Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

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

Project Report

Folks in Civil Conflicts

Florian Gubler & Christian Käslin

Zürich
25.05.2012

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of Originality

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

I hereby declare that the written work I have submitted entitled

Folks In Civil Conflicts

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

Author(s)

Last name	First name
Gubler	Florian
Käslin	Christian

Supervising lecturer

Last name	First name
Donnay	Karsten
Balietti	Stefano

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

The above written work may be tested electronically for plagiarism.

Zürich, 25.4.12
Place and date

[illegible] [illegible]
Signature

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

Print form

Agreement for free-download

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

Florian Gubler　　　　Christian Käslin

Table of content

Inhalt

Abstract

The idea of this model is to simulate the evolution of a city with two conflicting parties – in our case the police and the Mafia. We tried to analyse the influence of different parameters as for example how brutally both sides proceed or how large the population density is.
Based on the Epstein model, we made several changes to work with only one agent. The agent could dynamically change sides. He could be part of the Mafia and hurt other agents. He could be part of the normal population or he could be a policeman and arrest other agents. The simulation shows that the world evolves best over time when one group is clearly stronger. Also the population density should not be too high, otherwise the satisfaction would decrease.

Individual Contributions

Both team members contributed equally to this project. Christian was more focused on the actual execution and analysis of the simulation; therefore these parts were mostly designed and implemented by him. Florian, on the other hand, took care of the implementation of the basic classes, parameters and functions of the model to prepare Christian's part.

Introduction and Motivations

Motivation

Initially we were interested to have a closer look at this topic because of the recent wave of revolution in several totalitarian Islamic countries, also known as the *Arabic Spring*. There the governments tried with more or less success to suppress the revolutionary tendencies of their people. However, there exist already a lot of simulations about this topic. Therefore we – first Christian – thought of the conflicts happening in Mexico for the last several years. In contrast to the *Arabic Spring* this is not written about often in the newspapers at the moment because it is nothing new, but nevertheless it is extremely brutal and causes a lot of suffering. Instead of the government trying to oppress the population we have here two conflicting parties – the police force and the Mafia – which both try to establish control but meanwhile they need to a certain extent the support of the population because – in contrast to for example Syria – they cannot just completely oppress the population, because most of their forces are needed to fight against their direct enemy. There are a lot of similarities between the two scenarios. This makes it possible to letting oneself be inspired be existing models but the differences are still large enough that it is an independent task.
The model is, however, not specified only for Mexico but can rather be used generally to simulate conflicts between police and organised crime. It can even be applied between two arbitrary conflicting parties which recruit their members from the general population and therefore have to keep some degree of popularity.

Main Questions

A model is, of course, expected to answer some interesting questions as this is normally its purpose. Especially in the beginning it was quiet difficult to find, what questions these might be, because we did not know yet what our model would actually be able to do. In the end we came up with the following questions:

1. How will the situation evolve over time if we define a specific set of initial parameters and how does it change for a different set of input values?
2. How should the population act to improve or at least preserve their level of satisfaction?
3. How should the police proceed to keep control over the Mafia without letting the population suffer too much?
4. How can the Mafia intervene and therefore become more powerful?

In the end we did or could not answer all these questions. For example we dropped the idea of an active population and just assumed that they cannot break the cycle of violence by just refusing to help any of the conflicting parties.

Description of the Model

Basis

Our model is based on the model of Joshua M. Epstein. He modelled the behaviour when a central authority suppresses a decentralized rebellion. We adapted the main ideas with the vision, the arrest probability and others, for example the exponential dependencies of some of the parameters. The model of Epstein contains two different kinds of people, the police and the agents (general population with potentially rebellious tendencies) whereas our model contains only one kind of agents who can either support the Mafia or the police or just be normal inactive people. This offers the advantage that we can change the loyalties of an agent over time without completely redefining him and of course it is simpler to implement. Additional to the probability to get arrested by a policeman in the Epstein model we have a probability to get hurt by a Mafia member. The two probabilities are exactly symmetric.

General

In our model there are two conflicting parties, on one side the Police and on the other side the Mafia as a criminal organisation. One could, however, just as easy take two random parties which are fighting each other as we did not use characteristics unique to Mafia or Police. Between these two groups there is the general population which mostly remains passive but of course constantly supplies new members for both adversaries.

World

We model the world of the simulation as a two dimensional array of locations. Each location has specific or constant properties and can be occupied by an agent. Two fields are neighbours, if their distance (defined by row and column) in the array is not bigger than a certain number (called vision). If vision is for example two then the neighbours of a field are all fields which are not more than two lines and two columns away.

Hospital and Prison

When a person is injured or arrested, he is removed from the world array and is saved in the hospital or prison array. Both are one dimensional location arrays with an x-coordinate of -1 (hospital) or -2 (prison) and a y-coordinate which is equal to its index in the array. Starting with the jail time or injury value of the location where the agent was arrested or injured, a counter is decremented with each time step. When it reaches zero, the agent is released and placed on a randomly chosen free location on the map.

Behaviour

Movement:
Active members of both, Mafia and police, just go to a random empty field in their neighbourhood (this means that it is within *vision* fields from their current position). Normally, passive agents will try to avoid areas where the influence of Mafia and police is high.

Action:
A policeman will choose a random non-empty location in his neighbourhood and arrest the agent standing there with a probability which depends on the properties of the person in question and on the properties of his neighbours. Analogously, a Mafia member will injure a random neighbour of himself with a certain probability. Both, the probabilities and how long an arrested or injured person has to stay in prison or the hospital, will be defined later.

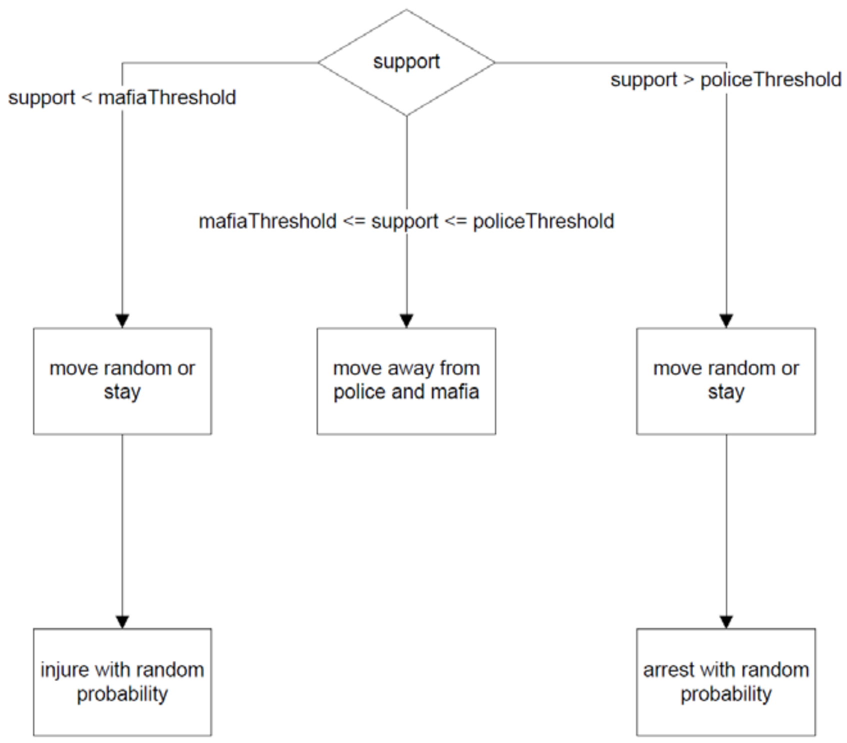
support
support < mafiaThreshold
support > policeThreshold
mafiaThreshold <= support <= policeThreshold
move random or stay
move away from police and mafia
move random or stay
injure with random probability
arrest with random probability

Figure 1: agent behaviour and action

Agents

In contrast to similar projects from previous years we have only one type of agent which just characterizes a normal person. This person is defined by several constant and non-constant properties which then define whether he is a policeman, a member of the Mafia or just part of the normal population. First we would like to present these properties and how they influence the person's actions:

1. Satisfaction: Defines how happy the person is. On one hand this is an interesting output, because the goal of the police should of course be to make the general population as happy as possible. On the other hand a person, who already has everything he wants, is less likely to take sides in the conflict, because he has a lot to lose but almost nothing to gain.
2. Wealth: The basic wealth of an agent is assumed to be constant over time for a specific agent and is therefore independent of the evolution of the conflict but of course it is different for every agent. We use this property as initial satisfaction of the agent; it is set between zero and one, where one means extremely rich.
3. Support: Defines whether a person is neutral or has tendencies towards one of the conflicting parties. It has a normally distributed initial value and will change over time depending on the person's other properties and on whom his neighbourhood supports. We are using only one property here and just say that, if a person's support is above a certain threshold (Police-Threshold: 0.75 by default), he acts as an active policeman and if it is below another threshold (Mafia-Threshold: 0.25 by default), he acts as an active Mafia member. If it is between the two thresholds

he is part of the general population. This definition is of course an approximation, as it for example neglects the fact that it takes time to train a policeman.

4. Courage: Defines how strongly a person's support depends on his neighbourhood. The more courageous an agent is, the less he will be influenced by his neighbours. This property is assumed constant over time but of course different for every agent.
5. Influence: Defines how strongly the person influences his environment. As an example one could say that a heavily muscled and tattooed man with a look on his face which promises violence will influence his neighbours more strongly than a ten year old child. Again this property varies from agent to agent but is constant over time for any specific person.
6. Willingness to assume a risk against police or Mafia: Defines the probability that the agent takes action against one of the conflicting parties.
7. Location: Where the person is standing right now and therefore who his neighbours are.
8. Previous probability to be arrested/injured: The probability to be arrested or injured during the last time step (is just copied from the value of the location where the agent was standing). It is interesting to see the change in the probabilities for a specific agent (influences his satisfaction).

All of these properties are within a range between zero and one. The constant properties (wealth, courage, influence and initial support) are normally distributed with a mean value of 0.5 and a standard deviation of 0.12 (empirically evaluated number).

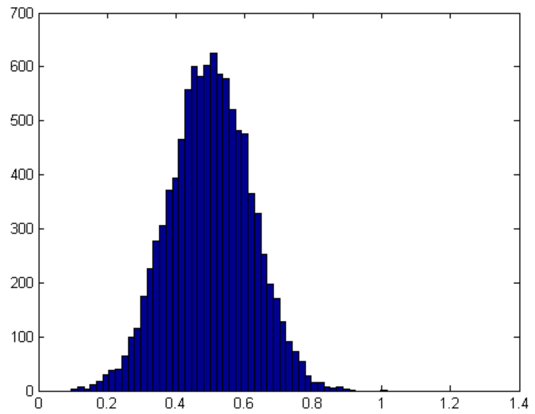

Figure 2: random distribution

Result of a test of a normal distribution with mean value of 0.5 and standard deviation of 0.12 with 10000 values. We chose these parameters, because we wanted of course a mean value of 0.5 (meaning neutral) and only a negligible amount of values not being in the set interval of [0,1] (these are just set to 0 or 1).

Locations

As described above, a person's actions are heavily influenced by his environment. Therefore we identify each location on our map with the following constant or non-constant values which define how a person standing at this location is affected:

1. x and y coordinates: The position of the location in the world array. This is important to identify the neighbours of a location. (assumed constant over time)

2. Vision: Defines the number of neighbouring locations with which an agent standing here is able to interact. By interaction we mean influence or being influenced as well as how far a person is able to move in a specific amount of time and how far away another agent can stand and still be arrested or injured by this agent. We basically just defined it as the number of fields in each direction. This property is deemed constant over time and for simplicity also for all locations.
3. Jail time: Defines for how long (in unit of time steps) a person standing on this location will have to go to prison if he is arrested by a policeman. This property is deemed constant over time and for simplicity also for all locations.
4. Severity of Injury: Defines for how long (in unit of time steps) a person standing on this location will have to go to the hospital if he is injured by a Mafia member. This property is deemed constant over time and for simplicity also for all locations.
5. Influence of Police and Mafia: Defines how powerful the two conflicting parties are at this location. It is just the sum of the influences of the active members of both parties in the neighbourhood (defined by the *vision* property) of the location.
6. Probability to be arrested or injured: Defines how likely it is that an agent standing here is arrested or injured. Obviously this depends on the properties of the agent who is standing here.
7. Person: The agent standing at this location, if it is occupied.

The properties vision, jail time and severity of injury have to be positive integers. On the other hand the influences and probabilities are real numbers between zero and one.

Changes of the properties:

Notation: X_k means the property X in the time step k.

Constants:

- Courage C
- Vision V
- Jail time JT
- Severity of Injury SI

Variables:

- Influence of the police IP
- Influence of the Mafia IM
- Total Support TS
- Satisfaction S

Probability to be arrested PA: $PA_{k+1} = (1 - TS_k) \cdot \left(1 - e^{-\frac{IP_{k+1}}{IM_{k+1}}}\right)$ The probability to be arrested obviously decreases with increasing support for the police. Additionally, it decreases if there are a lot of Mafia members present, because then it would be more risky for a policeman to arrest someone. We adopted the exponential dependency from Epstein.

Probability to be injured PI: $PI_{k+1} = TS_k \cdot \left(1 - e^{-\frac{IM_{k+1}}{IP_{k+1}}}\right)$ The probability to be injured obviously decreases with increasing support for the Mafia (and therefore decreasing total support). Additionally it decreases, if there are a lot of Policemen present because then it would be more risky for a Mafia member to injure someone. Again we adopted the exponential dependency from Epstein.

Satisfaction S: $S_{k+1} = S_k - \left(e^{-\frac{50}{JT}} \cdot \left(PA_k - PA_{k-1}\right) \cdot \left(1 - TS_k\right) + e^{-\frac{50}{SI}} \cdot \left(PI_k - PI_{k-1}\right) \cdot TS_k\right)$ The new satisfaction is the old satisfaction minus the increased probability to be injured or arrested compared to the last time step. Both probabilities are weighed with the consequences if the agent really were to be arrested or injured and their support because obviously it is worse to be arrested for a longer time or injured more severely but a very loyal member of one of the sides might accept the chance to suffer for his cause.

Risk against police RP: $RP_{k+1} = \frac{\left(1 - TS_k\right) \cdot C}{S_k \cdot PA \cdot JT}$ The risk one is ready to assume against the police is bigger for more courageous people and is decreased if the agent favours the police or if either the probability to be arrested or the jail time is increased. Additionally a very satisfied person has no reason to take any chances and will therefore be less ready to risk anything against one of the parties.

Risk against Mafia RM: $RM_{k+1} = \frac{TS_k \cdot C}{S_k \cdot PI \cdot SI}$ The risk one is ready to assume against the Mafia is bigger for more courageous people and is decreased if the agent favours the Mafia or if either the probability to be injured or the severity of an injury is increased. Additionally, a very satisfied person has no reason to take any chances and will therefore be less ready to risk anything against one of the parties.

Support TS: $TS_{k+1} = TS_k - e^{-\frac{1}{2} \cdot (1 - TS_k)} \cdot NP_{k+1} + e^{-\frac{1}{2} \cdot TS_k} \cdot NM_{k+1}$ The new support is just the previous one minus the current risk the agent is ready to assume against the police and plus the risk he is ready to assume against the Mafia. Both summands are damped to achieve some level of stability: The extremer the support becomes, the less it changes towards this extreme.

Output

As the whole idea of the simulation is to get an interesting output, it was quiet important to specify the interesting output parameters we wanted to get out of the system. These output parameters are actually just some of the properties of the agent class; therefore they are not pure output values but in truth also define the present state of the people and have an effect on the further development of the simulation. We deemed the following properties the most interesting:

- Average Satisfaction of the Population: As the main goal of the police should be to not only protect the population but to do so without troubling their daily lives, this for sure is an important information.

- Average Risk the agents are ready to take against Mafia and police: Although these parameters correlate with the support of the agents, we deemed it interesting, mostly because it was quiet helpful in the creating process to see the correlation and make changes in the formulas to update the parameters.
- Average Total Support of the Population: This complements the information about the number of active policemen and Mafia members, because it might be that a lot of people tend towards one of the two parties but are not (yet) ready to become active. This situation would not show up in the number of active people but it would definitely affect the average support.

Because of time issues we could, in the end, only implement the three dimensional analysis of the influence of the input parameters on the satisfaction and not of the support, although it is also saved after each time step.

Input

Next we had to define the input parameters to control or at least influence the wanted outputs defined above. In the following paragraph we will explain which inputs we chose, why they are of any significance and how we expected them to influence the outcome.

- Jail Time: Evidently this affects the actions of the agents because the longer people have to go to jail, if they are arrested, the less likely they will be to risk being arrested. On the down side there are unfortunately always innocents who are arrested as well (although with a smaller probability) and therefore the general satisfaction will probably decrease at least temporarily (in the long run it might increase again if the Mafia is weakened and therefore the chance to be injured decreases).
- Severity of Injury: Analogous to the jail time this affects both the support and the satisfaction of the people: The higher it is the more people will probably show tendencies towards the Mafia but their satisfaction will decrease – at least at first.
- Population Density: For a low population density the probability for an agent to meet another agent is smaller. Therefore there will probably be fewer changes in the output parameters as satisfaction and support, because they obviously do not change, if there is no other person around. When the person finally meets another one, however, the change in satisfaction and support will be more drastic: If an agent who had no neighbours before now suddenly meets a policeman the probability to be arrested jumps from zero to a value perhaps quiet far away from zero so this will heavily decrease his satisfaction and probably change his support too.
- Wealth (Satisfaction in the Beginning): As mentioned before, a rich or satisfied person is far less likely to take any action against one of the conflicting parties because he has little to gain and a lot to lose. Therefore the changes in support will probably be slower which again will affect the other properties.

Again we had to reduce the number of different inputs, this time because the simulation would have taken too long otherwise. Therefore we dropped the last point (Wealth).

Implementation

General

Both, agents and locations, are implemented as own classes in MATLAB. An agent or location object then contains the properties mentioned above and in the case of an agent a number by which to identify him. In addition there are several implemented member functions to manipulate these properties. Both classes are handle classes. This ensures that MATLAB always uses *call by reference* on objects of these classes which means that if an agent object is given to a function, this function uses the original object and not a copy. Therefore all changes made to the object are retained. This was necessary to allow the simulation to change the agent standing on a location using the locations member variable *person* because otherwise just a copy would have been used (and vice versa).

An agent with *number* zero is considered an empty agent: If a location is not occupied its member variable *person* is set to an empty agent with *number* =0.

Agent Member functions

· initAgent: Initializes the agent with his constant and initial values.
· newSat: Updates the agent's satisfaction value according to the changes of his situation.
· newRisk: Updates the agent's willingness to assume risks against the two conflicting parties.
· newSup: Updates the agent's support value.
· toPrison: Sends the agent to prison. This means that he is removed from the world array and stored in a separate prison array. At the arrest his satisfaction obviously drops significantly dependent on the jail time (the longer he has to go to prison, the unhappier he is of course). This is computed by the formula $S_{k+1} = S_k \cdot e^{-\frac{JT}{10}}$. His satisfaction will then remain constant while he is in prison and be increased again (but not to the previous level) when he is released.
· toHospital: Sends the agent to the hospital. This works analogously to being arrested, just with a hospital array instead of a prison array. The formula is the same as well.
· moveTo: Moves the agent to a specified new field and updates the changed properties of both the agent and the two locations.
· neighbours: Gets an agent's neighbouring locations by using the external function getNeighbours.
· move: *Moves* the agent to a random non-occupied field in his vicinity using the external function *movePerson*.

Location Member functions

- initLocation: Initializes the location with constant and initial values.
- probabilities: Updates the probability to be injured and arrested for the person standing on this field.
- newInfluences: Updates the influence of Mafia and Police at this location.
- newPenalties: We considered changing the jail time and severity of injury over time, assuming that if there were a lot of policemen and almost no Mafia agents, the police might lower the jail time to give the population more comfort (analogously for the Mafia). In the end we did not use it, however, because it hardly ever changed anything since relative difference between the number of policemen and active Mafia members was not high enough.
- neighbours: Gets the neighbouring fields of the location using the external function *getNeighbours*.

General Functions

- randomvalue: Gives back a vector of dimension n with normally distributed elements with mean value 0.5 and standard deviation 0.12.
- findAgents: Looks through the entire world and gives back an array with all non-empty agents and the number of agents there are.
- findAllAgents: Does basically the same thing as *findAgents* but includes the agents who are currently in prison or in the hospital.
- getNeighbours: If the input variable *chooser* is one, the function returns an array with all non-empty neighbouring locations and their number. If *chooser* is zero, it does the same with the empty neighbours.
- movePerson: Moves a given person according to the movement rules defined above. Additionally, it checks whether the person is an active policeman or Mafia member and calculates accordingly, whether the person can and does hurt or arrest another agent. If he does, the injured or arrested person is transported to prison or hospital using the corresponding agent member function.
- initAll: Sifts through the world array and initializes the locations and the agents standing on them. As the properties of the agents and locations depend on each other, the order of the initializations is important. This function is only executed once, in the beginning.
- updateAll: Analogous to *initAll* but instead of initializing the agents and locations this function is devised to update them to their new values using the corresponding member functions of the agent and location class; again the order of the updates is important. This function is executed after each time step.
- createWorld: Creates a world array of the size defined in the configuration file and fills it with the declared number of agents who are distributed on the map randomly. Then it assigns all the empty fields with empty agents and initializes all agents and locations using *initAll*.
- displayWorld: Plots a graphic display of the world array, marking the agents with dots of a colour between red and blue. Their colour is defined by their support where completely blue corresponds to a support of one (policeman), completely

red a support of zero (Mafia member) and everything in between is shown by a mixed colour.

- moveAll: Goes over all the agents in the world array and moves them using *movePerson*. This function is executed with each time step.
- reentry: Is called by *checkReentry*, when an agent is to be reintegrated into the world array. It just puts him into a randomly chosen empty field in the world array.
- checkReentry: Goes through the hospital and prison arrays and updates the remaining number of time steps until the prisoners or patients are released (just decrements the number with each time step). When an agent is to be released the function calls the function *reentry* to reintegrate him into the world array. Additionally it changes the satisfaction back: $S_{k+1} = S_k \cdot e^{\frac{JT}{10}}$.
- getStatistics: Collects the state of the output parameters (defined below) in each time step. These variables can then be plotted by *plotStatistics*.
- plotStatistics: Plots the output parameters given by the function *getStatisctics* of one simulation over time. This function was used primarily in the development of the simulation to check the results.
- analyseWorldsizePopulation: Analyses the impact which changes in the size of the world array and the number of agents have on the satisfaction of the population for constant values of jail time, injury and threshold to become active. The collected data is then presented in a three dimensional plot.
- analyseJailtimeInjury: Acts analogously to *analyseWorldsizePopulation* just this time the influence of changes in jail time and injury for a constant world and population is analysed and plotted.
- analysePoliceThresholdMafiaThreshold: Has the same functionality again but this time the influence of the thresholds for active agents is analysed and plotted.
- copy: Makes a copy of a matrix into a .mat file. This is used to make a deep copy. Because the objects in the world are all passed by reference, a shallow copy is made by default, which is not usable for value storage.
- Threshold for Mafia and police: Defines the support value an agent needs to be a member of the Mafia or a policeman.

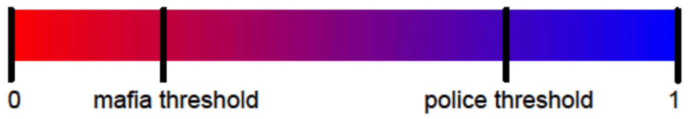

Figure 3: police and Mafia threshold

World map

The world map displays a 2D map with all agents, which are in the world. Agents in the prison or in the hospital are not displayed.

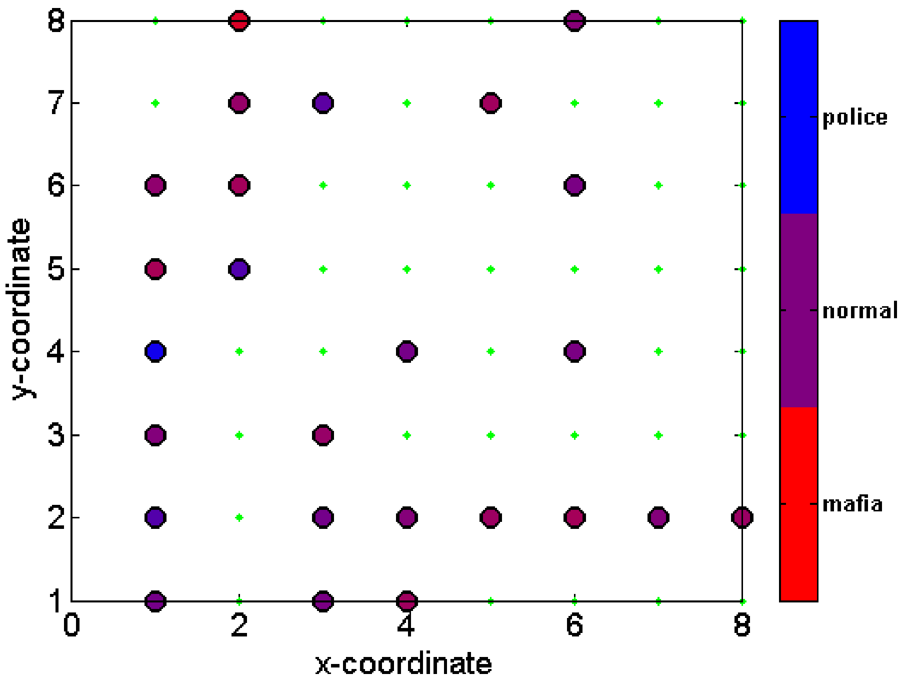

Figure 4 world map

Simulation Results and Discussion

Setup

The simulation consisted mainly of three parts: In each part one set of input parameters was variable while the others were constant.

1. Variable size of the world and population
2. Variable jail time and injury
3. Variable thresholds for active agents

We than had the *mainAllSimulation* script which basically just called all the functions at the right moment to execute the simulation. It is mostly identical to the *main* file but just collects the data without analysing it because we wanted to do that separately afterwards. It executed the three parts sequentially and stored the results in separate .mat files. We could then use the analysing parts of the main file to extract the useful information from the data which the *mainAllSimulation* procured.
The simulations are spitted in two different types. First the lifetime simulation describes how simulations with constant parameters evolve over time. Secondly the parameter sweep simulations compare two different parameters against each other.

In general and if not tested specific for a simulation the following parameters are used for all following simulations.

n_lifetime	100
n_worldHeight	10
n_worldWidth	10
n_agents	50
n_vision	1
n_jail time	5
n_injury	5
n_policeThreshold	0.75
n_mafiaThreshold	0.25

Lifetime simulation

Interesting input parameters are jail time and injury. The following three cases are studied, equal values for jail time and injury, higher jail time and higher injury.
With equal values for jail time and injury, the satisfaction decreases rapidly with our model. A possible reason for this is neither of the parties wins. Both are equal and therefore the amount of agents who gets arrested and the amount of agents who gets injured increases. Hence the satisfaction decreases.

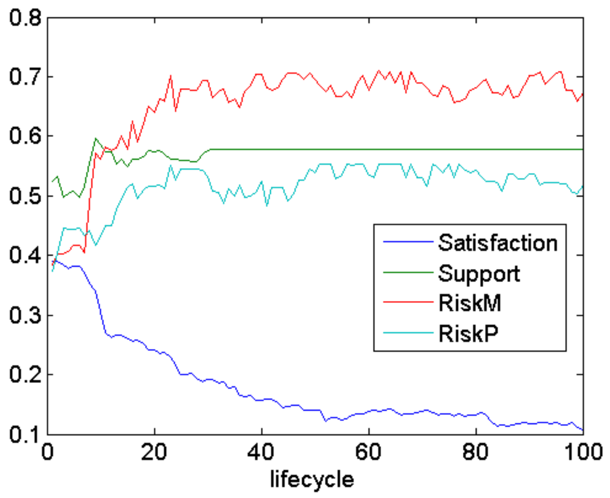

Figure 5: evolve over time with equal jail time and injury

To simulate a stronger police, one possibility is to make the jail time value higher than the injury value.

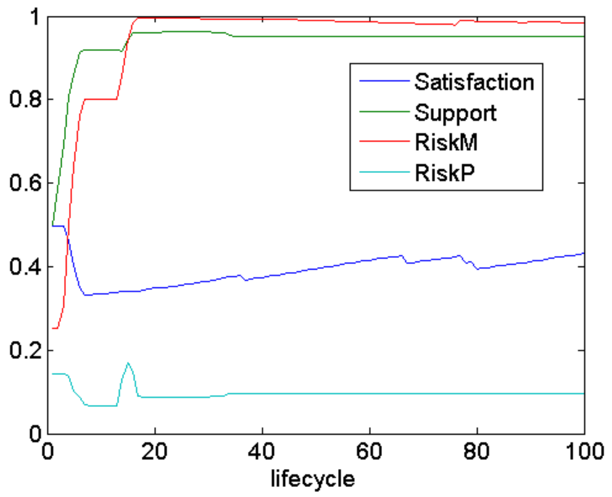

Figure 6: evolve over time with higher jail time

The simulation shows that the satisfaction does not decrease much. The risk against the police stays very low, because the agents fear the consequences. But the risk against the Mafia increases strongly. So the agents are willing to cooperate with the police and work against the Mafia.

To simulate a stronger Mafia, the injury value is higher than the jail time value.

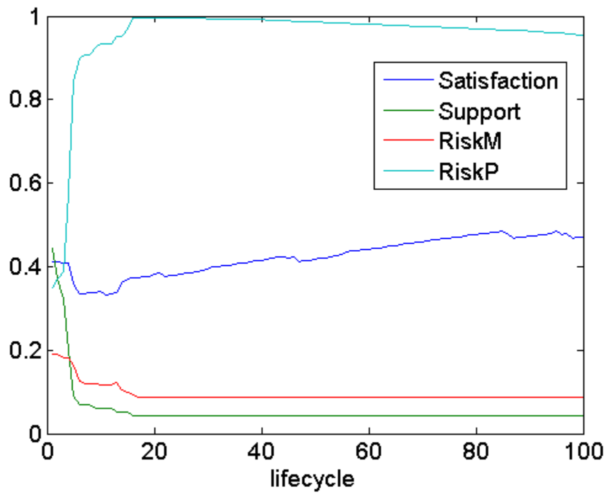

Figure 7: evolve over time with higher injury

This simulations show nearly an inverse plot to the last simulation. In this simulation the agents cooperate with the Mafia and work against the police. The satisfaction increases too.

This three simulations shows that if two parties have equal power and fight against each other, that satisfaction is lower as if one of the parties is stronger and has the control over the region. This is comparable with the real world. If one party takes care of the stability, the satisfaction is higher than if two or more parties fight against each other for the overall control.

Parameter sweep simulation

In this simulation only the average satisfaction at the end is of interest. Two parameters are compared against each other. The result is shown by a 3D plot.

World size and population:

To figure out how the model behave with differential population density, the world height and world width are compared to the amount of agents.

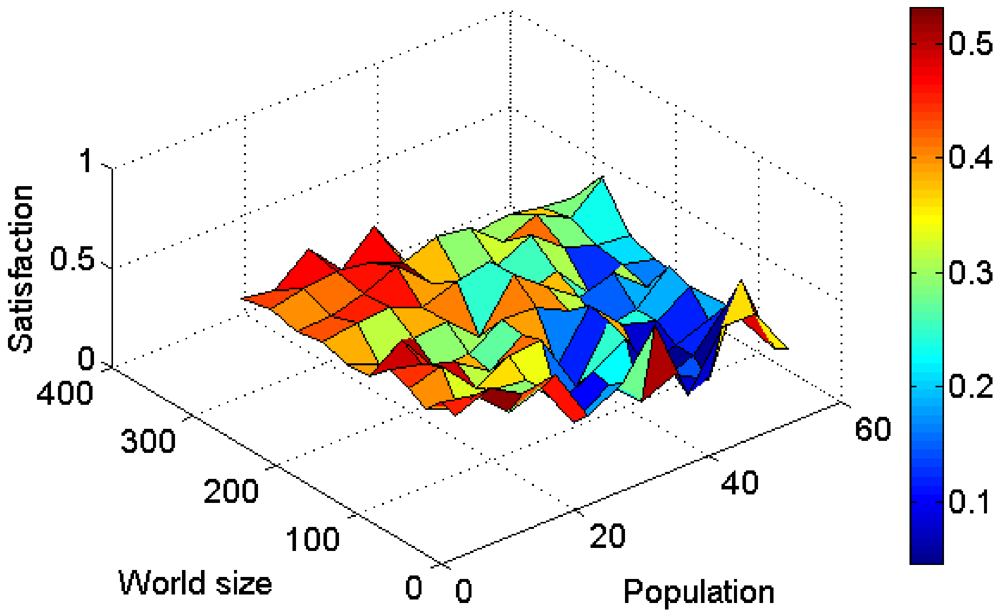

Figure 8: world size compared with population

The simulation shows that a higher population density decreases the satisfaction. This is because with a higher population density the probability that the neighbour field is occupied increases. Therefore the probability to get arrested or injured increases too. The highest satisfaction is reached with the lowest population density. In these worlds the agents rarely have neighbours around their field. So the interaction with other agents decreases and therefore the agents are happier in this model.

Jail time and severity of injury:

One possibility to compare different balance of power between the two parties in our model is to vary the jail time and the severity of injury. Higher jail time represents a stronger police; higher severity of injury represents a stronger Mafia.

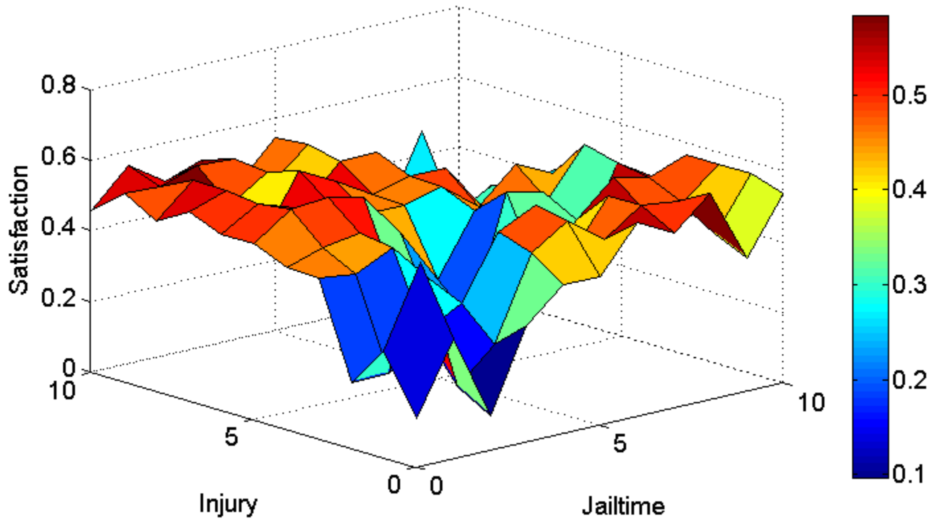

Figure 9: injury compared with jail time

<u>Police and Mafia threshold</u>:

The other possibility to compare in our model different balance of power between the two parties is to vary the threshold at which an agent becomes an active police or Mafia. Threshold equals zero for Mafia means there are no active Mafia members. With the threshold of 0.5, about half of the agents are active Mafia members. With threshold one for police, there are no active police men and with threshold 0.5, about half of the agents are active police men. The simulation shows that when almost all agents are either active Mafia members or active police men the satisfaction goes very low. If one side is stronger than the other the satisfaction is higher.

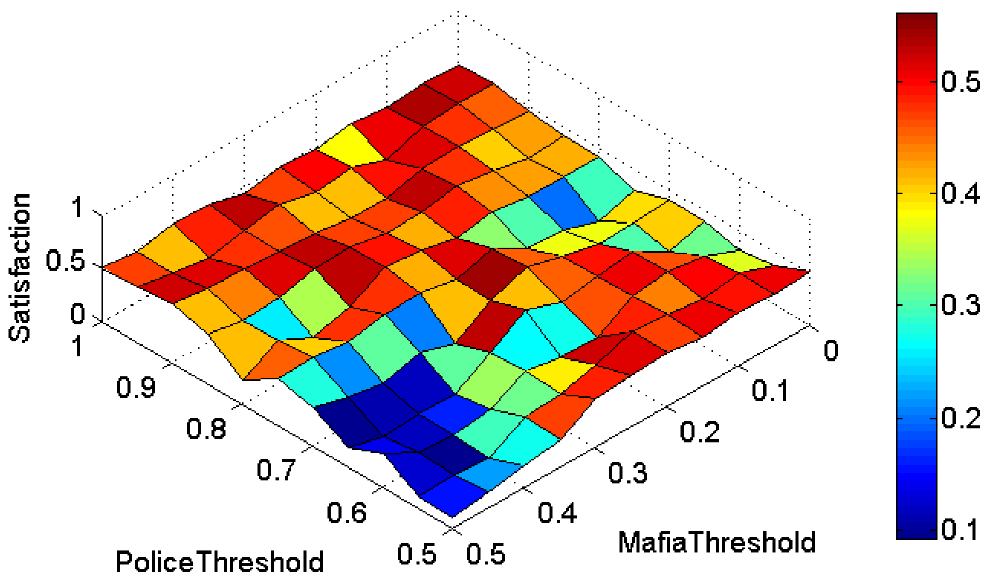

Figure 10: police threshold compared with Mafia threshold

Performance

Using classes for the agents and the location was very helpful allows a more natural implementation than just with variables and functions. But for simulation this was a bit of a handicap. The access time for a class is higher than for a normal variable. To simulate all in an appropriate time we could not use more than 100 agents. Also the saved simulated data are huge: With multiple runs it was several gigabytes.

Discussion

Our model shows that the situation with two equally strong parties leads to the worst satisfaction. At least one group should have control over a region to make them happy. Hence if one group is stronger than the other, the evolution over time will be positive and constant.
The general population should in general support the stronger group to decrease their calamity.
The police should in our model always increase the jail time if the severity of injury gets increased, to preserve their control.
The Mafia has to act the other way around and increase the severity of injury to preserve their control.

Summary and Outlook

We have model a conflict between two different groups with normal population in-between. The threshold at which an agent can act as policeman or Mafia member could vary dynamically. To make one group stronger either the jail time or the severity of

injury could be increased. The simulation has shown that one group had to be stronger for a sufficient satisfaction.
Further investigations could analyze a police operation, which could be simulated by adding at a given time several agents with support above the police threshold.

References

EPSTEIN, J. M. Modeling civil violence: An agent-based computational approach http://www.pnas.org/content/99/suppl.3/7243

Code

agent.m

```
classdef agent < handle
    % the "< handle" assures that when an agent-object is given to a
    % function it is passed by reference and not by value: Therefore the
    % Membervariable "person" of a location object is changed if the agent
    % in that location is changed. But if I want to change the "person"
    % Variable to another agent I can just do that with
    % obj.person=agent(a,b,c,d) and the person-reference will just be
    % redirected to the new agent without changing the old one


    properties

        number      % for identification of the agents, number = 0 is an empty field
without an agent
        wealth
        courage
        influence
        basicSupport    %General tendency of the agent towards Police or Mafia without
interference

        place          % present location of the agent
        satisfaction
        support        % Total Support
        riskM         % Risk, the agent is ready to assume against Mafia
        riskP         % Risk, the agent is ready to assume against Police

        pAbefore       %probability of Arrest one iteration step before: better here then
in
                       %location because the agent is moving and has to take it with
him...
        pIbefore       %probability of Injury one iteration step before (at the beginning
=0
                       %so that in the first change the pIbefore is set to 0)


    end


    methods
        % initializes the agent-object: has basically the role of a
        % constructor, but doesn't have problems with matrices.
        function initAgent(obj,num)
            obj.number=num;
            if(num==0)
                return
            end
            obj.wealth=randomvalue(1);
            obj.satisfaction=obj.wealth; %wealth is the initial value of the satisfaction
            obj.courage=randomvalue(1);
            obj.influence=randomvalue(1);
            obj.basicSupport=randomvalue(1);
            obj.support=obj.basicSupport;   %basic support is the initial value of
support
            obj.pAbefore=0;
            obj.pIbefore=0;
        end



        % the first input of the functions has to be the agent
        % itself: when using the function you can just wright
        % example.newSat() instead

        % Changes the satisfaction
        function newSat(obj)

            if(obj.place.jailtime ~= 0)
                obj.satisfaction=obj.satisfaction-(exp(-1/(obj.place.jailtime/50))*...
                    (obj.place.pArrest-obj.pAbefore)*(1-obj.support));
            end
            if(obj.place.injury ~= 0)
                obj.satisfaction=obj.satisfaction-(exp(-1/(obj.place.injury/50))*...
                    (obj.place.pInjury-obj.pIbefore)*(obj.support));
            end

            if(obj.place.pArrest == 0)
                obj.satisfaction = obj.satisfaction + rand(1)*0.005*(exp(-
2*obj.satisfaction));
            end

            if(obj.place.pInjury == 0)
                obj.satisfaction = obj.satisfaction + rand(1)*0.005*(exp(-
2*obj.satisfaction));
            end

            if(obj.satisfaction>1)
                obj.satisfaction=1;
            end
            if(obj.satisfaction<0)
                obj.satisfaction=0;
            end
            if(isnan(obj.satisfaction))
               disp('nan');
               obj.satisfaction = 0;
            end

            %save old Arrest and Injury value
            obj.pAbefore = obj.place.pArrest;
            obj.pIbefore = obj.place.pInjury;
        end

        % Changes the two risk-values.
        function newRisk(obj)
             %updates the Risk against the Police
            if(obj.satisfaction*obj.place.pArrest*obj.place.jailtime==0)
                %would result in dividing by zero (is most likely triggered by pArrest=0
                %(for example if there are no neighbours present)
                obj.riskP=(1-obj.support)*(1-obj.satisfaction)*3;
            else

                obj.riskP=(1-
obj.support)*obj.courage/(obj.satisfaction*obj.place.pArrest*obj.place.jailtime);
            end

            %updates the Risk against the Mafia
            if(obj.satisfaction*obj.place.pInjury*obj.place.injury==0)
```

```
                %would result in dividing by zero (most likely because
                %there are no mafia-members so pInjury=0
                obj.riskM=obj.support*(1-obj.satisfaction)*3;
            else
            
obj.riskM=obj.support*obj.courage/(obj.satisfaction*obj.place.pInjury*obj.place.injury);
            end
            
            % For Safety: The Risks have to be between 0 and 1: if they
            % exceed their boundaries they're just set to the
            % boundary-values
            if(obj.riskP>1)
                obj.riskP=1;
            end
            if(obj.riskP<0)
                obj.riskP=0;
            end
            if(obj.riskM>1)
                obj.riskM=1;
            end
            if(obj.riskM<0)
                obj.riskM=0;
            end
            
        end
        
        % Changes the total Support
        function newSup(obj)
            
            %updates the Support-Value
            obj.support=obj.support - exp(-1/2*(1-obj.support))*obj.riskP + exp(-
1/2*obj.support)*obj.riskM;
            
            %If the support would be leave the defined area: just set it to
            %the boundary-value
            if(obj.support>1)
                obj.support=1;
            end
            if(obj.support<0)
                obj.support=0;
            end
            
            
        end
        
        %sends agent to prison
        function toPrison(obj)
            
           global prison;
           global k; %constant to calculation a usable satisfaction
           %Create a prison-Cell-location where the agent will be put
           prisonCell=location;
           %initialize it with x=-2, y=position in the prison-array,the
           %jailtime (how long the prisoner has to stay there), vision and
           %injury are unimportant so just set them to zero
           prisonCell.initLocation(-2,length(prison)+1,obj.place.jailtime,0,0);
           %Set the Person in the Cell to the given agent
           prisonCell.person=obj;
           %take the jailtime from the place
           prisonCell.jailtime = obj.place.jailtime;
           %create an empty-agent and set the place where the arrested
           %agent stood to empty
           empty=agent;
           empty.number=0;
           obj.place.person=empty;
           %sets the location of the arrested person to his prison cell
           obj.place=prisonCell;
           %attaches the prisonCell to the prison array at its end.
           prison(length(prison)+1)=prisonCell;
           obj.satisfaction=obj.satisfaction/2; %half the satisfaction
           %being arrested severely lessens the satisfaction obviously: the more so the
longer the jailtime is
           obj.satisfaction=obj.satisfaction*exp(-obj.place.jailtime/k);

           %{
           output1='Prison'
           output2=obj.number
           output3=length(prison)
           %}
        end

        %sends agent to the hospital
        function toHospital(obj)

            global hospital;
            global k;

           %create new location with the "hospital-room"
           room=location;
           %initialize it with x=-2, y=position in the prison-array usw
           room.initLocation(-1,length(hospital)+1,0,obj.place.injury,0);
           %sets the rooms person to the patient
           room.person=obj;
           %take the injury from the place
           room.injury = obj.place.injury;
           %make new agent to set on the now empty field
           empty=agent;
           empty.number=0;
           obj.place.person=empty;
           %says to the patient where he is now
           obj.place=room;
           %attaches the room to the hospital array
           hospital(length(hospital)+1)=room;
           obj.satisfaction=obj.satisfaction/2;
           %being injured severely lessens the satisfaction obviously: the more so the
more severe the injury is
           obj.satisfaction=obj.satisfaction*exp(-obj.place.injury/k);

        end

        %moves an agent to a new location
        function moveTo(obj,target)
            %creates empty agent
            empty=agent;
            empty.initAgent(0);

            obj.place.person=empty; %old field is now empty
            target.person=obj;  %new field is now set to the agent
            obj.place=target;   %the agent is now set to the new field

        end

        %gets the neighbours using getNeighbours (unnecessary but nice)
        function [neighbours,counter] = neighbours(person,world,chooser)
           [neighbours,counter] = getNeighbours(person,world,chooser);
        end

        %moves an agent using movePerson (unnecessary but nice)
        function move(obj,world,hospital,prison)
            movePerson( obj,world,hospital,prison )
        end

    end

end
```

analyseJailtimeInjury.m

```
function [] = analyseJailtimeInjury(init)
%ANALYSEJAILTIMEINJURY
%Compares different jailtime and injury values and makes a 3D plot
    global prison
    global hospital

    averageSatisfactionArray =
zeros(length(init.model.param_jailtime),length(init.model.param_injury));
    for indexJailtime = 1:length(init.model.param_jailtime)
       for indexInjury = 1:length(init.model.param_injury)
            for rCount=1:init.globals.RUNS
                load(strcat('data/', init.globals.NAME ,
'world_JailtimeInjury_',int2str(init.model.param_jailtime(indexJailtime)),'_',int2str(ini
t.model.param_injury(indexInjury)),'_',int2str(rCount)));
                %get the last world
                world = worldArray(1 + ((init.model.n_lifetime-
1)*init.model.n_worldHeight):init.model.n_lifetime*init.model.n_worldHeight,(1 + (1-
1)*init.model.n_worldWidth):1*init.model.n_worldWidth);
                prison =
prisonArray(init.model.n_lifetime+1,1:prisonLengthArray(init.model.n_lifetime+1));
                hospital =
hospitalArray(init.model.n_lifetime+1,1:hospitalLengthArray(init.model.n_lifetime+1));
                [agents,amount] = findAllAgents(world);

                %averageSatisfaction = sum([agents.satisfaction]);
                averageSatisfaction = 0;
                for index = 1:amount-1
                    averageSatisfaction = averageSatisfaction +
agents(index).satisfaction;
                end
                averageSatisfaction = averageSatisfaction/amount;
                averageSatisfactionArray(indexInjury,indexJailtime) =
averageSatisfactionArray(indexInjury,indexJailtime) + averageSatisfaction;
            end
            averageSatisfactionArray(indexInjury,indexJailtime) =
averageSatisfactionArray(indexInjury,indexJailtime)/rCount;

       end
    end
    x = init.model.param_jailtime;
    y = init.model.param_injury;
    view(0,90);
    surf(x,y,averageSatisfactionArray);
    colorbar;
    set(gca,'FontSize',14);
    xlabel('Jailtime');
    ylabel('Injury');
    zlabel('Satisfaction');
end
```

analysePoliceThresholdMafiaThreshold.m

```
function [] = analysePoliceThresholdMafiaThreshold(init)
%ANALYSEPOLICETHRESHOLDMAFIATHRESHOLD
%Compares different police and mafie threshold values and makes a 3D plot
    global prison
    global hospital
    averageSatisfactionArray = zeros(length(init.model.param_mafiaThreshold),...
        length(init.model.param_policeThreshold));
    for indexPoliceThreshold = 1:length(init.model.param_policeThreshold)
        policeThreshold = init.model.param_policeThreshold(indexPoliceThreshold);
       for indexMafiaThreshold = 1:length(init.model.param_mafiaThreshold)
           for rCount=1:init.globals.RUNS
               mafiaThreshold = init.model.param_mafiaThreshold(indexMafiaThreshold);
```

```
                load(strcat('data/', init.globals.NAME ,
'world_PoliceMafiaThreshold_',...

num2str(policeThreshold),'_',num2str(mafiaThreshold),'_',int2str(rCount),'.mat'));
                %get the last world
                world = worldArray(1 + ((init.model.n_lifetime-
1)*init.model.n_worldHeight):...
                    init.model.n_lifetime*init.model.n_worldHeight,(1 + (1-
1)*init.model.n_worldWidth):1*init.model.n_worldWidth);
                prison =
prisonArray(init.model.n_lifetime+1,1:prisonLengthArray(init.model.n_lifetime+1));
                hospital =
hospitalArray(init.model.n_lifetime+1,1:hospitalLengthArray(init.model.n_lifetime+1));
                [agents,amount] = findAllAgents(world);

                averageSatisfaction = sum([agents.satisfaction]);%alternative way but not
faster
    %             averageSatisfaction = 0;
    %             for index = 1:amount-1
    %                 averageSatisfaction = averageSatisfaction +
agents(index).satisfaction;
    %             end
                averageSatisfaction = averageSatisfaction/amount;
                averageSatisfactionArray(indexMafiaThreshold,indexPoliceThreshold) =
averageSatisfactionArray(indexMafiaThreshold,...
                    indexPoliceThreshold) + averageSatisfaction;
            end
            averageSatisfactionArray(indexMafiaThreshold,indexPoliceThreshold) =
averageSatisfactionArray(indexMafiaThreshold,...
                indexPoliceThreshold)/rCount;

       end
    end
    x = init.model.param_policeThreshold;
    y = init.model.param_mafiaThreshold;
    view(0,90);
    surf(x,y,averageSatisfactionArray);
    colorbar;
    set(gca,'FontSize',14);
    xlabel('PoliceThreshold');
    ylabel('MafiaThreshold');
    zlabel('Satisfaction');
end
```

analyseWorldsizePopulation.m

```
function [] = analyseWorldsizePopulation(init)
%ANALYSEWORLDSIZEPOPULATION
%Compares different worldheight, worldwidth and agents values and makes a 3D plot
    global prison
    global hospital
    averageSatisfactionArray =
zeros(length(init.model.param_agents),length(init.model.param_worldHeight));
    for indexPopulation = 1:length(init.model.param_agents)
       for indexWorldsize = 1:length(init.model.param_worldHeight)
           for rCount=1:init.globals.RUNS
                load(strcat('data/', init.globals.NAME , 'world_PopulationSize_', ...
                    int2str(init.model.param_agents(indexPopulation)),'_', ...
                    int2str(init.model.param_worldHeight(indexWorldsize)),'x', ...

int2str(init.model.param_worldWidth(indexWorldsize)),'_',int2str(rCount)));
                %get the last world
                world = worldArray(1 + ((init.model.n_lifetime-
1)*init.model.param_worldHeight(indexWorldsize)): ...
                    init.model.n_lifetime*init.model.param_worldHeight(indexWorldsize),
...
```

```
                    (1 + (1-
1)*init.model.param_worldWidth(indexWorldsize)):1*init.model.param_worldWidth(indexWorlds
ize));
                prison =
prisonArray(init.model.n_lifetime+1,1:prisonLengthArray(init.model.n_lifetime+1));
                hospital =
hospitalArray(init.model.n_lifetime+1,1:hospitalLengthArray(init.model.n_lifetime+1));
                [agents,amount] = findAllAgents(world);

                averageSatisfaction = 0;
                for index = 1:amount-1
                    averageSatisfaction = averageSatisfaction +
agents(index).satisfaction;
                end

                averageSatisfaction =
averageSatisfaction/init.model.param_agents(indexPopulation);
                averageSatisfactionArray(indexWorldsize,indexPopulation) =
averageSatisfactionArray(indexWorldsize,...
                    indexPopulation) + averageSatisfaction;
            end
            averageSatisfactionArray(indexWorldsize,indexPopulation) =
averageSatisfactionArray(indexWorldsize,...
                indexPopulation)/rCount;

       end
    end
    x = init.model.param_agents;
    y = init.model.param_worldHeight.*init.model.param_worldWidth;
    view(0,90);
    surf(x,y,averageSatisfactionArray);
    colorbar;
    set(gca,'FontSize',14)
    xlabel('Population');
    ylabel('World size');
    zlabel('Satisfaction');
end
```

checkReentry.m

```
function [] = checkReentry(init,world)
%CHECKREENTRY
%counts the jailtime or injury value down, until the agent could be
%released and put back to the world
   global prison;
   global hospital;
   global k;
   prisonlength = length(prison);
   index = 2; %start at index 2 because the first field is empty
   while(index <= prisonlength)
      prison(index).jailtime = prison(index).jailtime - 1;
      if(prison(index).jailtime <= 0)
         %set satisfaction back
         prison(index).person.satisfaction=prison(index).person.satisfaction/ ...
             exp(-init.model.n_jailtime/k);
         reentry(world,prison(index).person);
         prison(index) = []; %delete the agent from prison
         prisonlength = prisonlength - 1;
         if(init.globals.DEBUG)
             disp('released')
         end
      else
         index = index + 1;
      end

   end
   hospitallength = length(hospital);
   index = 2;
   while(index <= hospitallength)
      hospital(index).injury = hospital(index).injury - 1;
      if(hospital(index).injury <= 0)
          %set satisfaction back
          hospital(index).person.satisfaction=hospital(index).person.satisfaction/ ...
              exp(-init.model.n_injury/k);
          reentry(world,hospital(index).person);
          hospital(index)= []; %delete agent from hospital
          hospitallength = hospitallength - 1;
          if(init.globals.DEBUG)
             disp('health')
          end
      else
          index = index + 1;
      end
   end

end
```

copy.m

```
function new = copy(this)
%COPY
%workaround to provide a deep copy
    save('temp.mat', 'this');
    Foo = load('temp.mat');
    new = Foo.this;
    delete('temp.mat');
end
```

createWorld.m

```
function world = createWorld(init)
%CREATEWORLD
%creates a world with the specific size and amount of agents
    height = init.model.n_worldHeight;
    width = init.model.n_worldWidth;
    peopleAmount = init.model.n_agents;
    world(height,width) = location; %allocates memory for the world array
    tempWorld(height,width) = location; %allocates memory for the tempWorld array

    %fill with people
    for index = 0:peopleAmount-1
        p = agent;
        loc=location;
        loc.vision = init.model.n_vision;
        p.place = loc;
        loc.person = p;
        p.initAgent(index+1);
        tempWorld(floor(index/width)+1,mod(index,width)+1) = loc;
    end

    %fill empy agents for empty fields
    for index = peopleAmount:(height*width-1)
        loc=location;
        loc.vision = init.model.n_vision;
        p = agent;
        p.number = 0;
        loc.person = p;
        tempWorld(floor(index/width)+1,mod(index,width)+1) = loc;
    end

    %random distribution of the agents within the world
    size = height*width;
    randPositions = randperm(size);

    for index = 0:size-1
       y = floor((randPositions(index+1)-1)/width)+1;
       x = mod(randPositions(index+1)-1,width)+1;
       world(y,x) =  tempWorld(floor(index/width)+1,mod(index,width)+1);
       world(y,x).x = x;
       world(y,x).y = y;
    end

    %initializes all the Locations and agents with their initial properties.
    initAll(world,init.model.n_vision,init.model.n_jailtime,init.model.n_injury);

    if(init.globals.DEBUG)
        disp('initialisiert');
    end
end
```

displayWorld.m

```
function [] = displayWorld(world)
%DISPLAYWORLD
%displays the current world
    height = length(world(:,1));
    width = length(world(1,:));

    for index = 0:height*width-1
       y = floor(index/width)+1;
       x = mod(index,width)+1;
       if(0 == world(y,x).person.number)
         %plot an empty field
         plot(x,y,'.g');
         hold on
       else
          %plot the agent with a color based on his support
          blue = world(y,x).person.basicSupport;
          red = 1 - world(y,x).person.basicSupport;
          plot(x,y,'-mo',...
            'LineWidth',2,...
            'MarkerEdgeColor','k',...
            'MarkerFaceColor',[red 0 blue],...
            'MarkerSize',10);
          hold on

       end
    end
    set(gca,'FontSize',14)
    xlabel('x-coordinate');
    ylabel('y-coordinate');
    colormap([1 0 0; 0.5 0 0.5; 0 0 1]);
    labels = {'mafia','normal','police'};
    lcolorbar(labels,'fontweight','bold');
    hold off
    drawnow
end
```

findAgents.m

```
function [ agents,finderCounter ] = findAgents( world )
%FINDAGENTS
% Gets a world and gives back all the non-empty agents in it

    width=length(world(1,:));
    heigth=length(world(:,1));
    agents(heigth*width)=agent;     %allocates memory for the agents array
    finderCounter=0;
    for k1=1:width
        for k2=1:heigth
            if(world(k2,k1).person.number~=0)
                finderCounter=finderCounter+1;
                agents(finderCounter)=world(k2,k1).person;
            end
        end
    end
end
```

findAllAgents.m

```
function [ agents,amount ] = findAllAgents( world )
%FINDALLAGENTS
% includes the people in prison and hospital
global hospital
global prison
    [agents,amount]=findAgents(world);  %First get the agents on the world map

    for k=2:length(hospital)    %Get all the non-empty hospital agents
        if(hospital(k).person.number~=0)
            amount=amount+1;
            agents(amount)=hospital(k).person;
        end
    end

    for k=2:length(prison)  %Get all the non-empty prison agents
        if(prison(k).person.number~=0)
            amount=amount+1;
            agents(amount)=prison(k).person;
        end
    end
end
```

getNeighbours.m

```
function [ neighbours,counter ] = getNeighbours( person,world,chooser )
%GETNEIGHBOURS
%gives back an array with all empty (chooser=0) or occupied (chooser=1) fields in the
neighbourhood.

    height = length(world(:,1));
    width = length(world(1,:));
    vision=person.place.vision;
    x=person.place.x;
    y=person.place.y;
    counter=0;
    neighbours((2*vision+1)^2,1)=location;   %allocates space for the Location-Array
    for k1=x-vision:x+vision        %goes throw the column
        for k2=y-vision:y+vision    %goes throw the rows
            if(k1>0 && k2>0 && k1<width && k2<height)
                if((chooser==0 && world(k2,k1).person.number==0)||(chooser==1 &&
world(k2,k1).person.number>0))
                    %if field is in the whished state (empty/occupied): wright it in the
neighbours array and
                    %increase the counter
                    if(k1~=x && k2~=y) %the persons own field shouldn't be added
                        counter=counter+1;
                        neighbours(counter)=world(k2,k1);
                    end
                end
            end
        end
    end
    if(counter == 0)        %if there are no avaible neigbours, the neighbours variable
would not have been defined
        neighbours = 0;
    end
end
```

getStatistics.m

```
function statistics = getStatistics(agents,amount)
%STATISTICS
    %Returns the average values of one round
    % 1. Satisfaction
    % 2. Support
    % 3. RiskM
    % 4. RiskP
    averageSatisfaction = 0;
    for index = 1:amount
        averageSatisfaction = averageSatisfaction + agents(index).satisfaction;
    end
    statistics(1) = averageSatisfaction/amount;

    averageSupport = 0;
    for index = 1:amount
        averageSupport = averageSupport + agents(index).support;
    end
    statistics(2) = averageSupport/amount;

    averageRiskM = 0;
    for index = 1:amount
        averageRiskM = averageRiskM + agents(index).riskM;
    end
    statistics(3) = averageRiskM/amount;

    averageRiskP = 0;
    for index = 1:amount
        averageRiskP = averageRiskP + agents(index).riskP;
    end
    statistics(4) = averageRiskP/amount;
end
```

initAll.m

```
function [] = initAll( world,vision,jailtime,injury )
%INITALL
% initializes first all Locations with their initial properties and then
% all the agents with their NON-CONSTANT properties (the constant ones have
% to be defined already)

    height = length(world(:,1));
    width = length(world(1,:));

    %initialize all Locations and agents
    for k1=1:width
        for k2=1:height
           world(k2,k1).initLocation(k1,k2,jailtime,injury,vision) %initializes the
constant values

           %the person and his constant values on the Location has to be already known

           if(world(k2,k1).person.number~=0)   %an empty field doesn't need to be
initialized
            world(k2,k1).person.place=world(k2,k1); % set the place of the agent to the
place where he's standing
            world(k2,k1).newInfluences(world); %initializes the Influence of Police Mafia
and TOT
```

```
            world(k2,k1).probabilities;        % initializes the probabilites to get hurt
or arrested
            world(k2,k1).person.newRisk;       %set the risks the agent is ready to take
           end
        end
    end

end
```

location.m

```
classdef location < handle
    % the "< handle" assures that when an agent-object is given to a
    % function it is passed by reference and not by value: Therefore the
    % membervariable "place" of an agent object is changed if his location
    % is changed.

    %LOCATION Summary of this class goes here
    %   Detailed explanation goes here

    properties
       x                %Column-number in the Array
       y                %Row-number in the  Array

       %maybe we don't even need the x and y because they're defined by the
       %coordinates in the array.

       jailtime         %how long an arrested agent has to stay in jail
       injury           %how long an injured agent has to stay in the hospital
       vision           %how far the Agent standing here is able to see/interact
                        %(how many grid-cells in eache straight direction)
       person           %the agent standing on this spot

       infMafia         %influence of Mafia in the vision-radius
       infPolice        %influence of the Police in the vision-radius
       infTot           %Sum of the influences of all agents within vision
       % The influences have to be defined from the outside as they depend
       % from oder location-objects
       pArrest        %probability that an Agent is arrested standig here (at the
beginning =0
                      %so that in the first change the pAbefore is set to 0)

       pInjury        %probability that an Agent is injured standig here


    end

    methods
        %Constructor: initializes the constant properties
        %if the field is empty: person.number=-1;
        function initLocation(obj,x,y,jailtime,injury,vision)
            obj.x=x;
            obj.y=y;
            obj.jailtime=jailtime;
            obj.injury=injury;
            obj.vision=vision;
            obj.infMafia=0;
            obj.infPolice=0;
            obj.infTot=0;
            obj.pArrest=0;
            obj.pInjury=0;

        end

        %Updates Injury and Jailtime: the more policemen and the less mafia
        %members there are the less the jailtime will become (is not used)
        function newPenalties(obj,world)
           [nPolice,nMafia] = totalActives(world);
```

```
            obj.pArrest=obj.pArrest + round(nMafia/nPolice - nPolice/nMafia); %round
rounds to integer
            if(obj.jailtime<1)   %doesnt make sense if it is below 1
                obj.jailtime=1;
            end

            obj.pInjury=obj.pInjury + round( - nMafia/nPolice + nPolice/nMafia);
            if(obj.injury<1)   %doesnt make sense if it is below 1
                obj.injury=1;
            end

        end

        %defines the probilites to be arrested and/or injured
        function probabilities(obj)

            if(obj.person.number==0) %empty field not interesting
                return
            end
            %test if agent on the field is properly initialized
            if(obj.person.place~=obj)
                warning('agent not properly initialized: in location.probabilities')
            end
            %adjust the influences
            if(obj.infMafia==0) %would be dividing by 0
               obj.pArrest=(1-obj.person.support);
            else
            obj.pArrest=(1-obj.person.support)*(1-exp(-obj.infPolice/obj.infMafia));
            end
            if(obj.infPolice==0)    %would be dividing by 0
               obj.pInjury=(obj.person.support);
            else
            obj.pInjury=(obj.person.support)*(1-exp(-obj.infMafia/obj.infPolice));
            end
        end

        %defines the new influences of Mafia, Police and Total
        function newInfluences(obj,world)
            global policeThreshold
            global mafiaThreshold
            % the new influences don't depend on the old ones so first set
            % them all to zero
            obj.infMafia=0;
            obj.infPolice=0;
            obj.infTot=0;
            if(obj.person.number==0)    %empty field: no need to update
                return
            end
             %to be safe: if the agent isn't properly defined: give a
             %warning
            if(obj.person.place~=obj)
                warning('agent not properly initialized: in location.newInfluences')
            end

            %get the neighbouring non-empty fields
            [neighbours,amount]=getNeighbours(obj.person,world,1);
            %adds up all the influences
            for k=1:amount
                obj.infTot=obj.infTot+neighbours(k).person.influence;
                if(neighbours(k).person.support>policeThreshold)
                    obj.infPolice=obj.infPolice+neighbours(k).person.influence;
                end
                if(neighbours(k).person.support<mafiaThreshold)
                    obj.infMafia=obj.infMafia+neighbours(k).person.influence;
                end
            end
            if(obj.person.support>policeThreshold)
                obj.infPolice=obj.infPolice + obj.person.influence;
            end
            if(obj.person.support<mafiaThreshold)
```

```
                obj.infMafia=obj.infMafia+obj.person.influence;
            end
        end

        %gets the neighbouring Fields, using getNeighbours on the agent on
        %the field (unnecessary but nice)
        function [neighbours,counter] = neighbours(loc,world,chooser)
            [neighbours,counter] = getNeighbours(loc.person,world,chooser);
        end

    end
end
```

main.m

```
% Main file
% IMPORTANT! Running the whole file at once is not recommended.
% The file is divided in several cells which can executed alone.
% On a multi core prozessor, the whole folder could be copied for each
% core.
% For each core a Matlab instance can be started and run a cell.

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% SIMULATIONS
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Do the simulation Population against Worldsize
% Clear workspace
close all
clear all
clc
% Add other directories to path
path(path,'conf/');
%load the configuration file
conffile
%or load saved configuration
%load('AlphaSim');

%initializes the hospital and prison globally and initializes them as
%locations (otherwise it wouldn't work)
%Beware: Because they are already initialized to a location here, the
%first array-element of both hospital and prison is useless: therefore
%the two arrays will always have one element more then there are people
%in hospital or prison
global prison;
prison=location;
prison(1).x=-2;
prison(1).y=1;
global hospital;
hospital=location;
hospital(1).x=-1;
hospital(1).y=1;

global policeThreshold
policeThreshold = init.model.n_policeThreshold;
global mafiaThreshold
mafiaThreshold = init.model.n_mafiaThreshold;

global k; %constant to calculation a usable satisfaction
k = 10;

global debug;
debug = init.globals.DEBUG; %to have access in class

for indexPopulation = 1:length(init.model.param_agents)
    init.model.n_agents = init.model.param_agents(indexPopulation);
    for indexWorldsize = 1:length(init.model.param_worldHeight)
        fprintf('Population vs Worldsize %d/%d.\n',(indexPopulation-1)* ...
            length(init.model.param_worldHeight) + indexWorldsize, ...
            length(init.model.param_agents)*length(init.model.param_worldHeight));
        init.model.n_worldHeight = init.model.param_worldHeight(indexWorldsize);
        init.model.n_worldWidth = init.model.param_worldWidth(indexWorldsize);

        world = createWorld(init);
        [agent,before]=findAllAgents(world);
        if(init.globals.DEBUG)
            disp(before)
        end

        %preallocate an array to save to world after each round
        worldArray = repmat(world,[(init.model.n_lifetime + 1) init.globals.RUNS]);
        prisonArray = repmat(location,[(init.model.n_lifetime + 1) init.model.n_agents]);
        hospitalArray = repmat(location,[(init.model.n_lifetime + 1)
init.model.n_agents]);
        prisonLengthArray = zeros(init.model.n_lifetime + 1,1);
        hospitalLengthArray = zeros(init.model.n_lifetime + 1,1);
        % Repeat the same simulation RUNS times
        for rCount=1:init.globals.RUNS

            fprintf('Starting Run: %d/%d of Simulation.\n', ...
                    rCount,init.globals.RUNS)
            fprintf('------------------------------------\n');

            %preallocate a statistics array
            statistics = zeros(init.model.n_lifetime,4);
            for index = 1:init.model.n_lifetime;
                %save the current world
                worldArray(1 + ((index-1)*init.model.n_worldHeight):index* ...
                    init.model.n_worldHeight,(1 + (rCount-
1)*init.model.n_worldWidth):rCount* ...
                    init.model.n_worldWidth) = copy(world);%make a deep copy
                prisonArray(index,1:length(prison)) = copy(prison);
                hospitalArray(index,1:length(hospital)) = copy(hospital);
                prisonLengthArray(index) = length(prison);
                hospitalLengthArray(index) = length(hospital);
                %compute statistics
                [agents,amount] = findAllAgents(world);
                statistics(index,:) = getStatistics(agents,amount);
                moveAll(world);
                updateAll(world);
                checkReentry(init,world);
            end

            index = index + 1;
            %save the last world
            worldArray(1 + ((index-1)*init.model.n_worldHeight):index* ...
                init.model.n_worldHeight,(1 + (rCount-1)*init.model.n_worldWidth):rCount*
...
                init.model.n_worldWidth) = world;
            prisonArray(index,1:length(prison)) = copy(prison);
            hospitalArray(index,1:length(hospital)) = copy(hospital);
            statistics(index,:) = getStatistics(agents,amount);

            [agents,amount] = findAllAgents(world);
            if(init.globals.DEBUG)
                disp(amount)
            end

            fprintf('\n\n');
            %save worldarray to disk
            save(strcat('data/', init.globals.NAME , 'world_PopulationSize_', ...
                int2str(init.model.n_agents),'_',int2str(init.model.n_worldHeight),'x',
...
                int2str(init.model.n_worldWidth), '_', int2str(rCount)
,'.mat'),'worldArray', ...

'statistics','prisonArray','hospitalArray','prisonLengthArray','hospitalLengthArray');
            fprintf('Finished Run:\n');
            fprintf('------------------------------------\n');
        end
    end
end

%% Do the simulation Jailtime against Injury

% Clear workspace
close all
clear all
clc
% Add other directories to path
path(path,'conf/');
%load the configuration file
conffile
%or load saved configuration
%load('AlphaSim');

%initializes the hospital and prison globally and initializes them as
%locations (otherwise it wouldn't work)
%Beware: Because they are already initialized to a location here, the
%first array-element of both hospital and prison is useless: therefore
%the two arrays will always have one element more then there are people
%in hospital or prison
global prison;
prison=location;
prison(1).x=-2;
prison(1).y=1;
global hospital;
hospital=location;
hospital(1).x=-1;
hospital(1).y=1;

global policeThreshold
policeThreshold = init.model.n_policeThreshold;
global mafiaThreshold
mafiaThreshold = init.model.n_mafiaThreshold;

global k;
k = 10;

global debug;
debug = init.globals.DEBUG; %to have access in class

for indexJailtime = 1:length(init.model.param_jailtime)
    init.model.n_jailtime = init.model.param_jailtime(indexJailtime);
    for indexInjury = 1:length(init.model.param_injury)
        fprintf('Jailtime vs Injury %d/%d.\n',(indexJailtime-
1)*length(init.model.param_injury) ...
            + indexInjury,
length(init.model.param_jailtime)*length(init.model.param_injury));

        init.model.n_injury = init.model.param_injury(indexInjury);

        world = createWorld(init);
        [agent,before]=findAllAgents(world);
        if(init.globals.DEBUG)
            disp(before)
        end

        %preallocate an array to save to world after each round
        worldArray = repmat(world,[(init.model.n_lifetime + 1) init.globals.RUNS]);
        prisonArray = repmat(location,[(init.model.n_lifetime + 1) init.model.n_agents]);
        hospitalArray = repmat(location,[(init.model.n_lifetime + 1)
init.model.n_agents]);
        prisonLengthArray = zeros(init.model.n_lifetime + 1,1);
        hospitalLengthArray = zeros(init.model.n_lifetime + 1,1);
        % Repeat the same simulation RUNS times
        for rCount=1:init.globals.RUNS
            fprintf('Starting Run: %d/%d of Simulation.\n', ...
                    rCount,init.globals.RUNS)
            fprintf('------------------------------------\n');
            %preallocate a statistics array
            statistics = zeros(init.model.n_lifetime,4);
            for index = 1:init.model.n_lifetime;
                %save the current world
                worldArray(1 + ((index-
1)*init.model.n_worldHeight):index*init.model.n_worldHeight,...
                    (1 + (rCount-
1)*init.model.n_worldWidth):rCount*init.model.n_worldWidth) = copy(world);%make a deep
copy
                prisonArray(index,1:length(prison)) = copy(prison);
                hospitalArray(index,1:length(hospital)) = copy(hospital);
                prisonLengthArray(index) = length(prison);
                hospitalLengthArray(index) = length(hospital);
                [agents,amount] = findAllAgents(world);
                statistics(index,:) = getStatistics(agents,amount);
                moveAll(world);
                updateAll(world);
                checkReentry(init,world);
            end

            index = index + 1;
            %save the last world
            worldArray(1 + ((index-
1)*init.model.n_worldHeight):index*init.model.n_worldHeight, ...
                (1 + (rCount-1)*init.model.n_worldWidth):rCount*init.model.n_worldWidth)
= world;
            prisonArray(index,1:length(prison)) = copy(prison);
            hospitalArray(index,1:length(hospital)) = copy(hospital);
            statistics(index,:) = getStatistics(agents,amount);

            [agents,amount] = findAllAgents(world);
            if(init.globals.DEBUG)
                disp(amount)
            end

            fprintf('\n\n');
            %save worldarray to disk
            save(strcat('data/', init.globals.NAME ,
'world_JailtimeInjury_',int2str(init.model.n_jailtime),...
                '_',int2str(init.model.n_injury), '_',
int2str(rCount),'.mat'),'worldArray','statistics',...
                'prisonArray','hospitalArray','prisonLengthArray','hospitalLengthArray');

            fprintf('Finished Run:');
            fprintf('------------------------------------\n');
        end


    end
end

%% Do the simulation policeThreshold againt mafiaThreshold
% Clear workspace
close all
clear all
clc
% Add other directories to path
path(path,'conf/');
%load the configuration file
conffile
%or load saved configuration
%load('AlphaSim');

%initializes the hospital and prison globally and initializes them as
%locations (otherwise it wouldn't work)
%Beware: Because they are already initialized to a location here, the
%first array-element of both hospital and prison is useless: therefore
%the two arrays will always have one element more then there are people
%in hospital or prison
global prison;
```

```
prison=location;
prison(1).x=-2;
prison(1).y=1;
global hospital;
hospital=location;
hospital(1).x=-1;
hospital(1).y=1;

global policeThreshold
policeThreshold = init.model.n_policeThreshold;
global mafiaThreshold
mafiaThreshold = init.model.n_mafiaThreshold;

global k;
k = 10;

global debug;
debug = init.globals.DEBUG; %to have access in class

for indexPoliceThreshold = 1:length(init.model.param_policeThreshold)
    policeThreshold = init.model.param_policeThreshold(indexPoliceThreshold);
    for indexMafiaThreshold = 1:length(init.model.param_mafiaThreshold)
        fprintf('policeThreshold vs mafiaThreshold %d/%d.\n',(indexPoliceThreshold-1)*...
            length(init.model.param_mafiaThreshold)+indexMafiaThreshold,...

length(init.model.param_policeThreshold)*length(init.model.param_mafiaThreshold));
        mafiaThreshold = init.model.param_mafiaThreshold(indexMafiaThreshold);

        world = createWorld(init);
        [agent,before]=findAllAgents(world);
        if(init.globals.DEBUG)
            disp(before)
        end

        %preallocate an array to save to world after each round
        worldArray = repmat(world,[(init.model.n_lifetime + 1) init.globals.RUNS]);
        prisonArray = repmat(location,[(init.model.n_lifetime + 1) init.model.n_agents]);
        hospitalArray = repmat(location,[(init.model.n_lifetime + 1)
init.model.n_agents]);
        prisonLengthArray = zeros(init.model.n_lifetime + 1,1);
        hospitalLengthArray = zeros(init.model.n_lifetime + 1,1);
        % Repeat the same simulation RUNS times
        for rCount=1:init.globals.RUNS

            fprintf('Starting Run: %d/%d of Simulation.\n', ...
                    rCount,init.globals.RUNS)
            fprintf('------------------------------------\n');

            %preallocate a statistics array
            statistics = zeros(init.model.n_lifetime,4);
            for index = 1:init.model.n_lifetime;
                %save the current world
                worldArray(1 + ((index-1)*init.model.n_worldHeight):index*...
                    init.model.n_worldHeight,(1 + (rCount-
1)*init.model.n_worldWidth):rCount*...
                    init.model.n_worldWidth) = copy(world);%make a deep copy
                prisonArray(index,1:length(prison)) = copy(prison);
                hospitalArray(index,1:length(hospital)) = copy(hospital);
                prisonLengthArray(index) = length(prison);
                hospitalLengthArray(index) = length(hospital);
                [agents,amount] = findAllAgents(world);
                statistics(index,:) = getStatistics(agents,amount);
                moveAll(world);
                updateAll(world);
                checkReentry(init,world);
            end

            index = index + 1;
            %save the last world
            worldArray(1 + ((index-
1)*init.model.n_worldHeight):index*init.model.n_worldHeight,...
```

```
                (1 + (rCount-1)*init.model.n_worldWidth):rCount*init.model.n_worldWidth)
= world;
            prisonArray(index,1:length(prison)) = copy(prison);
            hospitalArray(index,1:length(hospital)) = copy(hospital);
            statistics(index,:) = getStatistics(agents,amount);

            [agents,amount] = findAllAgents(world);
            if(init.globals.DEBUG)
                disp(amount)
            end

            fprintf('\n\n');
            %save worldarray to disk
            save(strcat('data/', init.globals.NAME
,'world_PoliceMafiaThreshold_',num2str(policeThreshold),...
                '_',num2str(mafiaThreshold), '_',
int2str(rCount),'.mat'),'worldArray','statistics','prisonArray',...
                'hospitalArray','prisonLengthArray','hospitalLengthArray');
            fprintf('Finished Run:');
            fprintf('------------------------------------\n');

        end
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Livetime Analysation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% show World

% Clear workspace
close all
clear all

% Add other directories to path
path(path,'conf/');
%load the configuration file
conffile
%load saved world data, the name should be insert by hand
%load(strcat('data/', init.globals.NAME , 'world_JailtimeInjury_','1','_','1','_1'));
%load(strcat('data/','Christianworld_PopulationSize_30_10x10_1'));
load(strcat('data/Christianworld_JailtimeInjury_10_6_1'));
worldHeight = 10;
worldWidth = 10;
rCount = 1;
pause on
for index = 1:(length(worldArray(:,1))/(worldHeight));
    index
    world = worldArray(1 + ((index-1)*worldHeight):index*...
        worldHeight,(1 + (rCount-1)*worldWidth):rCount*worldWidth);
    displayWorld(world);
    pause(0.1)
end

%% show statistic

% Clear workspace
close all
clear all
clc
% Add other directories to path
path(path,'conf/');
%load the configuration file
conffile
%load saved world data, the name should be insert by hand
%load(strcat('data/','Christianworld_PopulationSize_30_10x10_1'));
%load(strcat('data/Christianworld_JailtimeInjury_5_5_1'));
load(strcat('data/','Christianworld_PoliceMafiaThreshold_0.5_0.5_1.mat'));
```

```
plotStatistics(statistics,(length(statistics(:,1))));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End Analysation
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% world size & population -> satisfaction

% Clear workspace
close all
clear all
clc

global prison
global hospital
% Add other directories to path
path(path,'conf/');
%load the configuration file
conffile
%load saved world data
analyseWorldsizePopulation(init);

%% jailtime & injury -> satisfaction

% Clear workspace
close all
clear all
clc

global prison
global hospital
% Add other directories to path
path(path,'conf/');
%load the configuration file
conffile
%load saved world data
analyseJailtimeInjury(init);

%% policeThreshold & mafiaThreshold -> satisfaction

% Clear workspace
close all
clear all
clc

global prison
global hospital
% Add other directories to path
path(path,'conf/');
%load the configuration file
conffile
%load saved world data
analysePoliceThresholdMafiaThreshold(init);
```

moveAll.m

```
function world = moveAll(world)
%MOVEALL
%goes throw all agents and move them
    [agents,finderCounter] = findAgents(world);
    %random numbers, ever number appears only once
    randIndex = randperm(finderCounter);
    for index = 1:finderCounter
        movePerson(agents(randIndex(index)),world);
    end
end
```

movePerson.m

```
function [ person,world ] = movePerson( person,world)
%MOVEPERSON
% Moves a Person onto a field in his/her vision
global policeThreshold
global mafiaThreshold
global debug


    [neighbours,counter]=getNeighbours(person, world,0); %gets the empty, neighbouring 
fields and there number

    empty=agent;                %creates agent
    empty.initAgent(0);         %defines it as empty

    if(counter>0)   %The agent can only move if there are empty fields in his 
neighbourhood
                    %(but even if there arent he still can arrest or hurt people so don't 
just abbort)

        %the person can just stay put and not move at all: "moves to his
        %own field"
        if((person.support > policeThreshold) || (person.support < mafiaThreshold))

            neighbours(counter+1)=person.place;
            counter=counter+1;

            index=randi(counter,1);     %creates a random integer between 1 and the 
number of free neighbours
            if(index~=counter)              %if index==counter: person doesn't move at 
all
                x=neighbours(index).x;      %gets the column of the location in the world
                y=neighbours(index).y;      %gets line of the location in the world
                person.moveTo(world(y,x));  %moves the agent to his new location and sets 
all the member variables
            end
        else    %normal population
                %normal people try to avoid both police and mafia: go where
                %their influence is as small as possible. For people who
                %are rather mafia supportive the police influence is worse
                %than the mafia influence and vice versa
                minInf = 1000;
                indexMin = 1;
                for index = 1:counter
                    if((1-person.support)*neighbours(index).infPolice + 
person.support*neighbours(index).infMafia < minInf)
                       minInf = (1-person.support)*neighbours(index).infPolice + 
person.support*neighbours(index).infMafia;
                       indexMin = index;
                    end
                end
                x=neighbours(indexMin).x;
                y=neighbours(indexMin).y;

                %Insert a random element: sometimes the population just
                %moves randomly
                tester=rand;
                if(tester<0.1)
                    index=randi(counter,1);     %creates a random integer between 1 and 
the number of free neighbours
                    if(index~=counter)              %if index==counter: person doesn't 
move at all
                        x=neighbours(index).x;      %gets the column of the location in 
the world
                        y=neighbours(index).y;      %gets line of the location in the 
world
                    end
                end
                person.moveTo(world(y,x));

        end


    end

    if(person.support>policeThreshold)     %If the agent is an active policeman
        [neighboursA,counterA]=getNeighbours(person, world,1); %gets the occupied
neighbouring fields

        if(counterA==0)      %can only arrest if there are people
            return
        end

        index=randi(counterA,1); %random integer
        x=neighboursA(index).x;  %the field to check
        y=neighboursA(index).y;

        comp=rand;                   %random number between 0 and 1
        if(comp<(world(y,x).pArrest))    %if comp<pArrest: agent is arrested: his place
is set to 0,
                                         %the location where he was standing is now empty
            world(y,x).person.toPrison();
            if(debug)
             disp('arrested')
          end
        end
    end

    if(person.support<mafiaThreshold)     %If the agent is an active mafia supporter
        [neighboursI,counterI]=getNeighbours(person, world,1); %gets the occupied
neighbouring fields

        if(counterI==0)      %can only injure if there are people
            return
        end

        index=randi(counterI,1); %random integer
        x=neighboursI(index).x;  %the field to check
        y=neighboursI(index).y;
        comp=rand;                   %random number between 0 and 1
        if(comp<world(y,x).pInjury)    %if comp<pArrest: agent is arrested: his place is
set to 0,
                                       %the location where he was standing is now empty
            world(y,x).person.toHospital();
            if(debug)
             disp('hurt')
          end
        end
    end

end
```

plotStatistics.m

```
function [] = plotStatistics(statistics,cycles)
%PLOTSTATISTICS
    figure
    plot(1:cycles,statistics);
    set(gca,'FontSize',14);
    legend('Satisfaction','Support','RiskM', 'RiskP');
    xlabel('lifecycle');
end
```

randomvalue.m

```
function [ number ] = randomvalue(n)
%RANDOMVALUE
% gives back a pseudorandom number between 0 and 1 with mean 0.5 and
% standard deviation 0.12 like that the safety protocol that
% sets negative numbers to zero and numbers >1 to 1 is almost never used.
% The +0.5 sets the mean from zero to 0.5 and the *0.12 sets the variance to (0.12)^2

   number=0.12*randn(n,1)+0.5;

   if(number<0)
       number=0;
   end
   if(number>1)
       number=1;
   end

end
```

reentry.m

```
function [] = reentry(world,person)
%REENTRY
%set the agent on a free place in the world
    height = length(world(:,1));
    width = length(world(1,:));
    %find empty place to reentry
    y = floor(rand(1)*height) + 1;
    x = floor(rand(1)*width) + 1;
    while(0 ~= world(y,x).person.number)
        y = floor(rand(1)*height) + 1;
        x = floor(rand(1)*width) + 1;
    end
    %reentry
    world(y,x).person = person;
    person.place = world(y,x);

end
```

totalActives.m

```
function [ policemen,mafiosi ] = totalActives( world )
%TOTALACTIVES
global policeThreshold
global madiaThreshold
    mafiosi=0;
    policemen=0;
    [agents,counter]=findAgents(world);
    for k=1:counter
        if(agents(k).support>policeThreshold)
            policemen=policemen+1;
        end
        if (agents(k).support<madiaThreshold)
              mafiosi=mafiosi+1;
        end
    end

end
```

updateAll.m

```
function [  ] = updateAll( world )
%UPDATEALL
% updates all the Locations with their new values of Influence and
% Probabilities (the new agents have already been set by movePerson) and
% all the agents with their new values of Satisfaction, support and Risks
% (the new places have already been set by movePerson)


    height = length(world(:,1));
    width = length(world(1,:));

    % update Locations and agents
    for k1=1:width
        for k2=1:height
           if(world(k2,k1).person.number~=0) %if not an empty field

               %for precaution: if the agent's place isn't where he really
               %is: warning
            if(world(k2,k1).person.place~=world(k2,k1))
                warning('agent not properly initialized: in updateAll')
            end
            world(k2,k1).newInfluences(world);  % updates the Influence of Police Mafia
and TOT
            world(k2,k1).probabilities;         % updates the probabilites to get hurt or
arrested

            world(k2,k1).person.newSat;      %set the satisfaction
            world(k2,k1).person.newRisk;     %set the risks the agent is ready to take
            world(k2,k1).person.newSup;      %set the new support
           end
        end
    end
end
```