros(4,nb_agent,tf);
agents_info(1,:,1)=agents(1,:); % x coord
agents_info(2,:,1)=agents(2,:); % y coord

% Vector containing the number of staff in the building in time
nb_staff_curr = zeros(tf,1);
nb_staff_curr(1) = nb_staff;

% Vector containing the number of walking patient in the building in time
nb_patient_curr = zeros(tf,1);
nb_patient_curr(1) = nb_patient;

% Vector containing the number of bed patient in the building in time
nb_bed_patient_curr = zeros(tf,1);
nb_bed_patient_curr(1) = nb_bed;

% Vector containing the number of people in the building in time
nb_agent_curr = zeros(tf,1);
nb_agent_curr(1) = nb_agent;

% Vector containing the number of bed in the building in time
nb_bed_curr = zeros(tf,1);
nb_bed_curr(1) = nb_bed;

% Vector containing the number of staff in the building in time
nb_patient_tot_curr = zeros(tf,1);
nb_patient_tot_curr(1) = nb_patient_tot;

% State variable to see if the staff already rescued the first time when
% they reach the exit
not_first_time=zeros(1,nb_staff);


%% TIME STEPPING
% velocity time steped with simple Euler
% Once velocity calculated Displacement using S= V.dt (acceleration not
% included as for small times steps it has virtually no influence

mass=1;

counter_display=1; % Conter for the display

time_history = zeros(tf,1);

stop(1:nb_staff_curr)=0;
nb_total=nb_staff+nb_patient;

% Iterating over time
for time=1:tf
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Initialization

    nb_staff_curr(time) = size(staff_ordered_to_bed,2);
    nb_patient_curr(time) = size(agents,2)-nb_staff_curr(time);
    nb_agent_curr(time) = size(agents,2);
    nb_bed_curr(time) = size(bed_free_coords,2);
    nb_bed_patient_curr(time) = size(bed_free_coords,2)+nb_staff_curr(time);
    nb_patient_tot_curr(time) = nb_patient_curr(time)+nb_bed_patient_curr(time);

    time_history(time)=time;

```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iterating over each agent to calculate their new position by knowing
% the force acting on them at this time
for i=1:size(agents,2)

    % Look for the old velocity of the agent
    if(time==1)
        oldUx=0;
        oldUy=0;
    else
        oldUx=agents(3,i);
        oldUy=agents(4,i);
    end

    % Update the velocity of each agent (old velocity plus
    % acceleration*Dt)
    if (isfinite(Fx(i)) && isfinite(Fy(i)))
        agents(3,i)=Fx(i)*Dt/mass+oldUx;
        agents(4,i)=Fy(i)*Dt/mass+oldUy;
    end

    % Calculate the norm of this new velocity
    vel_norm=sqrt(agents(3,i)^2+agents(4,i)^2);

    % Look that the agent are not going faster that what the are able
    % to
    if (vel_norm>agents(5,i))
        agents(3,i)=agents(3,i)/vel_norm*agents(5,i);    %scale velocities
        agents(4,i)=agents(4,i)/vel_norm*agents(5,i);
    end

    % Update finally their position
    new_x_position = agents(1,i)+(agents(3,i)*Dt);
    new_y_position = agents(2,i)+(agents(4,i)*Dt);

    if(wall_map(round(new_y_position),round(new_x_position))==1)
        agents(1,i)=new_x_position;
        agents(2,i)=new_y_position;
    else
        [agents(1,i) agents(2,i)] = in_wall(new_x_position,new_y_position,agents(3,i),agents(4,i),wall_map);
    end

    % Storing position in time matrix
    agents_info(1,i,time)=agents(1,i);
    agents_info(2,i,time)=agents(2,i);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Iterating over time to calculate the new forces acting on the agent
% at their new position

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FOR THE STAFF MEMBERS

distance_staff_bed = zeros(nb_bed,nb_staff_curr(time));
distance_staff_exit = zeros(nb_exit,nb_staff_curr(time));

counter=0;
%   bed_counter=0;

% Loop over all the staff
for i=1:nb_staff;

    % Look if the staff are close to a bed
    for k=1:nb_bed
        distance_staff_bed(k,i)=distance_alpha_beta(agents(1,i),agents(2,i),bed_coords(2,k),bed_coords(1,k));
        if (distance_staff_bed(k,i)<=sqrt(2))
            agents(6,i)=2; % change the state of the state from 'free' to 'with a bed'
            %                agents(5,i) = agents(5,i)*(2/3);
        end
    end

    distance_staff_exit(i)=distance_alpha_beta(agents(1,i),agents(2,i),exit_coords(2,5),exit_coords(1,5));
    % Distance between staff member and exit, exit position has
    % been presented with its midpoint, i.e. exit_coords(:,10)

```

```
        %Conditions over staff states

        if (distance_staff_exit(i)<3 && agents(6,i)==2 && size(bed_free_coords,2)==0 )
            % For a staff to change state from 2 to 3, three conditions must be satisfied
            % 1) staff is at the exit
            % 2) staff is with a bed
            % 3) there are no beds to be rescued
            agents(6,i)=3;
            bed_e_x_attribuated(:,:,i)=0;
            bed_e_y_attribuated(:,:,i)=0;
            stop(i)=0;
            % stop is a variable used for visualization purposes when staff
            % member reaches the bed
            nb_bed_patient_curr(time)=nb_bed_patient_curr(time)-1;
            nb_patient_tot_curr(time)=nb_patient_tot_curr(time)-1;
        end

        if (distance_staff_exit(i)<3 && agents(6,i)==2 && size(bed_free_coords,2)~=0 )
            % For a staff to change state from 2 to 4, three conditions must be satisfied
            % 1) staff is at the exit
            % 2) staff is with a bed
            % 3) there are still beds to be rescued
            agents(6,i)=4;
            [temp1 temp2]=gradientmap(wall_map,bed_free_coords(:,1));
            %temp1 & temp2 - temporary values for position vectors
            bed_e_x_attribuated(:,:,i)=temp1;
            bed_e_y_attribuated(:,:,i)=temp2;
            %            bed_counter=bed_counter+1;
            nb_bed_curr(time)=nb_bed_curr(time)-1; % one bed less in the building
            nb_bed_patient_curr(time)=nb_bed_patient_curr(time)-1;
            stop(i)=0;
            bed_free_coords(:,1)=[];
            % after the first free bed coordinates have been used for
            % computation of gradient map now they are being deleted
        end

        if (distance_staff_exit(i)<3 && agents(6,i)==5)
            % For a staff to change state from 5 to 3, three conditions must be satisfied
            % 1) staff is at the exit
            % 2) staff has no bed
            % 3) staff has been residual at the very beginnig of time steping
            agents(6,i)=3;
            bed_e_x_attribuated(:,:,i)=0;
            bed_e_y_attribuated(:,:,i)=0;
            bed_e_x(:,:,i)=0;
            bed_e_y(:,:,i)=0;
            stop(i)=0;
        end

        % Calculate the forces

        if (agents(6,i)==1) % staff is going to the attribued bed
            [Fx(i) Fy(i)]=force_on_alpha(agents,i,bed_e_x(:,:,i),bed_e_y(:,:,i));
            Fx(i) = Fx(i) + fx_bound(round(agents(2,i)),round(agents(1,i)));
            Fy(i) = Fy(i) + fy_bound(round(agents(2,i)),round(agents(1,i)));
        end

        if (agents(6,i)==2) % staff has a bed and is going to the exit
            [Fx(i) Fy(i)]=force_on_alpha(agents,i,exit_e_x(:,:),exit_e_y(:,:));
            Fx(i) = Fx(i) + fx_bound(round(agents(2,i)),round(agents(1,i)));
            Fy(i) = Fy(i) + fy_bound(round(agents(2,i)),round(agents(1,i)));
            stop(i)=stop(i)+1;
            if (stop(i)<90)
                Fx(i)=0;
                Fy(i)=0;
                agents(3,i) = 0;
                agents(4,i) = 0;
            end
        end

        if (agents(6,i)==4) % staff is going back to the attribuated bed
            [Fx(i) Fy(i)]=force_on_alpha(agents,i,bed_e_x_attribuated(:,:,i),bed_e_y_attribuated(:,:,i));
            Fx(i) = Fx(i) + fx_bound(round(agents(2,i)),round(agents(1,i)));
            Fy(i) = Fy(i) + fy_bound(round(agents(2,i)),round(agents(1,i)));
            %             agents(5,i) = agents(5,i)*(3/2);
        end

        if (agents(6,i)==5) % residual staff is going to the exit
```

```
            [Fx(i) Fy(i)]=force_on_alpha(agents,i,exit_e_x(:,:),exit_e_y(:,:));
            Fx(i) = Fx(i) + fx_bound(round(agents(2,i)),round(agents(1,i)));
            Fy(i) = Fy(i) + fy_bound(round(agents(2,i)),round(agents(1,i)));
        end
    end

    % Deleting staff members that have reached with the corresponding data
    % so that the adequate switch between indices is achieved

    for i=1:size(agents,2)
        if (agents(6,i)==3)
            counter=counter+1;
            index(counter)=i;
        end
    end

    deleted = 0;
    for i=1:counter
        agents(:,index(counter)-deleted)=[];
        bed_e_x(:,:,index(counter))=[];
        bed_e_y(:,:,index(counter))=[];
        bed_e_x_attribuated(:,:,index(counter))=[];
        bed_e_y_attribuated(:,:,index(counter))=[];
        stop(index(counter))=[];
        deleted = deleted + 1;
    end

    %Updating the number of staff
    nb_staff=nb_staff-counter;
    nb_staff_curr(time)=nb_staff_curr(time)-counter;
    nb_agent_curr(time)=nb_agent_curr(time)-counter;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % FOR THE PATIENTS


    distance_patient_exit = zeros(1,size(agents,2)-nb_staff);
    counter_patient_outside = 0;
    index_patient_outside = [];

    % Look if the patient are close to the exit
    for i= nb_staff+1:size(agents,2)
        distance_patient_exit(i)=distance_alpha_beta(agents(1,i),agents(2,i),exit_coords(2,5),exit_coords(1,5));
        if (distance_patient_exit(i)< 3)
            counter_patient_outside = counter_patient_outside+1;
            index_patient_outside(counter_patient_outside)=i; % saved patient
        end
    end

    distance_patient_exit=[];

    % Updating the patient situation
    if (counter_patient_outside~=0)
        for i=1:counter_patient_outside
            agents(:,index_patient_outside(i))=[];
            agents_info(:,i,time)=0;
            nb_patient_curr(time)=nb_patient_curr(time)-1;
            nb_patient_tot_curr(time)=nb_patient_tot_curr(time)-1;
            nb_agent_curr(time)=nb_agent_curr(time)-1;
        end
    end

    % Calculate the forces
    for i=nb_staff+1:size(agents,2)
        [Fx(i) Fy(i)]=force_on_alpha(agents,i,exit_e_x(:,:),exit_e_y(:,:));
        Fx(i) = Fx(i) + fx_bound(round(agents(2,i)),round(agents(1,i)));
        Fy(i) = Fy(i) + fy_bound(round(agents(2,i)),round(agents(1,i)));
    end

    nb_patient=nb_patient_curr(time);
    nb_total=nb_staff+nb_patient_curr(time);
    if(nb_agent_curr(time)==0)
        break;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    counter_display=counter_display+1;
    if (counter_display==5)
```

```
        if (firstplot==1)
            %   figure
            firstplot=0;
        end
        %      if(real_time_display==1)
        Display_agents_on_map(agents,nb_staff,wall_coord, bed_coords,firstplot);
        %     end
        counter_display=1;
    end
end


%% DELETE USELESS ENTRIES IN AGENT_INFO MATRIX

agents_info(:,:,time+1:tf)=[];

%% SOME STATISTICS

figure,
plot(time_history(1:time),nb_patient_tot_curr(1:time),'r');
set(gca, 'FontSize', 18);
xlabel('Time iteration');
ylabel('Number of patients inside the building');
```

T staff2bed | staff2bed.m

```
function conex_s2b =staff2bed(staff, beds)
conex_s2b=zeros(2,size(staff,2));
dist=zeros(size(beds,2) ,size(staff,2));
for i=1:size(beds,2)
    for j=1:size(staff,2)
        dist(i,j)=sqrt((staff(1,j)-beds(2,i))^2+(staff(2,j)-beds(1,i))^2);
    end
end

 for k=1:size(staff,2)
     min=1000;
     for i=1:size(dist,1)
         for j=1:size(dist,2)
             if(dist(i,j)<=min)
                 min=dist(i,j);
                 idx_staff=j;
                 idx_bed=i;
             end
         end
     end
     dist(idx_bed,:)=1000;
     dist(:,idx_staff)=1000;
     conex_s2b(1,k)=idx_staff;
     conex_s2b(2,k)=idx_bed;

 end
```

U find_free_bed | find_free_bed.m

```
function bed_free_idx = find_free_bed(connex_s2b, nb_bed)
bed_free_idx=[];
free_bed_counter=1;
for i=1:nb_bed
    same =0;
    for j=1:size(connex_s2b,2)
        if(connex_s2b(2,j)==i)
            same=1;
        end
     end
     if(same==0)
         bed_free_idx(free_bed_counter)=i;
         free_bed_counter=free_bed_counter+1;
         same=0;
     end
 end
```

V evacuation | evacuation.m

```matlab

function [agents_info bed_coords nb_patient_curr nb_bed_patient_curr nb_patient_tot_curr nb_staff_curr]=evacuation(num_agents,perc_staff, perc_be
%% HOSPITAL EVACUATION MATLAB SIMULATION

close all;
clc;

%% INITIALIZATION
num_staff=round(num_agents*(perc_staff/100));
 num_beds =round((num_agents-num_staff)*(perc_beds/100));
 num_patients=round(num_agents-num_staff-num_beds);

 % Read the image and store informations
 [agent, raw_map, wall_map, exit_coords, bed_coords] = getfile_rand_staff(num_staff,num_patients,num_beds, image_name);

 % Create a matrix containing both staff and patients
 agents=horzcat(agent.staff,agent.patient);

 % Number of staffs
 nb_staff = size(agent.staff,2);
 % Number of patients (WALKING PATIENTS)
 nb_patient = size(agent.patient,2);
 % Number of people (agents)
 nb_agent = nb_staff + nb_patient;
 % Number of beds
 nb_bed = size(bed_coords,2);
 % Number of exits
 nb_exit = size(exit_coords,2);
 % Number of total patients (walking and bedded)
 nb_patient_tot = nb_patient+nb_bed;

 % Look for the coordinates of the walls (just needed for visualization)
 [wall_coord(:,1) wall_coord(:,2)]=find(wall_map==0);

 % Compute the gradient map for the exit
 [exit_e_x exit_e_y]=gradientmap(wall_map,exit_coords);

 % Connectity array between staff and bed
 connex_s2b=staff2bed(agent.staff,bed_coords);

 % Index of bed which is still free
 bed_free_idx = find_free_bed(connex_s2b, nb_bed);

 % Array containing initial index of agents
 agent_idx = 1:nb_agent;

 % Initialization of states
 for i = 1:nb_agent
     flag_in_array=0;
     for j=1:size(connex_s2b,2)
         if(connex_s2b(1,j)==i)
             flag_in_array=1;

         else
             if(i<=nb_staff)
                 agents(6,i)=5;
             else
                 agents(6,i)=0;
             end
         end
         if(flag_in_array==1)
             agents(6,i)=1;
         end
     end
 end

 % Compute the gradient map for the different beds
 bed_e_x = zeros(size(raw_map,1),size(raw_map,2),nb_bed);
 bed_e_y = zeros(size(raw_map,1),size(raw_map,2),nb_bed);
 for i=1:nb_bed
     [bed_e_x_temp bed_e_y_temp]=gradientmap(wall_map,bed_coords(:,i));
     bed_e_x(:,:,i)=bed_e_x_temp;
     bed_e_y(:,:,i)=bed_e_y_temp;
 end

 % Initialization for the plotting
 firstplot=1;
```

```
% The boudnary force on every agent is added after because it is not
% dependent on many agent, it is just dependent on the location of the
% agent
[fx_bound fy_bound]=boundary(wall_map);
Fx=zeros(nb_agent,1);
Fy=zeros(nb_agent,1);

% Compute the direction field for the agents
e_x = zeros(size(raw_map,1),size(raw_map,2));
e_y = zeros(size(raw_map,1),size(raw_map,2));

for i=1:size(agent_idx,2);
    for j=1:size(connex_s2b,2)
        if((agent_idx(i)==connex_s2b(1,j))&&((agents(6,agent_idx(i))==1)||(agents(6,agent_idx(i))==4)))
            e_x=bed_e_x(:,:,connex_s2b(2,j));
            e_y=bed_e_y(:,:,connex_s2b(2,j));

        end
    end
    if((agents(6,agent_idx(i))==0)||(agents(6,agent_idx(i))==2)||((agents(6,agent_idx(i))==5)))
        e_x = exit_e_x;
         e_y = exit_e_y;
     end

     % Initialize the forces
     [Fx(agent_idx(i)) Fy(agent_idx(i))]=force_on_alpha(agents(:,agent_idx),i,e_x,e_y);

     Fx(agent_idx(i)) = Fx(agent_idx(i)) + fx_bound(agents(2,agent_idx(i)),agents(1,agent_idx(i)));
     Fy(agent_idx(i)) = Fy(agent_idx(i)) + fy_bound(agents(2,agent_idx(i)),agents(1,agent_idx(i)));

end


% Time iterations
tf = 20000;
% Time step
Dt=0.1;

% Storage of time position of all the agents and attribuated bed for each staff (x coord, y coord, x bed coord, y bed coord; ID; time)
agents_info=zeros(4,nb_agent,tf);
agents_info(1,:,1)=agents(1,:); % x coord
agents_info(2,:,1)=agents(2,:); % y coord

% Vector containing the number of staff in the building in time
nb_staff_curr = zeros(tf,1);
nb_staff_curr(1) = nb_staff;

% Vector containing the number of walking patient in the building in time
nb_patient_curr = zeros(tf,1);
nb_patient_curr(1) = nb_patient;

% Vector containing the number of bed patient in the building in time
nb_bed_patient_curr = zeros(tf,1);
nb_bed_patient_curr(1) = nb_bed;

% Vector containing the number of people in the building in time
nb_agent_curr = zeros(tf,1);
nb_agent_curr(1) = nb_agent;

% Vector containing the number of bed in the building in time
nb_bed_curr = zeros(tf,1);
nb_bed_curr(1) = nb_bed;

% Vector containing the number of staff in the building in time
nb_patient_tot_curr = zeros(tf,1);
nb_patient_tot_curr(1) = nb_patient_tot;


%% TIME STEPPING
% velocity time steped with simple Euler
% Once velocity calculated Displacement using S= V.dt (acceleration not
% included as for small times steps it has virtually no influence

mass=1; % ATTENTION: THIS SHOULD BE MAYBE A PROPERTY OF EVERY AGENT? OR JUST USE 1 AND CHANGE CHE DESIRED VELOCITY OF EVERY AGENT TYPE?

counter_display=1; % Counter for the display

```

```
time_history = zeros(tf,1);

stop(1:nb_staff_curr)=0;
% Iterating over time


for time=1:tf

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Initialization
    nb_staff_curr(time+1)=nb_staff_curr(time);

    nb_patient_curr(time) = size(agent_idx,2)-nb_staff_curr(time);
    nb_agent_curr(time) = size(agent_idx,2);
    nb_bed_curr(time) = size(bed_free_idx,2);
    nb_bed_patient_curr(time) = size(bed_free_idx,2)+nb_staff_curr(time);
    nb_patient_tot_curr(time) = nb_patient_curr(time)+nb_bed_patient_curr(time); % ATTENTION: THE NUMBER OF TOTAL PATIENT, AS THE NUMBER OF BED

    time_history(time)=time;

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Iterating over each agent to calculate their new position by knowing
    % the force acting on them at this time
    for i=1:size(agent_idx,2)

        % Look for the old velocity of the agent
        if(time==1)
            oldUx=0;
            oldUy=0;
        else
            oldUx=agents(3,agent_idx(i));
            oldUy=agents(4,agent_idx(i));
        end

        % Update the velocity of each agent (old velocity plus
        % acceleration*Dt)
        %if (isfinite(Fx(i)) && isfinite(Fy(i)))
        agents(3,agent_idx(i))=Fx(agent_idx(i))*Dt/mass+oldUx;
        agents(4,agent_idx(i))=Fy(agent_idx(i))*Dt/mass+oldUy;
        %end

        % Calculate the norm of this new velocity
        vel_norm=sqrt(agents(3,agent_idx(i))^2+agents(4,agent_idx(i))^2);

        % Look that the agent are not going faster that what the are able
        % to
        if (vel_norm>agents(5,agent_idx(i)))
            agents(3,agent_idx(i))=agents(3,agent_idx(i))/vel_norm*agents(5,agent_idx(i));    %scale velocities
            agents(4,agent_idx(i))=agents(4,agent_idx(i))/vel_norm*agents(5,agent_idx(i));
        end

        % Update finally their position
        new_x_position = agents(1,agent_idx(i))+(agents(3,agent_idx(i))*Dt);
        new_y_position = agents(2,agent_idx(i))+(agents(4,agent_idx(i))*Dt);


        if(wall_map(round(new_y_position),round(new_x_position))==1)
            agents(1,agent_idx(i))=new_x_position;
            agents(2,agent_idx(i))=new_y_position;
        else
            [agents(1,agent_idx(i)) agents(2,agent_idx(i))] = in_wall(new_x_position,new_y_position,agents(3,agent_idx(i)),agents(4,agent_idx(i)
        end

        % Storing position in time matrix
        agents_info(1,agent_idx(i),time)=agents(1,agent_idx(i));
        agents_info(2,agent_idx(i),time)=agents(2,agent_idx(i));
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    % Iterating over time to calculate the new forces acting on the agent
    % at their new position

    distance_staff_exit = zeros(nb_exit,nb_staff_curr(time));

    counter_staff_useless = 0;
```

```
index_staff_useless = [];
counter_patient_outside = 0;
index_patient_outside = [];

% Loop over all the agents
for i=1:size(agent_idx,2)

    if((agents(6,agent_idx(i))==1)||(agents(6,agent_idx(i))==4))
        % Look if the staff are close to a bed
        for s_conx=1:size(connex_s2b,2)
            if(agent_idx(i)==connex_s2b(1,s_conx))
                array_idx=s_conx;
            end
        end
        distance_staff_bed= distance_alpha_beta(agents(1,agent_idx(i)),agents(2,agent_idx(i)),bed_coords(2,connex_s2b(2,array_idx)),bed_coord
        if (distance_staff_bed<=sqrt(2))
            agents(6,agent_idx(i))=2; % change the state of the state from 'free' to 'with a bed'
            for s_conx=1:size(connex_s2b,2)
                if(agent_idx(i)==connex_s2b(1,s_conx))
                    array_idx=s_conx;
                end
            end
            connex_s2b(1,array_idx)=0;
            agents(5,agent_idx(i))=(2/3)*agents(5,agent_idx(i));
        end
    end

    % Look if the staff are close to the exit
    for l=1:nb_exit
        distance_staff_exit(l,agent_idx(i))=distance_alpha_beta(agents(1,agent_idx(i)),agents(2,agent_idx(i)),exit_coords(2,l),exit_coords(1,
        if (distance_staff_exit(l,agent_idx(i))<=3.5 && ((agents(6,agent_idx(i))==2) || (agents(6,agent_idx(i))==5)))
            agents(6,agent_idx(i))=3; % change the state of the state to 'at the exit'
        elseif((agents(6,agent_idx(i))==0)&&distance_staff_exit(l,agent_idx(i))<=3.5)
            counter_patient_outside = counter_patient_outside+1;
            index_patient_outside(counter_patient_outside)=i; % saved patient
            nb_patient_curr(time+1)=nb_patient_curr(time)-1;
            nb_patient_tot_curr(time+1)=nb_patient_tot_curr(time)-1;
            nb_agent_curr(time+1)=nb_agent_curr(time)-1;

        end
    end


    for i=1:size(agent_idx,2);
        for j=1:size(connex_s2b,2)
            if((agent_idx(i)==connex_s2b(1,j))&&((agents(6,agent_idx(i))==1)||(agents(6,agent_idx(i))==4)))
                e_x=bed_e_x(:,:,connex_s2b(2,j));
                e_y=bed_e_y(:,:,connex_s2b(2,j));

            end
        end
        if((agents(6,agent_idx(i))==0)||(agents(6,agent_idx(i))==2)||((agents(6,agent_idx(i))==5)))
            e_x = exit_e_x;
            e_y = exit_e_y;
        end

        [Fx(agent_idx(i)) Fy(agent_idx(i))]=force_on_alpha(agents(:,agent_idx),i,e_x,e_y);
        if ((agents(6,agent_idx(i))==2)&&(stop(agent_idx(i))<300))

            Fx(agent_idx(i))=0;
            Fy(agent_idx(i))=0;
            agents(3,agent_idx(i)) = 0;
            agents(4,agent_idx(i)) = 0;
            stop(agent_idx(i))=stop(agent_idx(i))+1;
        elseif(agents(6,agent_idx(i))==3)
            bed_free_idx = find_free_bed(connex_s2b, nb_bed);
            if(size(bed_free_idx,2)==0)
                counter_staff_useless=counter_staff_useless+1;
                index_staff_useless(counter_staff_useless)=i;
                nb_bed_patient_curr(time+1)=nb_bed_patient_curr(time)-1;
                nb_patient_tot_curr(time+1)=nb_patient_tot_curr(time)-1;
                nb_staff_curr(time+1)=nb_staff_curr(time)-1;
                nb_agent_curr(time+1)=nb_agent_curr(time)-1;
            elseif(size(bed_free_idx,2)>0)
                link_s2b= staff2bed([agents(1,agent_idx(i)) agents(2,agent_idx(i))]',bed_coords(:,bed_free_idx));
                link_s2b(1)=agent_idx(i);
                link_s2b(2)=bed_free_idx(link_s2b(2));
                connex_s2b=horzcat(link_s2b,connex_s2b);
```

```
                    if(size(bed_free_idx,2)>1)
                        bed_free_idx = find_free_bed(connex_s2b, nb_bed);
                    else
                        bed_free_idx=[];
                    end
                    agents(6,agent_idx(i))=4;
                    bed_free_idx = find_free_bed(connex_s2b, nb_bed);
                    nb_bed_patient_curr(time+1)=nb_bed_patient_curr(time)-1;
                    nb_patient_tot_curr(time+1)=nb_patient_tot_curr(time)-1;
                    stop(agent_idx(i))=0;
                    agents(5,agent_idx(i))=(3/2)*agents(5,agent_idx(i));

                end
            else
                Fx(agent_idx(i)) = Fx(agent_idx(i)) + fx_bound(ceil(agents(2,agent_idx(i))),ceil(agents(1,agent_idx(i))));
                Fy(agent_idx(i)) = Fy(agent_idx(i)) + fy_bound(ceil(agents(2,agent_idx(i))),ceil(agents(1,agent_idx(i))));

            end
        end

    end

    % Updating of agents situation


    delete_idx=horzcat(index_staff_useless,index_patient_outside);

    agent_idx(delete_idx)=[];
    delete_idx=[];

    if(size(agent_idx,2)==0)
        break;
    end

    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
    counter_display=counter_display+1;
    if (counter_display==50)

        if (firstplot==1)
            %   figure
            firstplot=0;
        end
        if(real_time_display==1)
            Display_agents_on_map(agents(:,agent_idx),nb_staff_curr(time),wall_coord, bed_coords,firstplot);
        end
        counter_display=1;
    end
end

%% DELETE USELESS ENTRIES IN AGENT_INFO MATRIX

agents_info(:,:,time+2:tf)=[];
nb_bed_patient_curr(time+1:tf)=[];
nb_patient_curr(time+1:tf)=[];
nb_patient_tot_curr(time+1:tf)=[];
nb_staff_curr(time+1:tf)=[];
%% SOME STATISTICS

%figure,
%plot(1:time+1,nb_patient_curr(1:time+1),'r--',1:time+1,nb_bed_patient_curr(1:time+1),'r-.',1:time+1,nb_patient_tot_curr(1:time+1),'r-',1:time+1,
%xlabel('Time iteration');
%ylabel('Unity of agent type inside the building');
%legend('Walking patient','Bed patient','Patient (bed+walking)','Staff');

test=1;
```

W Display_agents_on_map | Display_agents_on_map.m

```
function [] = Display_agents_on_map( agent,num_staff,  wall_coord, bed, firstplot )
% Function which recieves a structure with agent groups, a image with the
% walls and the sinks, and displays them in an image.
cla
axis equal
hold on
h =plot(wall_coord(:,2),wall_coord(:,1),'.k');
```

```
h = scatter(bed(2,:),bed(1,:),9,[0.25 1 0.5], 'filled');
%h=plot(bed(2,:),bed(1,:),'.y');
 if(size(agent,2)>0)
     h= plot(agent(1,1:num_staff), agent(2,1:num_staff),'.b');
 end
 if(num_staff<size(agent,2))
     h= plot(agent(1,num_staff+1:size(agent,2)), agent(2,num_staff+1:size(agent,2)),'.r');
 end
 % scatter(wall_coord(:,2),wall_coord(:,1),4,[0 0 0], 'filled');
 % scatter(agent(1,1:num_staff), agent(2,1:num_staff),5,[1 1 0],'filled');
 % scatter(agent(1,num_staff:size(agent,2)), agent(2,num_staff:size(agent,2)),5,[0.5 0 1],'filled');
 % scatter(bed(2,:),bed(1,:),9,[0.25 1 0.5], 'filled');
 %hold off
 %hfig = imgcf;
 %refreshdata(h)

 %drawnow;
 pause(.1)


 end

```

X scene_2d | scene_2d.m

```
function [] = video_2d( history, wall_coord, bed, t_total, fps, num_staff, num_bed_pat,file_name)
% Function for creating avi files from simulation history files.
%hisory is an array containing the time history of agent possitions and
%speeds.
%wall_coord is a vector of wall coordinates
%bed is a vector of bed coordinates
%t_total is the length of the video
vidObj = VideoWriter(file_name);
open(vidObj); %%create a video object
 frame_total=t_total*fps;
 step_size=round(size(history,3)/frame_total); %% Calculate the number of time steps skipped between each frame
 extreme=max(wall_coord);
  loc_free_staff= extreme(1)+5;
  loc_free_patients = extreme(1)+10;
  loc_free_beds = extreme(1)+15;
  total_beds=max(num_bed_pat);
 fig=figure;
 set(fig, 'Position', [0 0 640 480] ); %fix figure size
 axis equal %make scale uniform in image
 for frame=1:step_size:size(history,3)
     staff=[];
     patient=[];
     staff_out=[];
     pat_out=[];
     counter=1;
     out_counter=1;
     for i=1:num_staff
         if((history(1,i,frame)==0)&&(history(2,i,frame)==0))
             staff_out(2,out_counter)=loc_free_staff;
             staff_out(1,out_counter)=40+2*out_counter;
             out_counter=out_counter+1;
         else
             staff(1,counter)=history(1,i,frame);
             staff(2,counter)=history(2,i,frame);
             counter=counter+1;
         end
     end
     if(size(staff_out,2)>0)
     staff = horzcat(staff,staff_out);
     end
     counter=1;
     out_counter=1;
     for i=num_staff+1:size(history,2)
         if((history(1,i,frame)==0)&&(history(2,i,frame)==0))
             pat_out(2,out_counter)=loc_free_patients;
             pat_out(1,out_counter)=40+2*out_counter;
             out_counter=out_counter+1;
         else
             patient(1,counter)=history(1,i,frame);
             patient(2,counter)=history(2,i,frame);
             counter=counter+1;
```

```
        end
    end
    if(size(pat_out,2)>0)
    patient=horzcat(patient,pat_out);
    end
    hold on

    if(num_bed_pat(frame)<total_beds)

        for i=1:(total_beds-num_bed_pat(frame))
    h=scatter(40+2*i,loc_free_beds,9,[0 1 0],'filled');
        end
    end


    h = scatter(wall_coord(:,2),wall_coord(:,1),1,[0 0 0], 'filled');
    % h =plot(wall_coord(:,2),wall_coord(:,1),'.k');
    h = scatter(bed(2,:),bed(1,:),9,[0 1 0], 'filled');
    %h=plot(bed(2,:),bed(1,:),'.y');
    if(size(staff,2)>0)
        h= scatter(staff(1,:), staff(2,:),10,[0 0 1],'filled');
        %h= plot(agent(1,1:num_staff), agent(2,1:num_staff),'.b');
    end
    if(size(patient,2)>0)
        h= scatter(patient(1,:), patient(2,:),10,[1 0 0],'filled');
        %h= plot(patient(1,:), patient(2,:),'.r');
    end


    %%3d crap
    currFrame = getframe;
    writeVideo(vidObj,currFrame);
    cla;
end


close(vidObj);


%scatter(wall_coord(:,2),wall_coord(:,1),5,[0 0 0], 'filled');
%scatter(agent(1,:), agent(2,:),5,[1 0.5 0],'filled');
%scatter(bed(2,:),bed(1,:),5,[0.25 1 0.5], 'filled');

%refreshdata
% drawnow
```

Y scene_3d | scene_3d.m

```
function [] = video_3d( history, wall_coord, bed, t_total, fps, num_staff)
% Function which recieves a structure with agent groups, a image with the
% walls and the sinks, and displays them in an image.
vidObj = VideoWriter('example1.avi');
open(vidObj);
frame_total=t_total*fps;
step_size=round(size(history,3)/frame_total);
last_frame=size(history,3)/fps;

for frame=1:step_size:size(history,3)

    for i=1:num_staff
        staff(1,i)=history(1,i,frame);
        staff(2,i)=history(2,i,frame);
    end
    for i=num_staff+1:size(history,2)
        patient(1,i)=history(1,i,frame);
        patient(2,i)=history(2,i,frame);
    end
    screen_size = get(0, 'ScreenSize');

   fig=figure
     set(fig, 'Position', [0 0 640 480] );

 axis([0 max(wall_coord(:,2)) 0 max(wall_coord(:,1)) 0 120 0 1])
sizew=[1 1 3];
sizea=[1 1 2.5];
sizel=[0.75 0.75 2.5];
```

```
sizeb=[2 3 2];
sizep=[2 0.5 0.25];
sizeh=[0.5 0.5 0.75];

%plot walls
for i=1:size(wall_coord,1)
   plotcube( sizew, [wall_coord(i,2) wall_coord(i,1) 0], 1,[0 0 0]);
end
%plot staff
for i=1:size(staff,2)
    if((staff(1,i)~=0)&&(staff(2,i)~=0))
   plotcube( sizea, [staff(1,i) staff(2,i) sizel(3)], 1,[1 1 1]);%torso
   plotcube( sizel, [staff(1,i)+0.125 staff(2,i)+0.125 0], 1,[0 1 0]);%legs
   plotcube( sizeh, [staff(1,i)+0.25 staff(2,i)+0.25 sizea(3)+sizel(3)], 1,[1 0.5 1]);%head
    end
end
%plot patients
for i=1:size(patient,2)
    if((patient(1,i)~=0)&&(patient(2,i)~=0))
   plotcube( sizea, [patient(1,i) patient(2,i) sizel(3)], 1,[0.5 1 0]);
   plotcube( sizel, [patient(1,i)+0.125 patient(2,i)+0.125 0], 1,[0 1 0]);
   plotcube( sizeh, [patient(1,i)+0.25 patient(2,i)+0.25 sizea(3)+sizel(3)], 1,[1 0.5 1]);
    end
end
%plot beds
for i=1:size(bed,2)
   plotcube( sizeb, [bed(2,i) bed(1,i) 0], 1, [0 0.5 1]);
   plotcube( sizep, [bed(2,i) bed(1,i) sizeb(3)], 1, [1 1 1]);
end
 axis([0 max(wall_coord(:,2)) 0 max(wall_coord(:,1)) 0 120 0 1])
    currFrame = getframe;
   writeVideo(vidObj,currFrame);
   close all;
end


close(vidObj);


%scatter(wall_coord(:,2),wall_coord(:,1),5,[0 0 0], ’filled’);
%scatter(agent(1,:), agent(2,:),5,[1 0.5 0],’filled’);
%scatter(bed(2,:),bed(1,:),5,[0.25 1 0.5], ’filled’);

%refreshdata
% drawnow
```

Z video_2d | video_2d.m

```
function [] = video_2d( history, wall_coord, bed, t_total, fps, num_staff, num_bed_pat,file_name)
% Function for creating avi files from simulation history files.
%hisory is an array containing the time history of agent possitions and
%speeds.
%wall_coord is a vector of wall coordinates
%bed is a vector of bed coordinates
%t_total is the length of the video
vidObj = VideoWriter(file_name);
open(vidObj); %%create a video object
frame_total=t_total*fps;
step_size=round(size(history,3)/frame_total); %% Calculate the number of time steps skipped between each frame
extreme=max(wall_coord);
 loc_free_staff= extreme(1)+5;
 loc_free_patients = extreme(1)+10;
 loc_free_beds = extreme(1)+15;
 total_beds=max(num_bed_pat);
fig=figure;
set(fig, ’Position’, [0 0 640 480] ); %fix figure size
axis equal %make scale uniform in image
for frame=1:step_size:size(history,3)
    staff=[];
    patient=[];
    staff_out=[];
    pat_out=[];
    counter=1;
    out_counter=1;
    for i=1:num_staff
        if((history(1,i,frame)==0)&&(history(2,i,frame)==0))
```

```
            staff_out(2,out_counter)=loc_free_staff;
            staff_out(1,out_counter)=40+2*out_counter;
            out_counter=out_counter+1;
        else
            staff(1,counter)=history(1,i,frame);
            staff(2,counter)=history(2,i,frame);
            counter=counter+1;
        end
    end
    if(size(staff_out,2)>0)
    staff = horzcat(staff,staff_out);
    end
    counter=1;
    out_counter=1;
    for i=num_staff+1:size(history,2)
        if((history(1,i,frame)==0)&&(history(2,i,frame)==0))
            pat_out(2,out_counter)=loc_free_patients;
            pat_out(1,out_counter)=40+2*out_counter;
            out_counter=out_counter+1;
        else
            patient(1,counter)=history(1,i,frame);
            patient(2,counter)=history(2,i,frame);
            counter=counter+1;
        end
    end
    if(size(pat_out,2)>0)
    patient=horzcat(patient,pat_out);
    end
    hold on

    if(num_bed_pat(frame)<total_beds)

        for i=1:(total_beds-num_bed_pat(frame))
    h=scatter(40+2*i,loc_free_beds,9,[0 1 0],'filled');
        end
    end


    h = scatter(wall_coord(:,2),wall_coord(:,1),1,[0 0 0], 'filled');
    % h =plot(wall_coord(:,2),wall_coord(:,1),'.k');
    h = scatter(bed(2,:),bed(1,:),9,[0 1 0], 'filled');
    %h=plot(bed(2,:),bed(1,:),'.y');
    if(size(staff,2)>0)
        h= scatter(staff(1,:), staff(2,:),10,[0 0 1],'filled');
        %h= plot(agent(1,1:num_staff), agent(2,1:num_staff),'.b');
    end
    if(size(patient,2)>0)
        h= scatter(patient(1,:), patient(2,:),10,[1 0 0],'filled');
        %h= plot(patient(1,:), patient(2,:),'.r');
    end


    %%3d crap
    currFrame = getframe;
    writeVideo(vidObj,currFrame);
    cla;
end


close(vidObj);


%scatter(wall_coord(:,2),wall_coord(:,1),5,[0 0 0], 'filled');
%scatter(agent(1,:), agent(2,:),5,[1 0.5 0],'filled');
%scatter(bed(2,:),bed(1,:),5,[0.25 1 0.5], 'filled');

%refreshdata
% drawnow
```

A video_3d | video_3d.m

```
function [] = video_3d( history, wall_coord, bed, t_total, fps, num_staff)
% Function which recieves a structure with agent groups, a image with the
% walls and the sinks, and displays them in an image.
vidObj = VideoWriter('example1.avi');
open(vidObj);
```

```
frame_total=t_total*fps;
step_size=round(size(history,3)/frame_total);
last_frame=size(history,3)/fps;

for frame=1:step_size:size(history,3)

    for i=1:num_staff
        staff(1,i)=history(1,i,frame);
        staff(2,i)=history(2,i,frame);
    end
    for i=num_staff+1:size(history,2)
        patient(1,i)=history(1,i,frame);
        patient(2,i)=history(2,i,frame);
    end
    screen_size = get(0, 'ScreenSize');

   fig=figure
     set(fig, 'Position', [0 0 640 480] );

 axis([0 max(wall_coord(:,2)) 0 max(wall_coord(:,1)) 0 120 0 1])
sizew=[1 1 3];
sizea=[1 1 2.5];
sizel=[0.75 0.75 2.5];
sizeb=[2 3 2];
sizep=[2 0.5 0.25];
sizeh=[0.5 0.5 0.75];

%plot walls
for i=1:size(wall_coord,1)
   plotcube( sizew, [wall_coord(i,2) wall_coord(i,1) 0], 1,[0 0 0]);
end
%plot staff
for i=1:size(staff,2)
    if((staff(1,i)~=0)&&(staff(2,i)~=0))
   plotcube( sizea, [staff(1,i) staff(2,i) sizel(3)], 1,[1 1 1]);%torso
   plotcube( sizel, [staff(1,i)+0.125 staff(2,i)+0.125 0], 1,[0 1 0]);%legs
   plotcube( sizeh, [staff(1,i)+0.25 staff(2,i)+0.25 sizea(3)+sizel(3)], 1,[1 0.5 1]);%head
    end
end
%plot patients
for i=1:size(patient,2)
    if((patient(1,i)~=0)&&(patient(2,i)~=0))
   plotcube( sizea, [patient(1,i) patient(2,i) sizel(3)], 1,[0.5 1 0]);
   plotcube( sizel, [patient(1,i)+0.125 patient(2,i)+0.125 0], 1,[0 1 0]);
   plotcube( sizeh, [patient(1,i)+0.25 patient(2,i)+0.25 sizea(3)+sizel(3)], 1,[1 0.5 1]);
    end
end
%plot beds
for i=1:size(bed,2)
   plotcube( sizeb, [bed(2,i) bed(1,i) 0], 1, [0 0.5 1]);
   plotcube( sizep, [bed(2,i) bed(1,i) sizeb(3)], 1, [1 1 1]);
end
 axis([0 max(wall_coord(:,2)) 0 max(wall_coord(:,1)) 0 120 0 1])
    currFrame = getframe;
   writeVideo(vidObj,currFrame);
   close all;
end


close(vidObj);


%scatter(wall_coord(:,2),wall_coord(:,1),5,[0 0 0], 'filled');
%scatter(agent(1,:), agent(2,:),5,[1 0.5 0],'filled');
%scatter(bed(2,:),bed(1,:),5,[0.25 1 0.5], 'filled');

%refreshdata
% drawnow
```

B driver_core | `driver_core.m`

```
agents=[20 30 40 50 60 70 80 90 100];% total nunmber of agents in simulation
staff2pat =[20 25 30 35 40 45 50]; %percentage of staff from all agents
bed2pat = [10 20 30 40 50 60 70 80 90]; %percentage of all pateints who are bedbound
count=36;
real_time_display=0;
```

```
%parellisation

matlabpool

 for c=1:size(bed2pat,2)
    for b=1:size(staff2pat,2)
       parfor a=1:size(agents,2)

        [results(a,b,c).history results(a,b,c).bed_coord results(a,b,c).nb_patient...
            results(a,b,c).nb_bed_patient results(a,b,c).nb_patient_tot results(a,b,c).nb_staff]=...
            evacuation(agents(a),staff2pat(b),bed2pat(c),'batch_image.bmp',real_time_display);
         results(a,b,c).num_agents=agents(a);
         results(a,b,c).perc_staff=staff2pat(b);
         results(a,b,c).perc_bed=bed2pat(c);

       end
    end
 end

 matlabpool close
```

C correct | correct.m

```
function [ results ] = correct( results )
%correct A function which attempts to "correct" results from when simulation
%trapped people in walls, causing the total time to reach the maximum in
%all cases.

for i=1:size(results,2)
    end_nb=min(results(i).nb_patient_tot);
    last_val=0;
    t=1;
     while(last_val==0)
         if(results(i).nb_patient_tot(t)==end_nb)
             last_val=1;
             results(i).time=t;
         else
             t=t+1;
         end

     end

 end

```

D iso_surf | iso_surf.m

```
function surfaces =iso_surf(results, axis_bed,axis_staff,axis_agents)

for i=1:size(results,2)

    for x=1:size(axis_bed,2)
        for y=1:size(axis_staff,2)
            for z=1:size(axis_agents,2)
                if((results(i).num_agents==axis_agents(z))&&(results(i).perc_staff==axis_staff(y))&&(results(i).perc_bed==axis_bed(x)))
                    surfaces(z,y,x)=size(results(i).history,3);
                 end
             end
         end
     end
 end
```

E iso_surf_correc | iso_surf_correc.m

```
function surfaces =iso_surf_correc(results, axis_bed,axis_staff,axis_agents)

for i=1:size(results,2)

    for x=1:size(axis_bed,2)
        for y=1:size(axis_staff,2)
```

```
            for z=1:size(axis_agents,2)
                if((results(i).num_agents==axis_agents(z))&&(results(i).perc_staff==axis_staff(y))&&(results(i).perc_bed==axis_bed(x)))
                    surfaces(z,y,x)=results(i).time;
                end
            end
        end
    end
end
```

F iso_surf | iso_surf.m

```
function surfaces =iso_surf(results, axis_bed,axis_staff,axis_agents)

for i=1:size(results,2)

    for x=1:size(axis_bed,2)
        for y=1:size(axis_staff,2)
            for z=1:size(axis_agents,2)
                if((results(i).num_agents==axis_agents(z))&&(results(i).perc_staff==axis_staff(y))&&(results(i).perc_bed==axis_bed(x)))
                    surfaces(z,y,x)=size(results(i).history,3);
                end
            end
        end
    end
end
```

G find_groups | find_groups.m

```
function max_in_area=find_groups(results, radii, step)
radii=radii.^2;

max_in_area.size=zeros(size(radii));


for idx=1:size(results,2)
    max_in_area(idx).size=zeros(size(radii));
    for time =1:step:size(results(idx).history,3)
        counter=zeros(size(radii));
        for a=1:size(results(idx).history,2)
            for b=1:size(results(idx).history,2)
                if((a~=b)&&(results(idx).history(1,a,time)~=0)&&(results(idx).history(1,b,time)~=0))
                    distsq=(results(idx).history(1,a,time)-results(idx).history(1,b,time))^2+(results(idx).history(2,a,time)-results(idx).history
                    for(r=1:size(radii,2))
                        if(distsq<radii(r))
                            counter(r)=counter(r)+1;
                        end
                    end
                end
            end
        end
        for r=1:size(radii,2)
            if(counter(r)>max_in_area(idx).size(r))
                max_in_area(idx).size(r)=counter(r);
                max_in_area(idx).time(r)=time;
            end
        end
    end
end
```