23

Biophysical Modeling using Cellular Automata

Bernhard Pfeifer
University for Medical Informatics and Technology Tyrol
Austria

1. Introduction

The definition of a model is as following: *a model is an abstract and simplified picture of a reality in the world*. From this definition it can be derived that each biomedical model is an approximation of an organism, organ, tissue, or cell. The reason for the need of models is, that the originals are much too complex to be described and understood completely.
Let us assume that an individual has to be completely modeled in a computer system, one has to calculate the information content. This individual should consist of 60 different atoms. Those are defined to their position with accuracy of $\pm 2 \cdot 10^{-12}[m]$. The composition of the atoms is neglected here. Furthermore, the dimension of the individual is given by $2[m] \cdot 0,5[m] \cdot 0,4[m]$. The simplification of position, the neglecting of composition, and the cubic form of the individual is of course a simplification, but let forget about this fact in this example.
Claude Shannon, the founder of the information theory Shannon (1948), defined the entropy **H** of a given information **I** over an alphabet **Z** by

$$H(I) = -\sum_{j=1}^{|Z|} p_j \cdot ld(p_j), \tag{1}$$

where p_j is the likelihood that the j^{th} symbol z_j of the alphabet **Z** appears in the given information text **I**. The unit of entropy is bit. When multiplying H with the number of characters given in the alphabet, the minimum necessary number of bits for representing the information is calculated.
The information content (entropy) of the above given example is calculated as $H = ld(60) = 5,9[bit]$ (Eq. 1). The number of different atom positions in the individual is $V/Vd + 1 = 0.4^3/(64 \cdot 10^{-36}[m^3]) + 1 = 625 \cdot 10^{31} + 1$ [atoms]. Having this it is possible to compute the necessary memory for storing the model: $625 \cdot 10^{31}[atoms] \cdot 5.9[bit/atom] = 3,68 \cdot 10^{34}$. No computer exists that is able to store and handle this model in a finite time.
On account, this simple example shows that using approximations of the reality is necessary to address with complexity for understanding life. Complex problems have to be split into treatable parts, which can then be implemented using modern computers. Such a proceeding is widely used in computer science e.g. when developing algorithms divide and conquer technique are applied for simplifying the problem. This allows computing partial solutions that are then combined to obtain the overall solution.
As it is inevitable to have approximations as models, it seems to be imperative to define the necessary details in the model. Bossel defined a model following: *the scope of the model is the*

key for any model development; its precise specification enables a clear and compact formulation of the model Bossel (1992); Fischer (2006).

2. Modeling techniques

Numerical simulation is typically based on continuous models. Problems, which consider independent variables e.g. spatial elements and time can be modeled using partial differential equations (PDE), and problem without considering more independent variables can be formulated using ordinary differential equations (ODE). Furthermore, cellular automata can be used for modeling spatiotemporal problems.

2.1 Ordinary differntial equations

As one example the Lotka-Volterra-Equation Volterra (1926), which is also know as Predator-Prey-Equation, is a mathematical system based on coupled differential equations, which describe the dynamics of predator and prey populations. The equations are based on three rules, which were constructed during World War II on observations of the fish stock in the Adria. The first rule describes the periodical fluctuations of a population. The fluctuations of the predator-population are phase-delayed compared to the prey-population. The second rule describes the stability of the mean value. Although there are periodical fluctuations, the number of the populations is constant. The last rule describes that the prey-population is growing faster than the predator-population. If predator- and prey-population becomes decimated for a defined period, then the prey-population recovers always faster than the predator-population.
This situation can be mathematical modeled using ordinary differential equations:

$$\frac{dx}{dt} = x(\alpha - \beta y) \tag{2}$$

$$\frac{dy}{dt} = y(\delta x - \gamma), \tag{3}$$

where y is the number of predator-population, x ist the number of prey-population, t represents the time, $\frac{dx}{dt}$ and $\frac{dy}{dt}$ represents the growth of the given populations against time. α is the exponential growing rate of the prey population, β is the predator-capture-rate, δ is the reproduction rate of the predator population, and γ the natural predator death rate.
Models of this type are of importance in theoretical biology, and in epidemiology e.g. for describing the processes of disease spreading.
The disadvantage is, that the result describes the behavior of the whole population without having the chance to look at each individual directly. Another point is, that it is impossible to model topological structures, which may get important for some simulation cases.

2.2 Partial differential equations

As described, using ordinary differential equations enables only to model one dimension. Therefore, in the Predator-Prey-Model the overall behaviors over the time can be modeled and expressed. Partial differential equations, however, consider change of state along several dimensions.
The Predator-Prey-Model can be extended using PDEÕs for modeling spatial variations. These variations are that a predator has to move in order to catch a prey, and a prey is able to move for evading a predator. The model can be described as following:

$$u_t(r,t) = \varphi\nabla^2 u(r,t) + \frac{\alpha}{b}u(b-u) - \gamma\frac{uv}{u+h} \tag{4}$$

$$v_t(r,t) = \psi\nabla^2 v(r,t) + \kappa\gamma\frac{uv}{u+h} - \mu v. \tag{5}$$

The first equation describes the prey population, and the second equation describes the predator population. $\frac{\alpha}{b}u(b-u)$ describes the local growth and mortality of the prey, $\gamma\frac{uv}{u+h}$ describes the predation, κ is the food utilization, and μ the mortality rate of the predator. The parameter α is the maximal growth rate of the prey, b is the carrying capacity of the prey population, and h ist the half-saturation abundance of prey. φ and ψ are the diffusion coefficients.

2.3 Spatiotemporal cellular automata modeling

Cellular automata John von Neumann (1966) are discrete dynamical systems, which allow designing spatiotemporal models based on cell-cell, cell-medium and cell-medium-cell interactions. Cellular models have been introduced by John von Neumann and Stanislaw Ulam as computer models for self-reproduction. The constructed automaton consisted of 29 different states per cell. In the 60ies John Horton Conway created the well known zero player game ÒGame of LifeÓ, which is based on a cellular automaton. Stephen WolframÕs book ÒA New Kind of ScienceÓ Stephen Wolfram (2002) in the year 2002 tried to show how powerful cellular automata are, and that they can be used for modeling in sciences. Based on this book, the public interest increased rapidly and many research laboratories used cellular automaton models for simulating the dynamics of spatiotemporal problems.

2.3.1 Concept behind cellular automata

Cellular automaton models consists of several interacting cells, each of which having a state transition function inside, which brings a cell state at time point t in another state for time point $t+1$. Therefore, a CA model can be anatomized in simple finite state machines, or finite automata.

2.3.1.1 Finite automata

are models for computers for devices with limited resources. A vending machine, but even a computer system we are used to work with comes into this category. Memory and computational resources are limited, but in spite of that, one is able to perform complex operations and simulation using those machines. A deterministic finite automaton can be defined as following

$$A = (Q, \Sigma, \delta, q_0, F), \tag{6}$$

where Q describes the set of states, Σ describes the input alphabet, δ is the state transition function that is defined as $\delta : Q \times \Sigma \to Q$. q_0 is the start state with $q_0 \in Q$. and F represents the set of accepting or final states with $F \subseteq Q$.

2.3.1.2 Finite automata interplay

A cellular automaton consists of a quantum of finite automata, which are collaborating. This collaboration is defined in the state transition function δ where the new state does not only depend on the actual cell states but also from the neighboring cell states. Furthermore, one has to decide, which cells should collaborate. Therefore, the cell adjustment and the neighborship

have to be defined. As each cell also has an output function, the definition of a cellular automaton can be written as follows

$$CA = (C, N, \Sigma, Y, Q, \delta, \sigma, q_0, F), \tag{7}$$

where C is the cell adjustment, N defines the neighborhood, Σ is the input alphabet, Y the output alphabet, and Q describes the set of states. δ is the state transition function with $\delta : Q \times Q^{n_i} \rightarrow Q$ with $n_i \in N$. σ is the output function with $\sigma : Q \rightarrow A$. q_0 is the start state with $q_0 \in Q$, and F represents the set of accepting or final states with $F \subseteq Q$.
As a cellular automaton has a cellular space or lattice, these models allow the visualization of the automaton states at each time point. The regular lattice $L \subseteq R^d$ consists of several individual cells, which interact using a neighborhood relation. The interaction neighborhood can be defined as

$$N_b^I(r) := \{r + c_i | c_i \in N_b^I\} \in L, \tag{8}$$

where N is the interaction neighborhood template, b is the coordinate number, r is the position of the cell and c_i denotes the interacting neighbors.
In two dimensions the only regular polygons forming a regular tesselation are triangles, rectangles and hexagons NY (1977). The neighborhood is defined as:

$$N_b = \{c_i, i = 1, ..., b : c_i = (cos(\frac{2\pi(i-1)}{b}), sin(\frac{2\pi(i-1)}{b}))\}. \tag{9}$$

3. Disease spread modeling using cellular automaton approach

3.1 History

In the year 431 before Christ, Thukydides noted down about a tragedy that devastated Athens citizens. The symptoms were dramatic. Young, healthy adults suddenly came down with an unexplainable disease. This outbreak was the start of an era where epidemics were recorded. Up to the 20ths century contagious diseases ran rampant and often incurable, which dramatically reduced the population in case of an outbreak. From the scores of epidemics occurred in the history, some of them have a special impact up to now: The outbreak came nearly two millennia after the outbreak in Athens, which brought death and bane over the world. Medical historians discovered that the plague occurred in the year 1331 in China, and decimated the population by 50 percent. Over existing trade routes the plague reached Krim in the year 1346, and from this hub Europe, Northern Africa and the Middle East. The name of this disease became the embodiment of horror: Black Death. At that time the disease and the infection was mysterious, but today we know that a bacterium named Yersinia pestis spread using flea living on black rats, infected individuals, and killed between 1347 and 1351 a third of the Europeans back then. Above all the plague bacterium can be transmitted from an infected individual to a healthy individual, which is known as airborne infection. The outbreak changed the behavior of the population in various ways. Some kept themselves away from remaining population to prevent from population contact, while others started to live an extensive life. The next outbreak of the plague appeared in 1896, and spread to nearly every part of the globe. By 1945, the death toll reached approximately 12 million.
Between 1918 and 1920 the Spanish Flu pandemic killed about 20-50 million people, especially young adults and teens with well working immune systems. No infection, no war, and no famine have ever had claimed that much victims in a little while. Surprisingly, the outbreak of the Spanish Flu had no evident impact, because in the heads the scare of the war was present, and nobody wanted to write about this epidemic.

An outbreak of the Asian Flu in 1957 resulted in an estimate of one million deaths. The Hong Kong Flu killed a population of about 700.000 individuals. AIDS, caused by the human immunodeficiency virus (HIV), was first recognized in the 1980s, and it has killed over 20 million people until now. This disease is now a pandemic, with an estimate of more than 40 million infected individuals at present. Apparently there are several factors, which perpetuate the spread of AIDS and other infectious diseases, including incautiousness (both sexually and drug abuse), misconceptions of the transmission and the immense belief in the development of modern medicine. It is worth pointing out in this context that about 90 percent of the death from infectious diseases worldwide is caused by only a few of diseases.

Most contagious diseases can be modeled using mathematical approaches to analyze and understand the epidemiological behavior or for predicting the process. Therefore, different approaches have been developed in the past. The classic S-I-R epidemic model, where class S denotes the number of susceptibles, class I denotes the number of invectives and class R denotes the number of recovered individuals. The sum of the given initial value problem is S(t)+I(t)+R(t)=N, with N being the number of observed population. However, the SIR model is not adequate to model natural birth and death, immigration and emigration, passive immunity and spatial arrangement adequately. To model infection diffusion through space, partial differential equations (PDE) are needed. With PDE models it is possible to simulate the spreading of a disease over a population in space and time. However, the integration of geographical conditions, demographic realities, and keeping track over each individual is impossible. For this purpose, cellular automaton (CA) models can be used. A CA model is a dynamical system in which time and space is discrete and is specified by a regular discrete lattice of cells and boundary conditions, a finite set of cells and states, a defined neighborhood relation, and a state transition function that is responsible for computing the dynamics of the cells over the time.

For this purpose cellular automaton (CA) models can be used. A CA model is a dynamical system in which time and space is discrete and is specified by a regular discrete lattice of cells and boundary conditions, a finite set of cells and states, a defined neighborhood relation, and a state transition function that is responsible for computing the dynamics of the cells over the time. CA models for highly dynamic disease spread simulation are widely known Beauchemin et al. (2004); Castiglione et al. (2007); Xiao et al. (2006) and shape-space interactions were introduced for enabling to simulate complex interacting systems. Dynamic bipartite graphs for modeling physical contact patterns were introduced, which resulted in more precisely modeling of individualsÕ movements. The graph can be built on actual census and available demographic data. When analyzing those graphs, the existing hubs can be found easily. It could be figured out that by using strategies like targeted vaccination combined with early detection without resorting to mass vaccination of a population an outbreak could be contained Eubank et al. (2004). The simulation application EpiSims Barrett et al. (2005), which has been developed at Los Alamos allows simulating different scenarios by modeling the interaction of the different individuals participating in the simulation. The knowledge about the paths enables to perform arrangements like quarantine or targeted vaccination to prevent the disease from further spreading. The model EpiSims was a reproduction of the city Portland (Oregon), but not a facsimile, because to model the habits of about 1,6 million individuals would be nearly impossible and furthermore a massive intrusion into privacy. EpiSims allows to set parameter values for the within-host disease model on demographics of each person, but also simulating the introduction of counter-measures such as quarantine, vaccination or antibiotic use can be done. The human mobility information

is derived from the TRANSIMS model, which estimates the movement of people based on census data and activity maps taken from defined samples of the population. Using this specific information Òsocial networkÓ can be modeled for understanding how epidemiology depends on those characteristics, and furthermore the calculation of the overall economic pecuniary impact is possible.

3.2 Generic disease spread modeling framework

3.2.1 The class `StepResult`

The Class `Stepresult` stores the computed parameters at time point t to generate statistics and a snapshot of each individual time point.

```
 1 package Pandemie;
 2
 3 public class StepResult {
 4     private static StepResult instance;
 5
 6    protected long passiveimmunityfrombirth;
 7    protected long susceptible;
 8    protected long infective;
 9    protected long recovered;
10    protected long killedbydisease;
11
12    protected long spontanous;
13    protected long vectored;
14    protected long contact;
15
16    protected long individuals;
17
18    protected long died;
19    protected long born;
20
21    protected long moved;
22    protected long immigrant;
23
24    protected long healty;
25    protected long latent;
26    protected long infectious;
27    protected long removed;
28
29    protected long lastDied;
30    protected long lastKilledDisease;
```

In the attributes (line 4 to 30) the basic computed disease and state values are stored. For accessing these attributes `get` methods are implemented.

```
31    public static synchronized StepResult getInstance() {
32         if (instance == null) {
33           instance = new StepResult();
34         }
35         return instance;
36       }
37
38       protected StepResult() {
39       }
40 }
```

Since the step results need to be accessible in different objects like a global attributes, a singleton pattern Gamma (1994) is used. The instance can be accessed by calling the static `getInstance()` method, which is able to access the protected constructor. Furthermore this method must be synchronized in case multithreading is used to guarantee data consistency.

3.3 The class `InfectionParameters`

The attributes that are managed by this class describe the disease, demographic and action parameters. The initialized values are one set with which it is possible to simulate an avian flu that is highly infective and has a high death rate. Furthermore, parameters for quarantine and medication can be set for simulating different scenarios.

```
1 package Pandemie;
2
3 import java.util.Random;
4
5 public class InfectionParameters {
6    private static InfectionParameters instance;
7
8    protected int latentPeriodDays = 3;
9    protected int infectousPeriodDays = 10;
10   protected int recoveredRemovedAfterDays = 15;
11   protected int incubationPeriodDays = 3;
12   protected int symptomaticPeriodDays = 4;
13
14   protected double birthrate = 0.002d;
15   protected double deathrate = 0.001d;
16
17   protected double virus_morbidity_percent = 0.63d;
18
19   protected double spontaneous_infection_rate = 0.000001d;
20
21   protected double vectored_infection_rate = 0.35d;
22   protected double contact_infection_rate = 0.45d;
23
24   protected double movement_probability = 0.4d;
25
26   protected double immigrantrate = 0.0000001;
27
28   protected boolean useQuarantine = false;
29
30   protected boolean handleMedication = false;
31   protected double medicationOne = 3.5d;
32   protected double medicationTwo = 5.5d;
33
34   protected int suspectibe_again_after_recover = 100;
35   protected int birthimmunityindays = 20;
36
37   protected long maxCellCapacity = 500;
38
39   protected static Random randomGenerator;
40
41   public static synchronized InfectionParameters getInstance() {
42      if (instance == null) {
43         randomGenerator = new Random();
44         instance = new InfectionParameters();
45      }
```

```
46         return instance;
47     }
48
49     protected InfectionParameters() {
50     }
51 }
```

As the access to these parameters is needed by many objects it is also implemented using the singleton pattern.

3.3.1 The class `DiseaseCell`

The `DiseaseCell` class represents one cell of the CA model, in which the individualsÕ takes place and interact among defined rules. The state transition function δ is inherited from the super class. This function is responsible for calculating the spreading, and how infected individuals have to be treated. Therefore, this function calculates death caught by the disease, followed by adding newborns and removing natural death cases. Then, immigrants and emigrants are estimated, the vectored, contact, and the spontaneous infection is computed and in the last step the individual movement is performed.

```
1 package Pandemie;
2
3 public class DiseaseCell extends Cell {
4     private ArrayList<CellIndividual> individuals;
```

In line 4 a dynamical data structure for storing the individuals that take place in the cell is introduced.

```
5     public DiseaseCell(int numberOfIndividuals) {
6         individuals = new ArrayList<CellIndividual>();
7
8         for (int i = 0; i <= numberOfIndividuals - 1; i++) {
9             individuals.add(new CellIndividual());
10        }
11    }
```

When a cell object is instantiated the constructor creates the dynamical data structure that holds the individuals. Furthermore, the number of individuals is generated in the loop and stored using the data structure.

```
12    public boolean performCellAction(CellularAutomaton ca) {
13        this.handleNaturalDeath();
14        this.handleNewborns();
15
16        this.handleImmigrants();
17        this.handleSpontanousInfections();
18
19        this.handleContactInfections(ca);
20        this.handleVectoredInfections();
21
22        this.updateCellIndividuals();
23
24        this.handleMovement(ca);
25        return true;
26    }
```

The state transition function is the main simulation component of the modeling framework. In this implemented model the functions for computing death, birth, spontaneous infections, immigrants, vectored infection contact infection, and individual movement are executed.

```
27   public void updateCellIndividuals() {
28      for (Iterator individualIterator = individuals.iterator();
29         individualIterator.hasNext();) {
30         CellIndividual
31            individual = (CellIndividual) individualIterator.next();
32
33         individual.updateIndividual();
34      }
35   }
```

The `updateIndividual()` method is called in order to initialize the models data for performing the subsequent simulation step over time *t*.

```
37   public void handleNewborns() {
38      InfectionParameters szParam = InfectionParameters.getInstance();
39      StepResult res = StepResult.getInstance();
40
41      ArrayList<CellIndividual> addIndividuals = new ArrayList<
42            CellIndividual>();
43
44      for (Iterator individualIterator = individuals.iterator();
45         individualIterator.hasNext();) {
46         CellIndividual individual = (CellIndividual) individualIterator.
47               next();
48
49         if ((individual.getAgeType() == AgeType.ADULT)
50               || (individual.getAgeType() == AgeType.TEEN)) {
51            if (isTheCase(szParam.getBirthrate())) {
52               CellIndividual newborn = new CellIndividual();
53               newborn.setAgeType(AgeType.KID);
54                newborn.setSusceptibleInDays(szParam.
55                         getBirthimmunityindays());
56
57               if (this.isTheCase(0.7d)) {
58                   newborn.setStateType(StateType.PASSIVEIMMUNEFROMBIRTH);
59               }
60
61               addIndividuals.add(newborn);
62               res.setBorn(res.getBorn() + 1);
63            }
64         }
65      }
66      if (addIndividuals != null)
67         individuals.addAll(addIndividuals);
68   }
69
70   public void handleNaturalDeath() {
71      InfectionParameters szParam = InfectionParameters.getInstance();
72
73      for (Iterator individualIterator = individuals.iterator();
74         individualIterator.hasNext();) {
75         CellIndividual individual = (CellIndividual) individualIterator.
```

```
76                  next();
77
78          if (individual.getStateType() == StateType.DIED) continue;
79          if (individual.getStateType() == StateType.KILLEDBYDISEASE)
80                  continue;
81
82          if (this.isTheCase(szParam.getDeathrate())) {
83              if ((individual.getAgeType() == AgeType.KID)
84                  || (individual.getAgeType() == AgeType.TEEN)
85                  || (individual.getAgeType() == AgeType.ADULT)) {
86                  if (this.isTheCase(0.7d)) {
87                      individual.setStateType(StateType.DIED);
88                  }
89              } else {
90                  individual.setStateType(StateType.DIED);
91              }
92          }
93      }
94  }
```

Both methods `handleNewborns()` and `handleNaturalDeath()` implements the natural growing and shrinking of a population caused by defined birth and death parameters. When an individual gets is born a temporary immunity is applied, which protects the individual from becoming ill by the spreading disease. Furthermore, in this model it is only possible for adults to get children, which is accordable with natural behavior. During the computation of natural death cases a stochastic function is used, which gives the different age classes (kids, teen, adult, elderly) a different likelihood of dieing.

```
95  public void handleImmigrants () {
96      InfectionParameters szParam = InfectionParameters.getInstance();
97      StepResult res = StepResult.getInstance();
98
99      if (this.isTheCase(szParam.getImmigrantrate())) {
100         CellIndividual immigrant = new CellIndividual ();
101         if (immigrant.getAgeType() == AgeType.KID)
102             immigrant.setAgeType(AgeType.ADULT);
103
104          individuals.add(immigrant);
105          res.setImmigrant(res.getImmigrant() + 1);
106     }
107 }
```

Defined by the immigration rate parameter the probability of a new immigrant is computed. If the function returns that a new immigrant is allowed to enter the simulation then the immigrant is added to the cell as new member. Furthermore, there is a restriction that only adults and elderly people are allowed to enter. If a individual not being part of this age type tries to enter, then the age class is adapted in order to fulfill the requirements.

```
108 protected void handleNeighborCellInfections (
109     CellularAutomaton ca, InfectionParameters szParam,
110     StepResult res, double probability) {
111     DiseaseCell regSZNeighbourCell;
112
113     for (Iterator individualIterator = individuals.iterator();
```

```
114         individualIterator.hasNext();) {
115         CellIndividual individual = (CellIndividual) individualIterator.
116                 next();
117
118         if (szParam.isUseQuarantine() &&
119            (individual.getQuarantineType() == QuarantineType.QUARANTINE))
120            continue;
121
122         for (Iterator it = neighbourCellIndexList.iterator();
123            it.hasNext();) {
124            Long element = (Long) it.next();
125            try {
126               regSZNeighbourCell = (DiseaseCell)
127                  ca.getCell(element);
128            } catch (Exception e) {
129               continue;
130            }
131
132            for (Iterator adjacentIndividual = regSZNeighbourCell.
133                getIndividuals().iterator();
134               adjacentIndividual.hasNext();) {
135               CellIndividual adjacent = (CellIndividual)
136                     adjacentIndividual.next();
137
138               if (szParam.isUseQuarantine() &&
139                  (individual.getQuarantineType() == QuarantineType.
140                    QUARANTINE))
141                  continue;
142
143
144               if (individual.getStateType() == StateType.INFECTIVE) {
145                  switch (adjacent.getStateType()) {
146                  case SUSCEPTIBLE:
147                     boolean infection = this.isTheCase(probability);
148                     if (adjacent.getSusceptibleInDays() > 0) infection =
149                           false;
150                     if (individual.getDiseaseCycle() ==     DiseaseCycle.
151                           LATENT)
152                        infection = false;
153
154                     if (infection) {
155                        adjacent.setStateType(StateType.INFECTIVE);
156                        adjacent.setDiseaseCycle(DiseaseCycle.LATENT);
157
158                        adjacent.setInfectedSinceDays(1);
159                        res.setContact(res.getContact() + 1);
160                     }
161                     break;
162                  }
163               }
164            }
165         }
166      }
167   }
168
169      protected void handleSameCellInfections (
170         StepResult res, double probability, boolean contactInfection) {
171      ...
```

```
172     }
```

Based on the given neighborhood relation the individuals in the cells do have a likelihood to interact. The methods `handleNeighborCellInfections()` and `handleSameCellInfections()` are responsible for computing these connection probabilities. Furthermore, when two individuals are contacting and one of them is suffering from the disease, the infection probability is computed and the individualÕs parameters are set. Due to the reason that the methods are quite similar the more complex ones code is depicted (line 108-177).

```
173     public void handleContactInfections(CellularAutomaton ca) {
174        InfectionParameters szParam = InfectionParameters.getInstance();
175        StepResult res = StepResult.getInstance();
176
177        handleSameCellInfections (res, szParam.getContact_infection_rate(),
178              true);
179        handleNeighborCellInfections (ca, szParam, res,
180           szParam.getContact_infection_rate());
181     }
182
183     public void handleVectoredInfections() {
184        InfectionParameters szParam = InfectionParameters.getInstance();
185        StepResult res = StepResult.getInstance();
186
187        handleSameCellInfections(res, szParam.getVectored_infection_rate(),
188              false);
189     }
```

The state transition function δ computes the so-called vectored infections and the contact infections. Thus the methods `handleContactInfections()` and `handleVectoredInfections()` exists, which are using the helper methods `handleNeighborCellInfections()` and `handleSameCellInfections()` described above.

```
190     public void handleSpontanousInfections () {
191        InfectionParameters szParam = InfectionParameters.getInstance();
192        StepResult res = StepResult.getInstance();
193
194        for (Iterator individualIterator = individuals.iterator();
195           individualIterator.hasNext ();) {
196           CellIndividual individual = (CellIndividual) individualIterator.
197                 next ();
198
199           if (this.isTheCase(szParam.getSpontaneous_infection_rate())) {
200              individual.setStateType(StateType.INFECTIVE);
201              individual.setDiseaseCycle(DiseaseCycle.LATENT);
202              individual.setInfectedSinceDays(1);
203              res.setSpontanous(res.getSpontanous()+1);
204           }
205        }
206     }
```

If spontaneous infection is turned on in the simulation parameters are used for computing a probability if a spontaneous infection occurs at the actual time point at the actual individual.

```
207 public void handleMovement(CellularAutomaton ca) {
208     InfectionParameters szParam = InfectionParameters.getInstance();
209     StepResult res = StepResult.getInstance();
210
211     long index = 0;
212     int whereToMove = 0;
213     int ctr;
214     DiseaseCell regSZNeighbourCell;
215
216     int cellMembers = this.individuals.size();
217     for (int cellNumber = 0; cellNumber <= cellMembers - 1; cellNumber++) {
218       if ((this.isTheCase(szParam.getMovement_probability())) &&
219           (this.individuals.size() > 0)) {
220           whereToMove =
221               InfectionParameters.randomGenerator.nextInt(
222              this.getNeighbours().size());
223
224           Iterator findIterator = this.getNeighbours().iterator();
225           ctr = 0;
226           while (findIterator.hasNext()) {
227              if (ctr >= whereToMove) break;
228              try {
229                 regSZNeighbourCell = (DiseaseCell) ca.getCell((Long)
230                         findIterator.next());
231                 index = regSZNeighbourCell.cellIndex;
232              } catch (Exception e) { }
233              ctr++;
234           }
235
236           try {
237              DiseaseCell newCellPosition = (DiseaseCell) ca.getCell(index);
238              if ((newCellPosition.individuals.size() < szParam.
239                      getMaxCellCapacity ())) {
240                 if ((newCellPosition != null) && (newCellPosition.
241                         individuals != null)) {
242                    CellIndividual individuum =    this.individuals.get(0);
243                    ArrayList<CellIndividual> copyIndividuals = new ArrayList
244                          <CellIndividual>();
245                    ArrayList<CellIndividual> newIndividuals = new ArrayList
246                          <CellIndividual>();
247                     newIndividuals.addAll(newCellPosition.individuals);
248                     newIndividuals.add(individuum);
249                    copyIndividuals.addAll(1, this.individuals);
250                    this.individuals.clear();
251                    this.individuals = copyIndividuals;
252                     newCellPosition.individuals.clear();
253                    newCellPosition.individuals = newIndividuals;
254
255                    res.setMoved(res.getMoved() + 1);
256                 }
257              }
258           } catch (Exception e) {}
259       }
260     }
261 }
```

The method `handleMovement()` computes using a random number if and where the individuals of the cell should move. Moving paths are strictly limited to the underlying

neighborhood relation. As expanded neighborhoods can be defined, it is possible that one individual can move long distances in one single time step. To give one example, using such a neighborhood relation enables to connect far-off places connected by infrastructur circumstances like airports. These far distance neighbors can be disconnected during the simulation, as airports were closed in China during the SARS outbreak.

3.3.2 The class `Cellindividual`

Each cell is able to hold a set of individuals, and furthermore, each individual has another finite state automaton working inside. Thus, it is possible to store the actual state and actual parameters of each individual. Using these parameters it is possible to control each individual separately. For example, it is possible to set quarantine parameters for some individuals or to use a special medication. These lists are also known as meme lists.

```
1 package Pandemie;
2
3 public class CellIndividual {
4     public enum StateType {
5           PASSIVEIMMUNEFROMBIRTH, SUSCEPTIBLE,
6           INFECTIVE, RECOVERED, KILLEDBYDISEASE, DIED
7     }
8
9     public enum AgeType {
10        KID, TEEN, ADULT, ELDERLY
11    }
12
13    public enum TreatmentType {
14        MEDICAL1, MEDICAL2, NOTREATMENT
15    }
16
17    public enum QuarantineType {
18        NORMAL, QUARANTINE
19    }
20
21    public enum DiseaseCycle {
22        HEALTHY, LATENT, INFECTIOUS, REMOVED, NIL
23    }
```

The class `CellIndividual` stores the memes and the different states of each individual. This class allows to model and extend any meme list for simulating social behavior more precicely.

```
24    private StateType stateType;
25    private AgeType ageType;
26    private QuarantineType quarantineType;
27    private TreatmentType treatmentType;
28    private DiseaseCycle diseaseCycle;
29
30    private int infectedSinceDays;
31    private int susceptibleInDays;
32    private double mortalityRateFactor = 1d;
```

Here, the representation of the individual states is implemented. The attributes have to be used for storing the individual memes and states.

```
33   public CellIndividual () {
34      this.setStateType(StateType.SUSCEPTIBLE);
35       this.setTreatmentType(TreatmentType.NOTREATMENT);
36      this.setDiseaseCycle(DiseaseCycle.HEALTHY);
37      this.setInfectedSinceDays(0);
38      this.setSusceptibleInDays(0);
39
40      int ageClass = InfectionParameters.randomGenerator.nextInt(4);
41      switch (ageClass) {
42        case 0 : this.setAgeType(AgeType.KID); break;
43        case 1 : this.setAgeType(AgeType.TEEN); break;
44        case 2 : this.setAgeType(AgeType.ADULT); break;
45        case 3 : this.setAgeType(AgeType.ELDERLY); break;
46        default : this.setAgeType(AgeType.ADULT); break;
47      }
48   }
```

When a new individual is generated the constrcutor must be used. Per definition a new individual is always in state *healthy* and *susceptible*, but using the `set` methods these parameters can be changed. The used age-type is dependent on a random number ranged form [1..4].

```
49   protected double computeMortalityRate (double morbidityValue,
50        InfectionParameters simParam) {
51      double value = 1.0d;
52
53      if (simParam.isHandleMedication()) {
54         value = this.isTheCase(0.5d) ?
55            simParam.getMedicationOne() : simParam.getMedicationTwo();
56      }
57
58      return morbidityValue / value;
59   }
```

The method `computeMortalityRate()` computes the probability of an individual to be killed by the disease dependent on given parameters available in the meme list.

```
60   protected void updateStateType (InfectionParameters simParam) {
61      switch (stateType)
62      {
63        case PASSIVEIMMUNEFROMBIRTH:
64           this.setSusceptibleInDays(this.getSusceptibleInDays()-1);
65           if (this.getSusceptibleInDays() < 1) {
66              this.setSusceptibleInDays(0);
67              this.setInfectedSinceDays(0);
68
69              this.setStateType(StateType.SUSCEPTIBLE);
70              this.setDiseaseCycle(DiseaseCycle.HEALTHY);
71           }
72         break;
73        case SUSCEPTIBLE:
74           this.setInfectedSinceDays(0);
75           this.setSusceptibleInDays(0);
76           this.setDiseaseCycle(DiseaseCycle.HEALTHY);
77           break;
78        case INFECTIVE:
```

```
79              this.setInfectedSinceDays(this.getInfectedSinceDays()+1);
80
81              if (this.getInfectedSinceDays() >=
82                simParam.getRecoveredRemovedAfterDays()) {
83                 double mortalityRate = computeMortalityRate
84                       (simParam.getVirus_morbidity_percent(), simParam);
85
86                 if (this.isTheCase(mortalityRate)) {
87                    this.setStateType(StateType.KILLEDBYDISEASE);
88                    this.setDiseaseCycle(DiseaseCycle.REMOVED);
89                 } else {
90                    this.setStateType(StateType.RECOVERED);
91                    this.setDiseaseCycle(DiseaseCycle.HEALTHY);
92                    this.setInfectedSinceDays(0);
93                    this.setSusceptibleInDays(
94                      simParam.getSuspectibe_again_after_recover());
95                 }
96              }
97              break;
98           case RECOVERED:
99              this.setSusceptibleInDays(this.getSusceptibleInDays()-1);
100             if (this.getSusceptibleInDays() < 1) {
101                this.setStateType(StateType.SUSCEPTIBLE);
102                this.setDiseaseCycle(DiseaseCycle.HEALTHY);
103             }
104             break;
105          case KILLEDBYDISEASE:
106             this.setDiseaseCycle(DiseaseCycle.REMOVED);
107             break;
108          case DIED:
109             this.setDiseaseCycle(DiseaseCycle.NIL);
110             break;
111       }
112    }
```

This functions is for updating the individuals state. The parameters are stored in the singleton object, which holds the data of the disease being simulated.

```
113    protected void updateDiseaseCycle (InfectionParameters simParam) {
114       switch (diseaseCycle)
115       {
116          case HEALTHY:
117            break;
118          case LATENT:
119             if (this.getInfectedSinceDays() > simParam.getLatentPeriodDays
120                   ())
121                this.setDiseaseCycle(DiseaseCycle.INFECTIOUS);
122             break;
123          case INFECTIOUS:
124             if (this.getStateType() == StateType.KILLEDBYDISEASE)
125                this.setDiseaseCycle(DiseaseCycle.REMOVED);
126
127             if (this.getStateType() == StateType.RECOVERED)
128                this.setDiseaseCycle(DiseaseCycle.HEALTHY);
129             break;
130          case REMOVED:
131             this.setDiseaseCycle(DiseaseCycle.NIL);
```

```
132             break;
133         }
134     }
```

This functions is for updating the individuals disease life cycle state. The parameters are also stored in the singleton object, which holds the data of the disease being simulated.

```
135     public void updateIndividual () {
136         InfectionParameters simParam = InfectionParameters.getInstance();
137         StepResult sRes = StepResult.getInstance();
138 
139         updateStateType (simParam);
140         updateDiseaseCycle (simParam);
141 
142         if (simParam.isUseQuarantine()) checkQuarantine ();
143 
144         adaptStatistics (sRes);
145     }
```

Each individual state needs to be updated after a simulation step. The method `updateIndividual()` handles this and calls a method for updating the step and individual statistics for performing analysis afterwards.

```
146     public void adaptStatistics (StepResult sRes) {
147         switch (stateType)
148         {
149           case PASSIVEIMMUNEFROMBIRTH:
150               sRes.setPassiveimmunityfrombirth(
151                sRes.getPassiveimmunityfrombirth()+1);
152            break;
153           case SUSCEPTIBLE:
154               sRes.setSusceptible(sRes.getSusceptible()+1);
155               break;
156           case INFECTIVE:
157               sRes.setInfective(sRes.getInfective()+1);
158               break;
159           case RECOVERED:
160               sRes.setRecovered(sRes.getRecovered()+1);
161               break;
162           case KILLEDBYDISEASE:
163               sRes.setKilledbydisease(sRes.getKilledbydisease()+1);
164               break;
165           case DIED:
166               sRes.setDied(sRes.getDied()+1);
167         }
168 
169         switch (diseaseCycle)
170         {
171           case HEALTHY:
172               sRes.setHealty(sRes.getHealty()+1);
173               break;
174           case LATENT:
175               sRes.setLatent(sRes.getLatent()+1);
176               break;
177           case INFECTIOUS:
178               sRes.setInfectious(sRes.getInfectious()+1);
```

```
179             break;
180           case REMOVED:
181             sRes.setRemoved(sRes.getRemoved()+1);
182             break;
183           case NIL:
184             sRes.setRemoved(sRes.getRemoved()+1);
185             break;
186         }
187     }
```

Updates the general statistics data after each simulation steps.

```
188     public void checkQuarantine () {
189         InfectionParameters simParam = InfectionParameters.getInstance();
190         switch (stateType)
191         {
192         case INFECTIVE:
193           if (this.getInfectedSinceDays() > simParam.getIncubationPeriodDays
194                ()) {
195             this.quarantineType = QuarantineType.QUARANTINE;
196           } else this.quarantineType = QuarantineType.QUARANTINE;
197           break;
198          default: this.quarantineType = QuarantineType.QUARANTINE;
199         }
200     }
201 }
```

The method `checkQuarantine()` is used by the state transition function in case the quarantine option is enabled. If an individual is infected, if the individual shows symptoms, and if quarantine is enabled then the individual is set to quarantine. In this case the individual has no, or a very limited chance, to infect a healthy individual.

3.4 The class `DiseaseSpreadCellularAutomaton`

```
 1 package Pandemie;
 2
 3 public class DiseaseSpreadCellularAutomaton extends CellularAutomaton {
 4     public static int timers = 0;
 5
 6     public void compute() {
 7         StepResult sRes = StepResult.getInstance();
 8         InfectionParameters simParam = InfectionParameters.getInstance();
 9
10         long timer;
11         System.out.println (sRes.getHeader());
12
13         for (timer = this.getStartTime(); timer <= this.getStopTime();
14               timer++) {
15             System.out.print(timer + "\t");
16
17             super.compute();
18
19             this.writeSpread("individuals", false);
20             this.writeSpread("susceptible", false);
21             this.writeSpread("infected", false);
22             this.writeSpread("recovered", false);
```

```
23          this.writeSpread("combined", true);
24
25          adaptParameters(timer, 15, true, false, simParam,
26             sRes, 1.2d, 1.5d, 1.1d, 1.05d);
27          useQuarantineAfter (timer, 50, false, simParam);
28       }
29    }
```

The class `DiseaseSpreadCellularAutomaton` is inherited from the basic class named `CellularAutomaton`. The function of the `compute()` method is to iterate through the CA cells and calls the state transition function δ. Therefore, the method iterates from the start timer to the end point and calls the compute method of the super class. The super class itself calls the method `performAction()`, which is known as the state transition function δ. Furthermore, using the helper method `writeSpread()` the simulation step data is persistently stored, and the method `adaptParameters()` is used for adapting the social behavior and the contact probability. The method `useQuarantineAfter()` could be used for drastic intervention into the system - the usage of quarantine can be enabled and parametrized.

```
30    public long countIndividuals() {
31       long individuals = 0;
32       for (Iterator cellIterator = this.getCellList().iterator();
33          cellIterator.hasNext ();) {
34          DiseaseCell cell = (DiseaseCell) cellIterator.next();
35
36          for (Iterator individualIterator = cell.getIndividuals().
37                iterator();
38             individualIterator.hasNext();) {
39             CellIndividual indiv =
40                (CellIndividual) individualIterator.next();
41
42             if ((indiv.getStateType() != StateType.DIED) &&
43                (indiv.getStateType() != StateType.KILLEDBYDISEASE))
44                individuals++;
45          }
46       }
47
48       return individuals;
49    }
50
51    public void useQuarantineAfter (long timer, int time, boolean doIt,
52       InfectionParameters simParam) {
53       if ((timer >= time) && (doIt))
54          simParam.setUseQuarantine(true);
55    }
56
57    public void adaptParameters(long timer, long reduceAfter, boolean doIt,
58       boolean stopSpontanous, InfectionParameters simParam,
59       StepResult sRes, double reduceSpontanousFactor,
60       double reduceMorbidityFactor, double reduceContactFactor,
61       double reduceVectoredFactor) {
62       if ((stopSpontanous) && (sRes.getInfective() > 0))
63             simParam.setSpontaneous_infection_rate(0.0d);
64
65       if ((doIt) && ((timer % reduceAfter) == 0)) {
66          if (sRes.getInfective() > 0)
67             simParam.setSpontaneous_infection_rate(
```

```
68                  simParam.getSpontaneous_infection_rate() /
69                  reduceSpontanousFactor);
70
71              simParam.setVirus_morbidity_percent(
72                  simParam.getVirus_morbidity_percent() / reduceMorbidityFactor);
73              simParam.setContact_infection_rate(
74                  simParam.getContact_infection_rate() / reduceContactFactor);
75              simParam.setVectored_infection_rate(
76                  simParam.getVectored_infection_rate() / reduceVectoredFactor);
77          }
78      }
79  }
```

3.5 Sample of a virus disease spread simulation

3.5.1 Geographic model

3.5.1.1 Austria

In the first simulation scenario a map of Austria was used. The model was simplified due to a homogenous population density over the whole country. The used map is depicted in figure 1.

Fig. 1. Geographical map of Austria with its nine states.

3.5.1.2 Tyrol

For the second simulation scenario the state Tyrol was chosen. Tyrol has 660.000 inhabitants, where about 115.000 inhabitants are living in the capital Innsbruck. The total area is 10.628 square kilometers. The area of settlement is about 1.600 square kilometers. Figure 2 depicts the geographical map of Tyrol and the population density is figured using colors from white, light yellow up to red.

3.5.1.3 Parameters

The simulated infectious disease used for the simulation is similar to the avian flu, except for the imperative difference that this virtual virus can be transmitted between human beings directly with a relatively high likelihood. Therefore, this virtual form of the H5N1 avian flu virus can be considered a dangerous mutation, which could have the power to effect an epidemic/pandemic situation. Table 2 depicts the parameters that have been used for the simulation experiments.

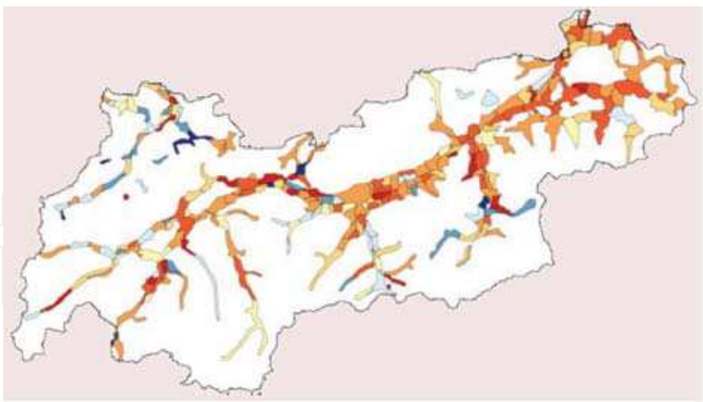


Fig. 2. population density of state Tyrol. The used colors (from white to red) for the population densities specify the density steps from 0, 200, 400, 600, 800 and 1000 inhabitants per square kilometer. The color light gray was used to describe the non-state area.

Table 1. default

Description	Value
Latent period in days	3
Infectious period in days	10
Recovered or removed after days	15
incubation period in days	3
Symptomatic period in days	4
Natural birth rate in percent	0.002
Natural death rate in percent	0.001
Virus morbidity in percent	0.63
Spontaneous infection rate in percent	0.00001
Vectored infection rate in percent	0.4
Contact infection rate in percent	0.6
Movement probability in percent	0.4
Immigration rate in percent	0.0000001
Re-Susceptible (temporary immunity) after days	100
Temporary immunity after birth in days	20

Table 2. Different parameters that were used during the simulation. After 100 time steps, the temporary immunity (Re-Susceptible after days) is lost completely. The parameter values for the infection cycle and the virus morbidity were chosen from the knowledge about the H5N1 human infections. The infection rate (vectored and contact) is supposed to be high in order to simulate a very aggressive (mutated) form of the virus that easily spreads from one individual to another. The other parameters were taken to model the behavior of the state Tyrol best possible.

3.5.2 State transition function δ

The algorithm iterates through each cell of the CA. Each cell represents a small area of the used geographical map and performs the operations of the n individuals placed in the cell (=location). The above described method `performCellAction()` computes the next discrete time step by considering following steps:

1. Handle the natural death cases
2. Handle the natural birth cases
3. Compute death caused by the disease
4. Compute the immigrants
5. Compute vectored infections
6. Compute contact infections
7. Compute spontaneous infections
8. Handle recovered individuals
9. Handle re-susceptible
10. Perform movement operations of the individuals
11. Adapt parameters according specification
12. Create output for actual time step

The steps (1-11) are performed until the specified number of time steps for the simulation is reached. During the simulation process snapshots of the actual distributions are created and furthermore, the data for subsequent statistical analysis is generated and stored. With this information it is possible to track each individual and to reconstruct the occured interactions. This enables the usage of statistical approaches for better understanding the disease spread mechanisms and to identify the best possible way to stop the spreading.

3.5.3 Simulation results

3.5.3.1 Austria

Three scenarios were simulated. The infection seed point was set to the capital of Austria, Vienna. In scenario A, neither medical treatment was provided nor was quarantine declared. In scenario B, two different medications were used for the treatment, but quarantine was not considered. The medication was aimed at increasing the healing chances by 45-55 percent. In scenario C, individuals were submitted to both, medical treatment and quarantine. Furthermore, the social behavior changes of the individuals during the simulation was considered. These behaviors werde modeled because when a disease is circulating, individuals are very cautious contacting others to minimize their own risk of infection.
Figure 3 depicts the development of the susceptibles over the time. As expected from declaring a quarantine status in scenario C, the infection spread stops.
Figure 4 shows the characteristics of the infection over the time in percent and in Figure 5, the fatal cases are illustrated. Assuming that there is no medication, and no quarantine declared, the highest death toll is observed. The difference between scenario B and C is based on the fact that in scenario B the medication is given from the first day on, whereas in scenario C the medication and the quarantine start 50 days after the outbreak.

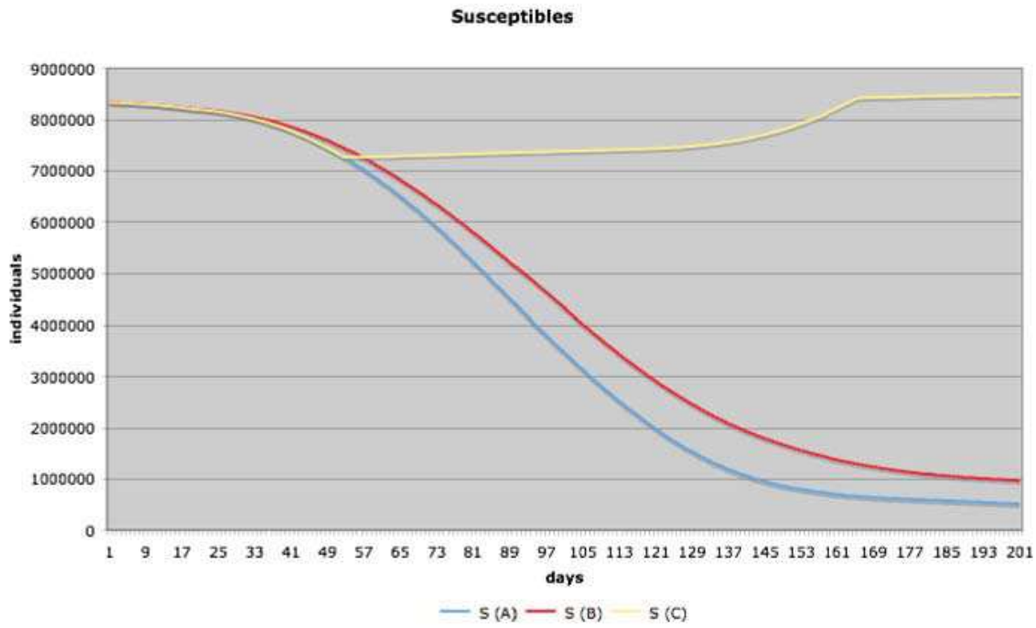


Fig. 3. Susceptible individuals over the simulated period.

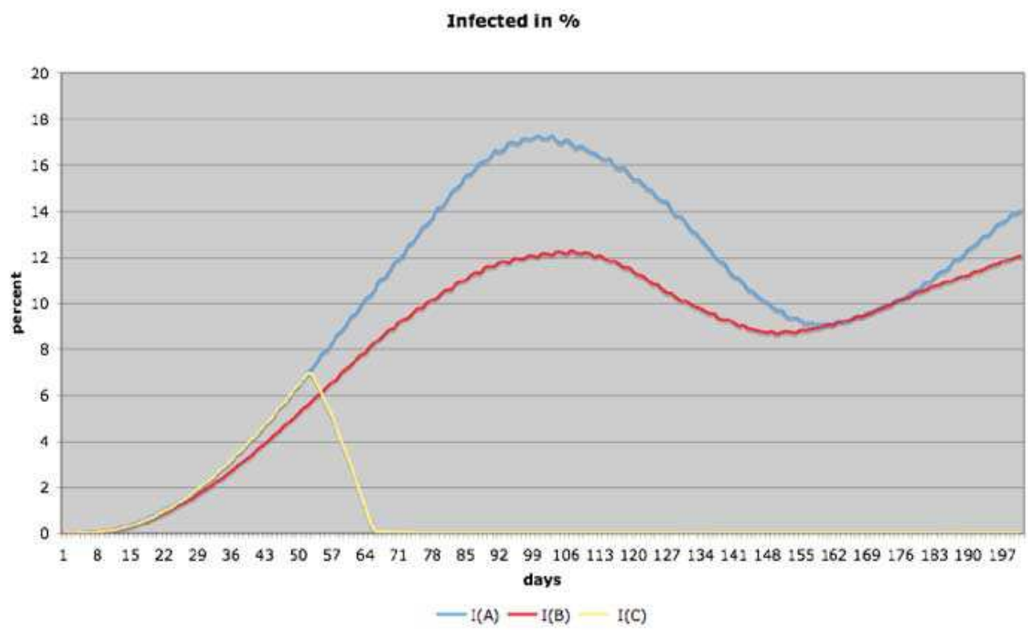


Fig. 4. Development of the infection over the simulated period.

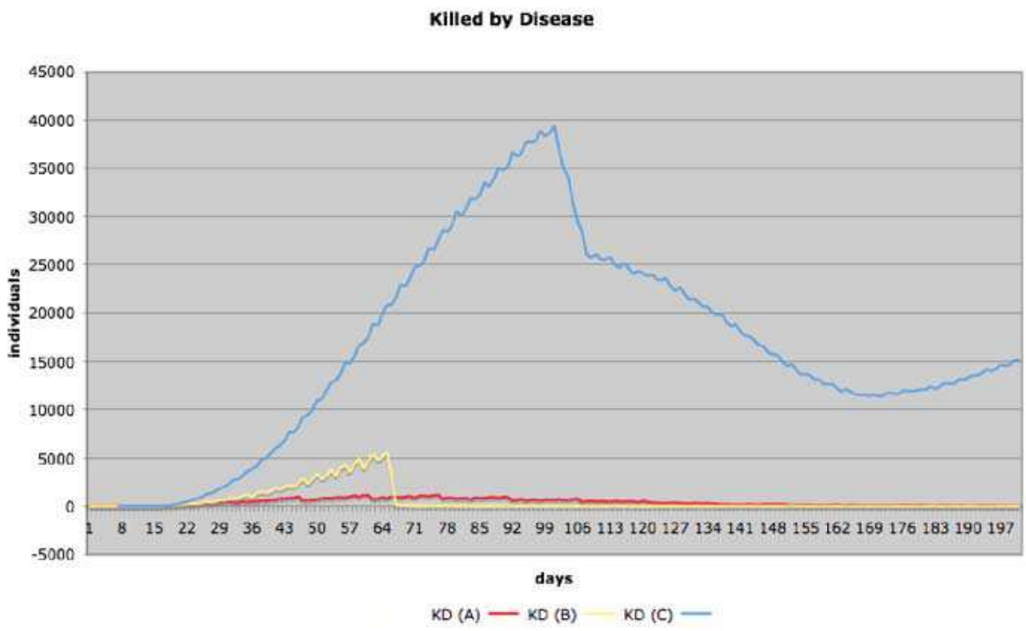


Fig. 5. Fatal cases of the three simulated scenarios.

Figure 6 depicts the parameters individuals, susceptible, infected and removed. As presented in figure 4b, the medi-cation slows down the spreading and reduces the fatal cases dramatically. When quarantine is consistently applied, the spread is controlled after a few days.

Figure 7 depicts the spatial results of the scenarios A, B, C at time point 50 days after outbreak. The dots and grey surfaces depict the areas where infected individuals are located.

At time point 65 days after outbreak (figure 8) the difference between the three simulated scenarios can be seen clearly. When no treatment and no quarantine are applied, the infection spreads the most. The enacted quarantine (C) was able to stop the disease from further spreading few days, the fatal cases were also reduced in scenario B but the disease was still spreading.

3.5.3.2 Tyrol

Eight different scenarios were simulated 4. The seed point of the infection was set to the capital Innsbruck. In the first scenario (scenario A), the disease spread in the state Tyrol where medical treatment was performed. Two different drugs are available for infected individuals. Drug one reduces the death rate by 55 percent, whereas drug two reduces the death rate by 45 percent. The social behavior of the individuals changes during the simulation time, which would also occur in a real situation. When a fatal disease is circulating, individuals are very cautious contacting others to minimize their infection risk. The second scenario (scenario B) is similar to scenario A with the difference that no medical treatment is performed. Scenario C and D is equal to A and B with the difference that there is no adaptation of the social behavior. Scenario E and F is equal to scenario A and B with the difference that after 50 time steps a strictly controlled quarantine is introduced. In the last two scenarios (An, Bn), the same simulation parameters were applied as in A and B with the difference that no geographical and population density was used. Therefore, each cell covers the mean number of individuals from

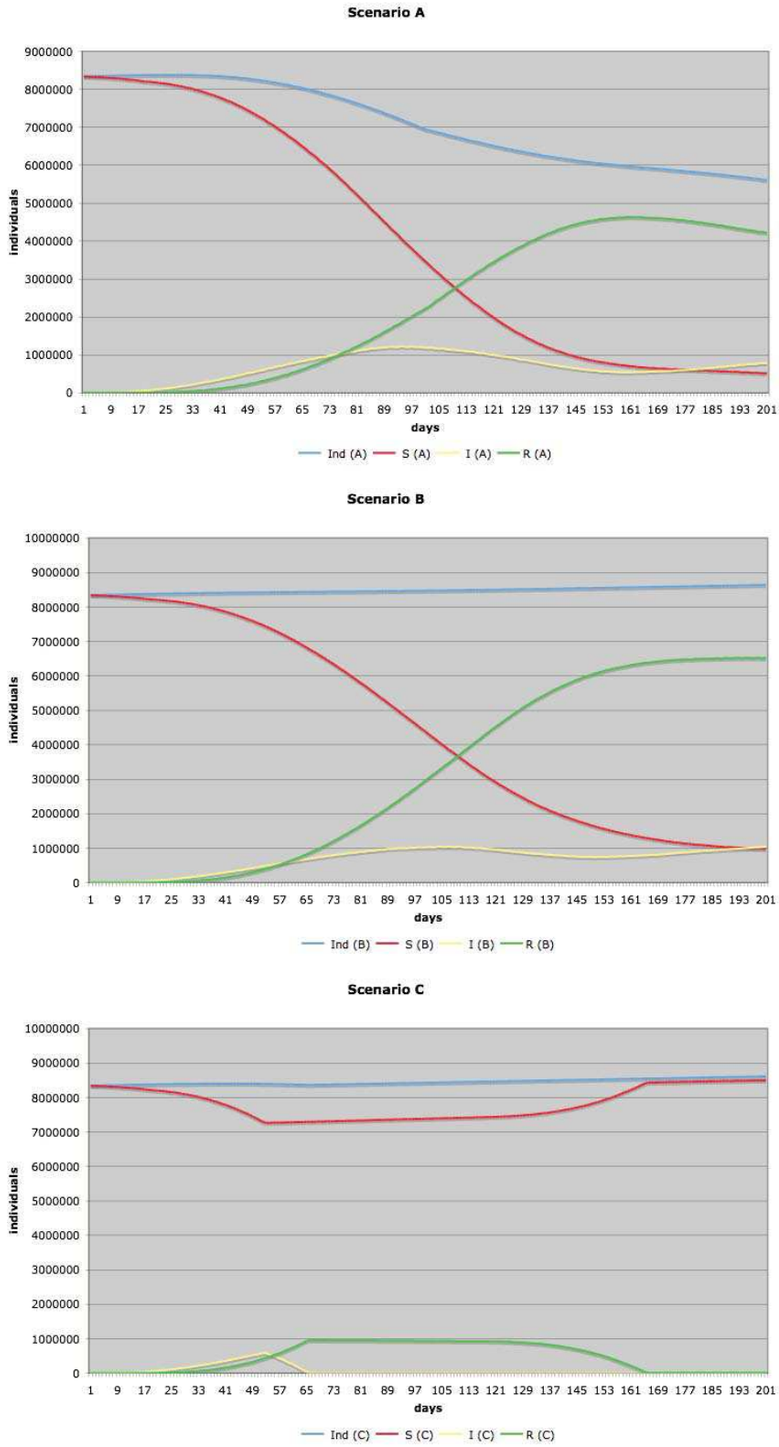


Fig. 6. Population, susceptibles, infected and removed individuals over the simulated period.

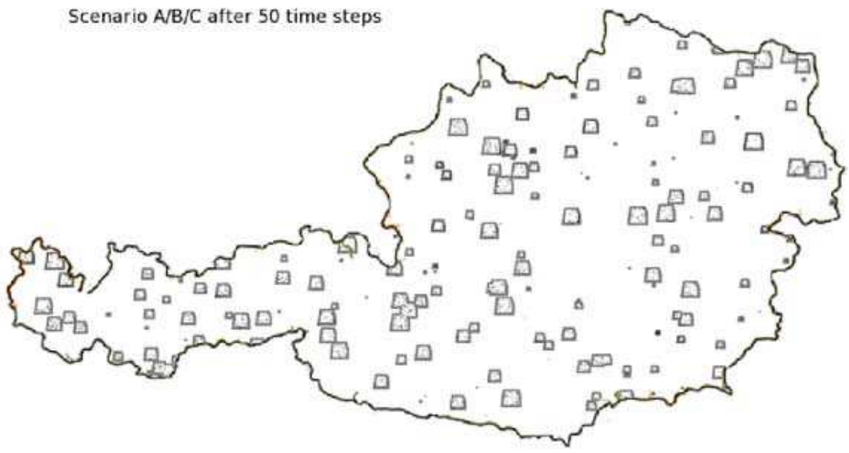


Fig. 7. Screenshot of the spatial result for scenarios A, B, C 50 days after outbreak.

the state Tyrol model. By comparing these scenarios with A and B, it is possible to find out the relevance of geographical (natural barriers) and population density information. The blue color (Ind) is used for the population, red color (S) depicts the susceptible individuals, yellow (I) is used to visualize the infected individuals and the green color (R) was taken to depict the removed or temporarily immune individuals. The virus's transmissibility (R_0 value) is such that each infectious case gives rise to 3.4 secondary infectious cases. The following figures (from figure 9 to figure 16) depict the classes susceptible (S), infected (I), and removed (R).

Table 3. default

scenario	medication	quarantine	social behavior	geographical conditions
A	x		x	x
B			x	x
C	x			x
D				x
E	x	x	x	x
F		x	x	x
An	x		x	
Bn			x	

Table 4. Overview of the different simulation scenarios in tabular view (x stands for true, no character for false). For more information see text.

In figure 17, the changes in population over the time are depicted. Figure 18 shows the fatal cases caused by the disease aggregated per month.
In the simulation, the value for the natural birth rate was 0.002 and the natural death rate 0.001. An infected individual can be removed during the simulation for three different reasons. The first way is that the individual is removed because of natural death, and then the individual can be removed because the disease ended fatal and the third way to be removed to another class is that the individual got healthy again. Figure 19 shows the percentage between natural

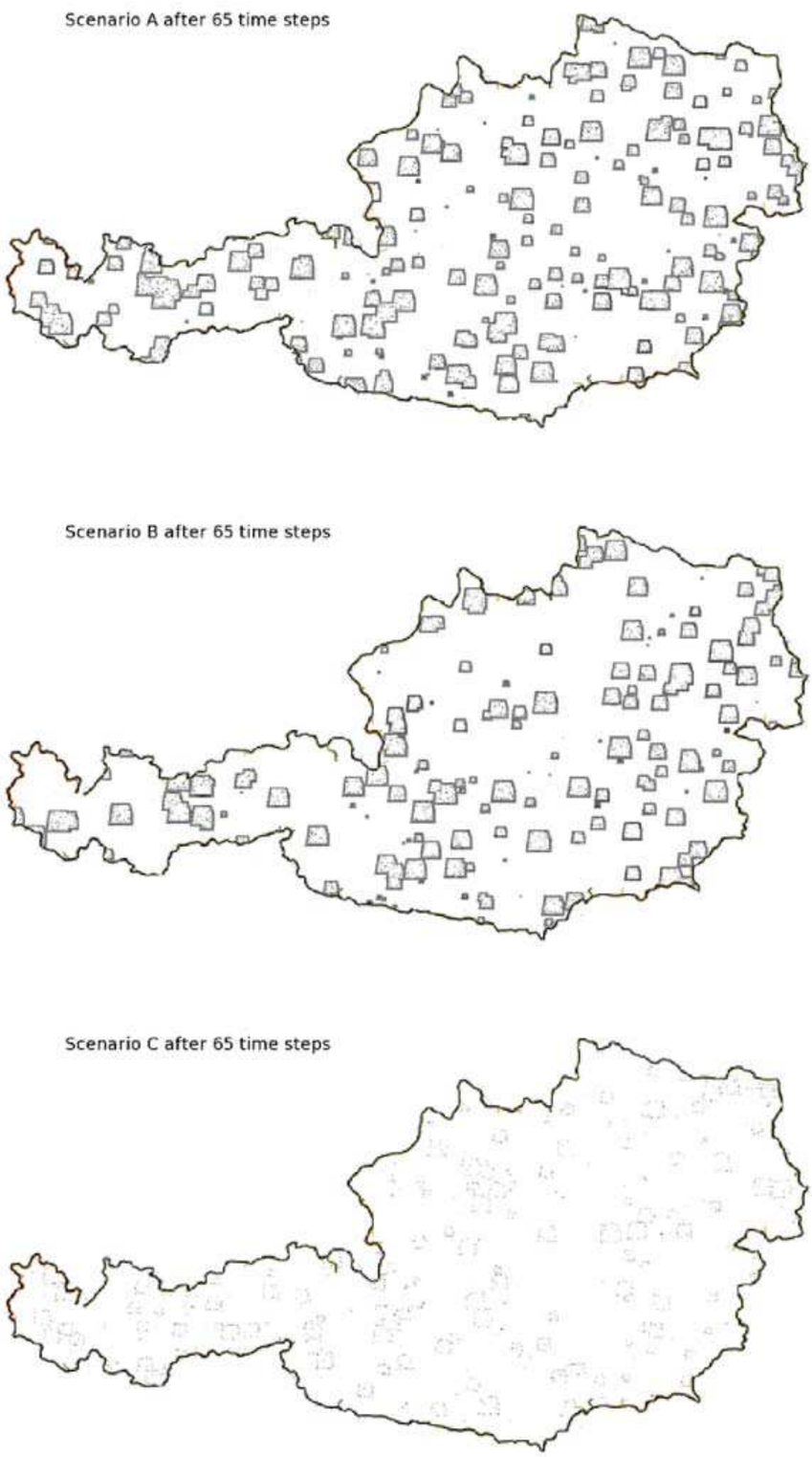


Fig. 8. Screenshot of the spatial result for scenarios A, B, C 65 days after outbreak.

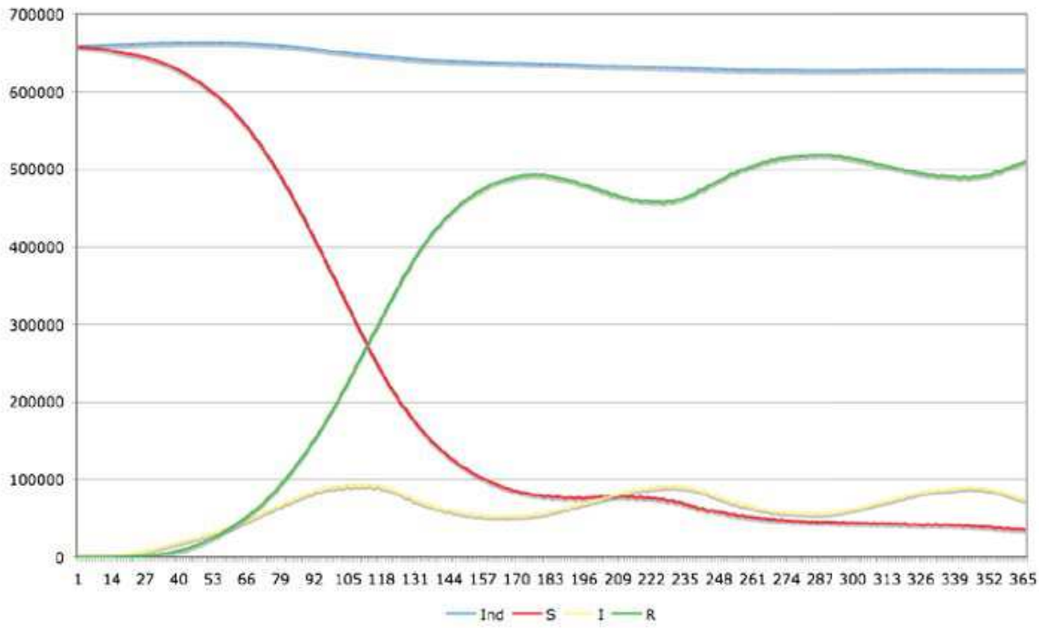


Fig. 9. Scenario A. Medical treatment is performed, and social behavior changes during the arising situation.

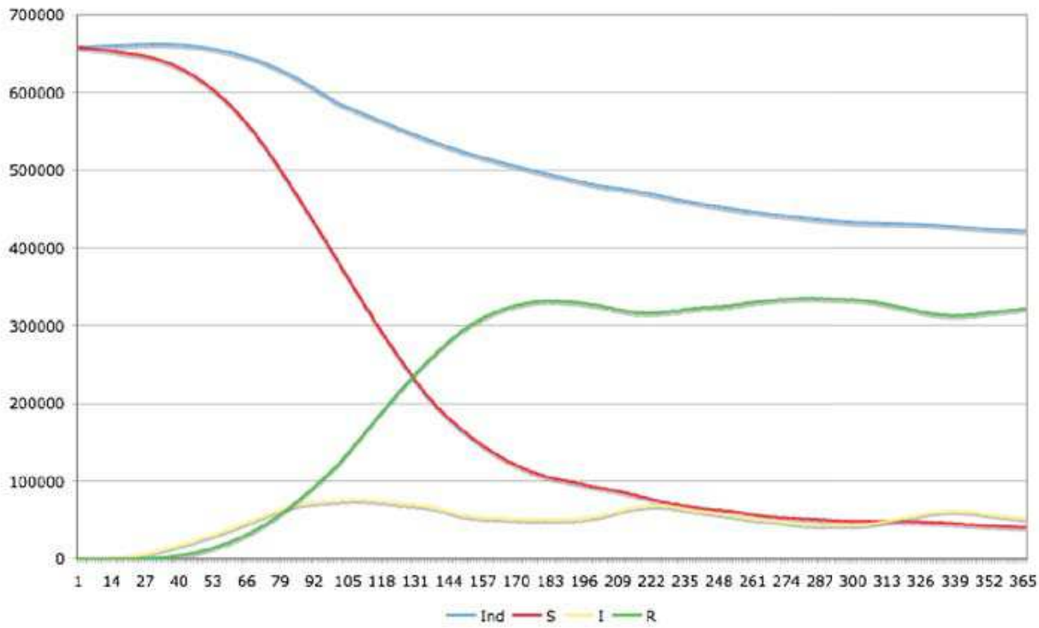


Fig. 10. Scenario B. No medical treatment is performed. Only the social behavior changes during the simulation run.

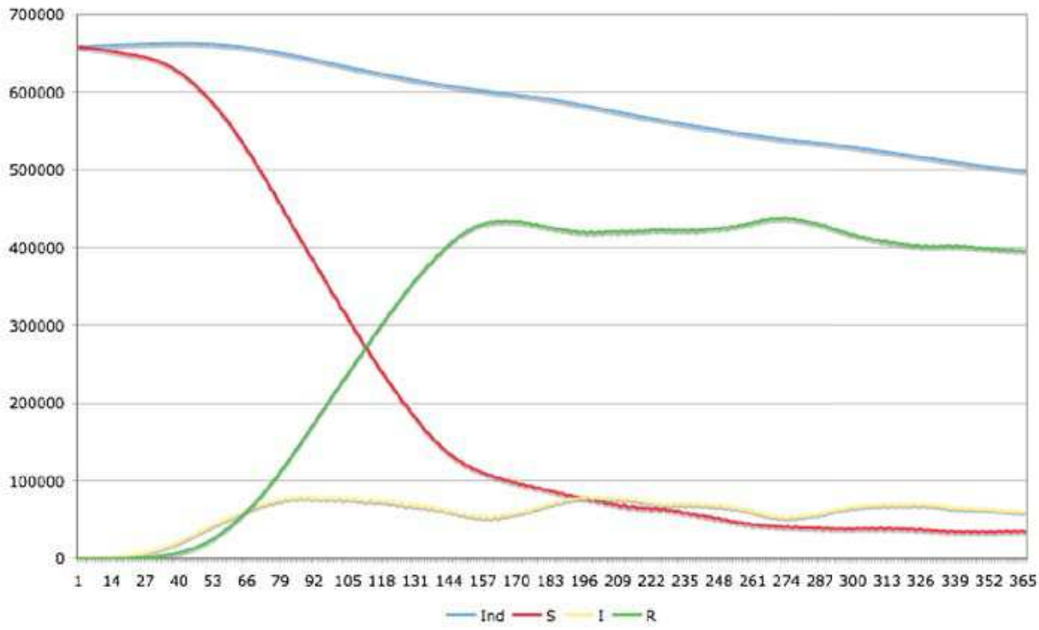


Fig. 11. Scenario C. Medical treatment is performed, but no changes in the individuals' behavior is simulated.

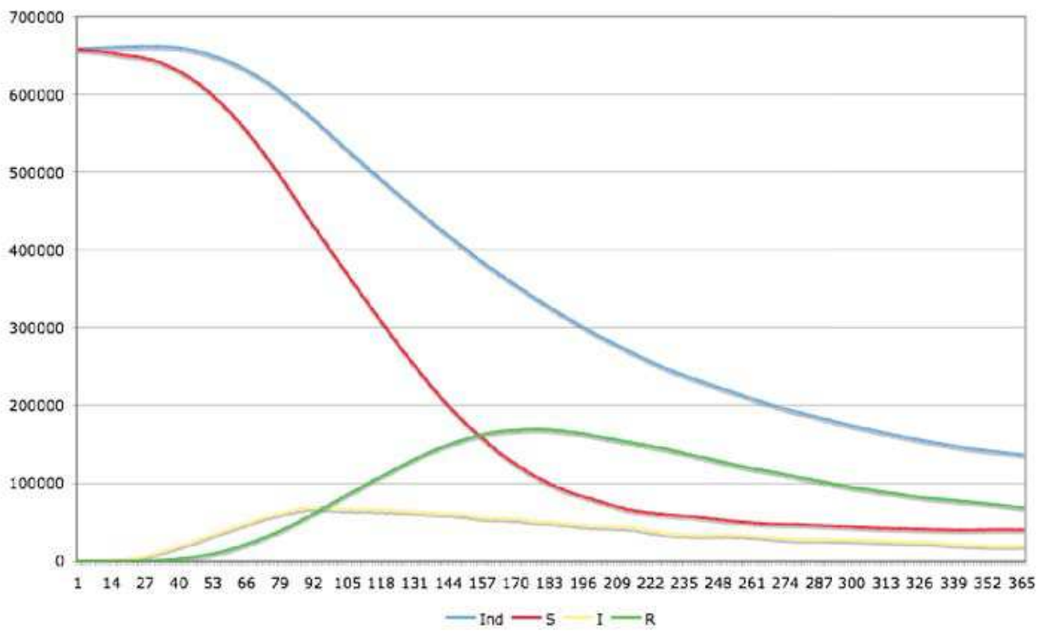


Fig. 12. Scenario D. No medical treatment and no change in the behavior is applied.

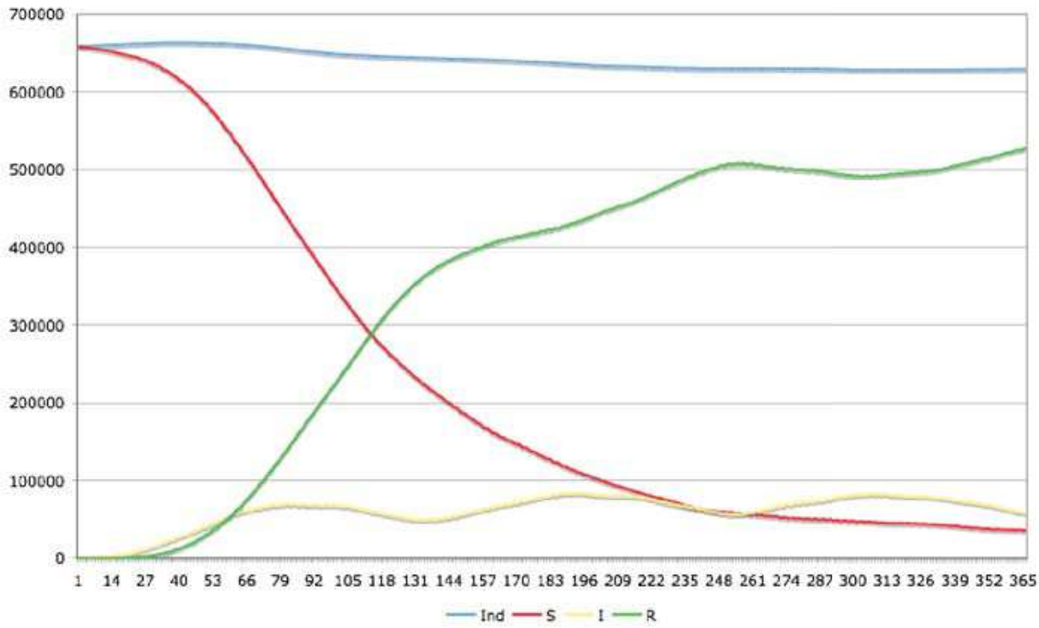


Fig. 13. Scenario E. Equal to scenario A with the difference, that after 50 days a controlled quarantine is applied.

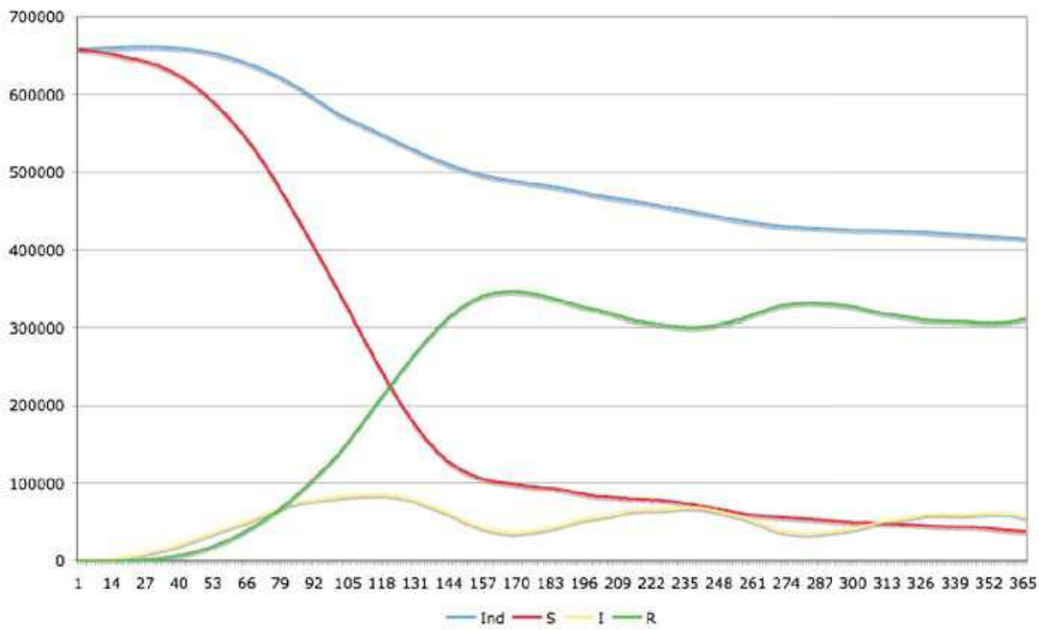


Fig. 14. Scenario F. Equal to scenario B with the difference, that after 50 days a controlled quarantine is applied.

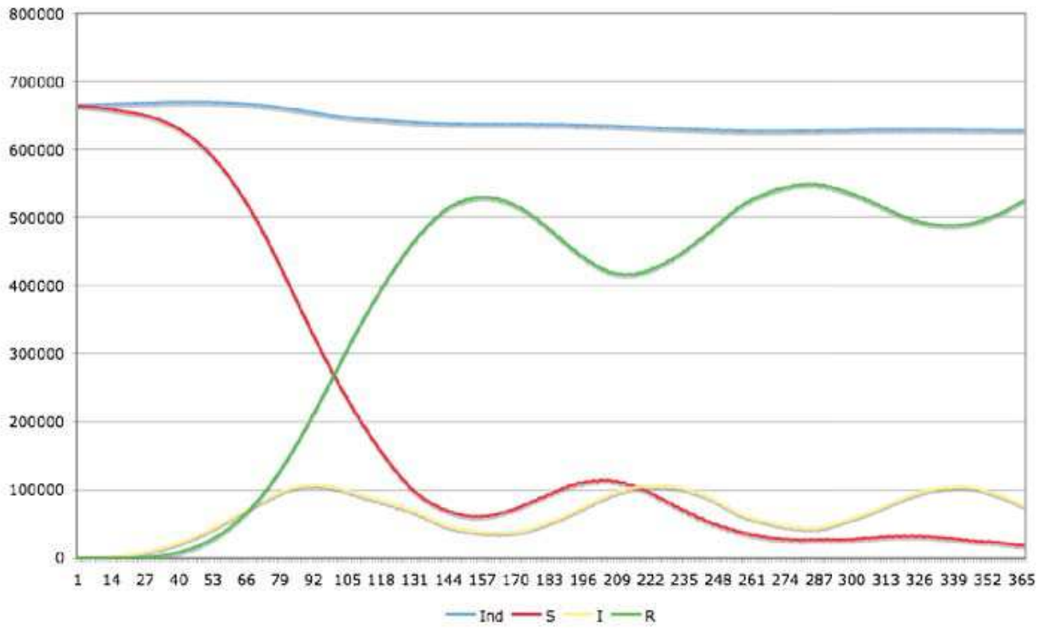


Fig. 15. Scenario An. Equal to scenario A with the difference that no geographical information was used. The population was therefore homogenous.

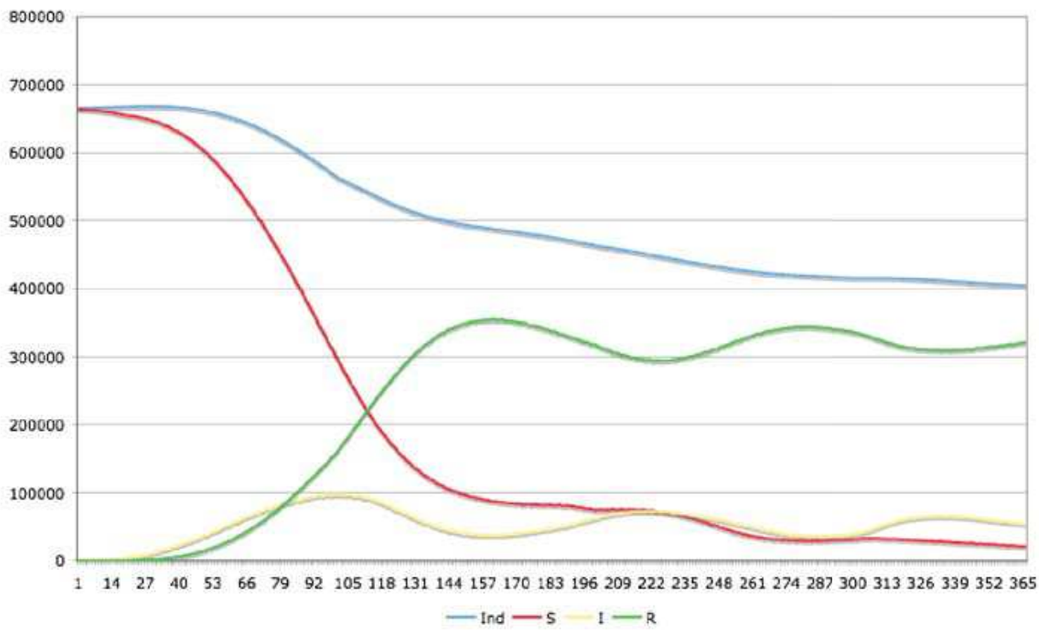


Fig. 16. Scenario Bn. Equal to scenario B with the difference that no geographical information was used. The population was therefore homogenous.

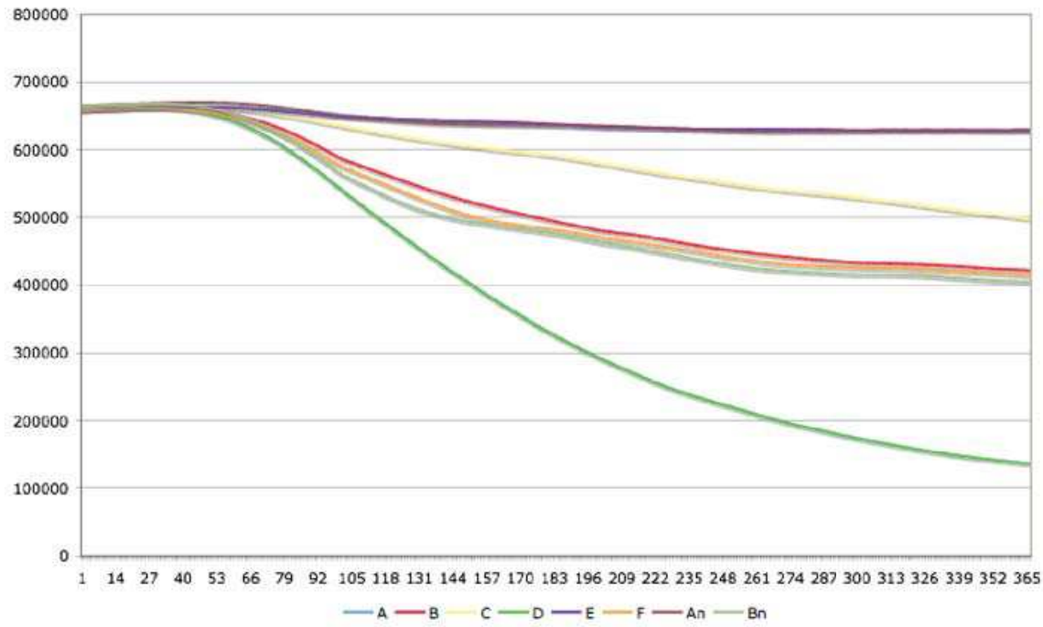


Fig. 17. Population change over the time.

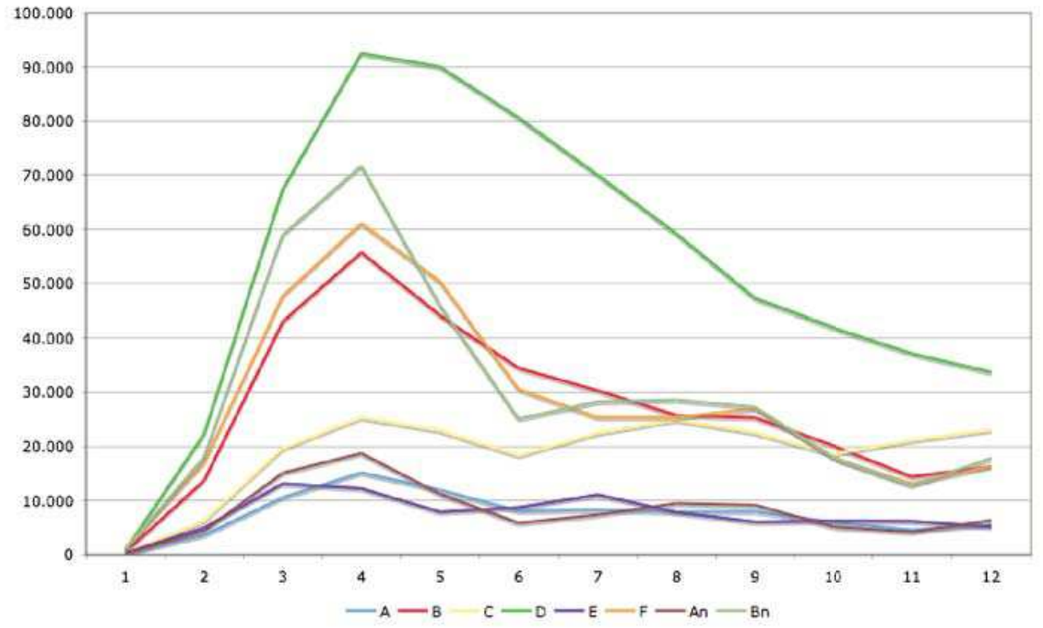


Fig. 18. Fatal cases aggregated per month.

death and diseases fatal cases for the simulated scenarios A to F and An, and Bn. The presented values are mean values per day.

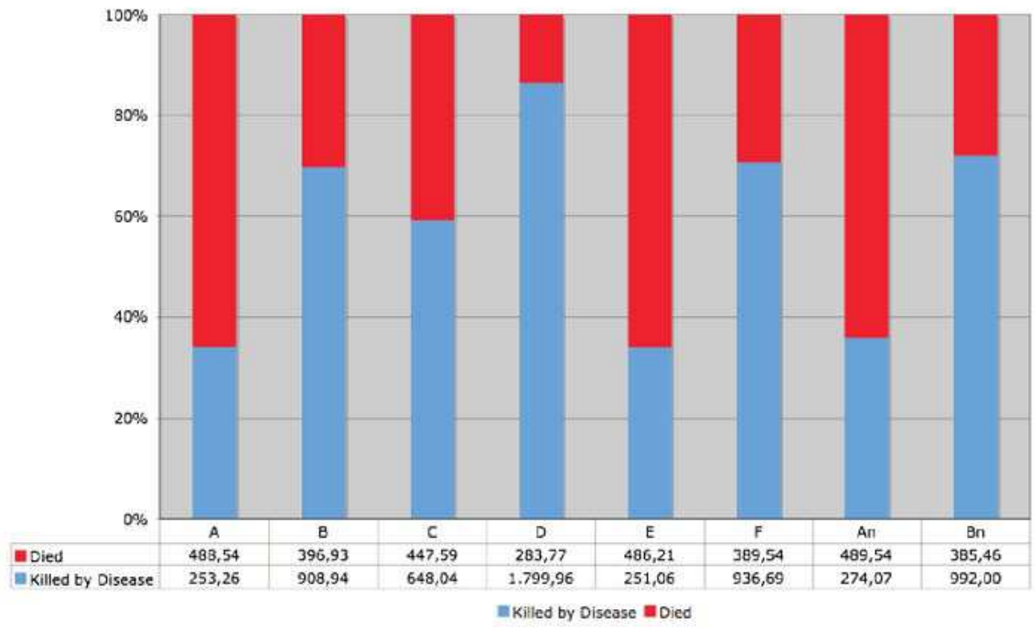
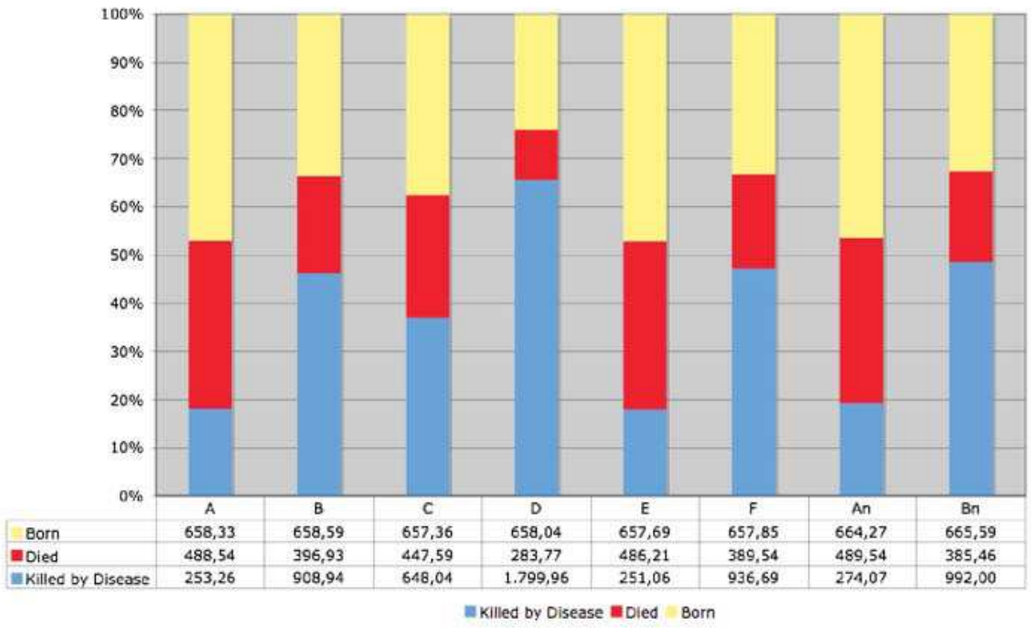


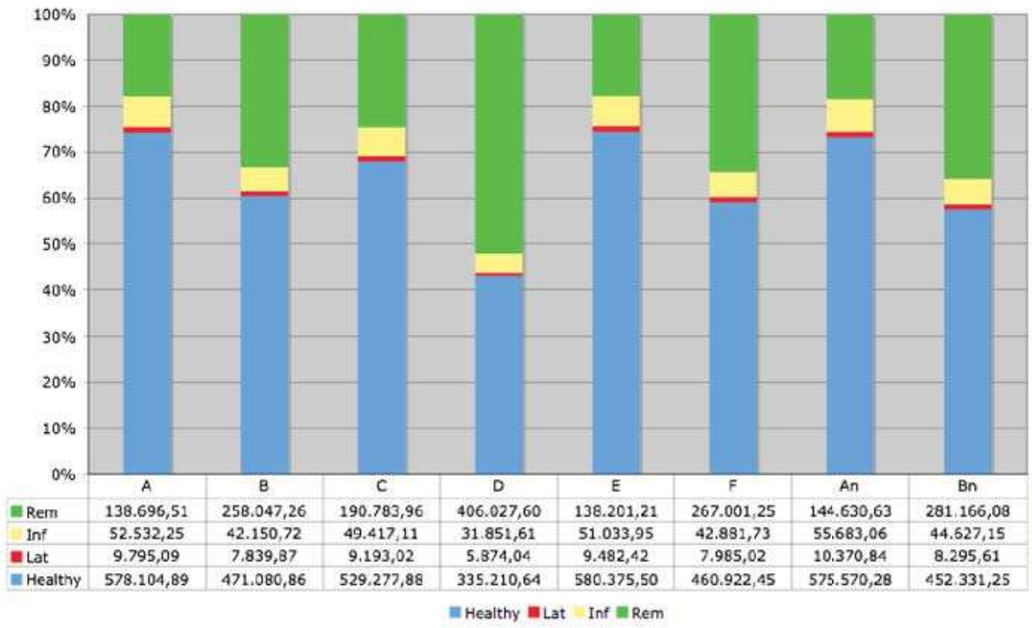
Fig. 19. Percentage diagram with mean cases per day. The figure shows clearly, that in the scenarios B, Bn, F, and especially D the death cases caused by the disease prevail.

When adding the natural births, it is possible to predict the population development during the disease spread. Figure 20 shows, as expected, that the population decreases in each scenario, but scenario D shows a dramatic decrease in the population.
The parameters used for the diseases life cycle were 3 days for the latent period, 10 days for the infectious period, the incubation period was set to 3 days and the symptomatic one to 4 days. After 15 days, the individual get removed or recovers from the disease. Figure 21 shows the ratio between the disease life cycle states. It is clearly visible that in simulation scenario D the most infections occur and the disease is able to spread the most. Furthermore, when comparing the scenarios A with An and B with Bn it can be figured out that the presence of geological and demographic realities reduces the spreading in a natural way.
The pie chart presented in figure 22 shows the perceptual distribution of the fatal cases per day.
The simulation showed that the geographic structure of the area is of importance as natural barriers slow down the velocity of the spread. A slower velocity coupled with the change of the natural behavior of the individuals helps to reduce cases of death. Because in this simulation the temporary immunity was set to 100 days, the figures show a periodicity with decreasing amplitude.
As a cellular automaton has a cellular space or cellular lattice, these models allow the visualization of the automaton states at each time point. The regular lattice consists of several individual cells, which interact using a neighborhood relation. In this simulation, a Moore neighborhood instead of a von Neumann neighborhood or 2-radial neighborhood was taken for the simulation. It has to be noted down that when simulating the scenarios using a



	A	B	C	D	E	F	An	Bn
Born	658,33	658,59	657,36	658,04	657,69	657,85	664,27	665,59
Died	488,54	396,93	447,59	283,77	486,21	389,54	489,54	385,46
Killed by Disease	253,26	908,94	648,04	1.799,96	251,06	936,69	274,07	992,00

Fig. 20. Development of the population during the outbreak within the different scenarios.



	A	B	C	D	E	F	An	Bn
Rem	138.696,51	258.047,26	190.783,96	406.027,60	138.201,21	267.001,25	144.630,63	281.166,08
Inf	52.532,25	42.150,72	49.417,11	31.851,61	51.033,95	42.881,73	55.683,06	44.627,15
Lat	9.795,09	7.839,87	9.193,02	5.874,04	9.482,42	7.985,02	10.370,84	8.295,61
Healthy	578.104,89	471.080,86	529.277,88	335.210,64	580.375,50	460.922,45	575.570,28	452.331,25

Fig. 21. Tracking of the disease life cycle for the individuals for the different simulation scenarios.

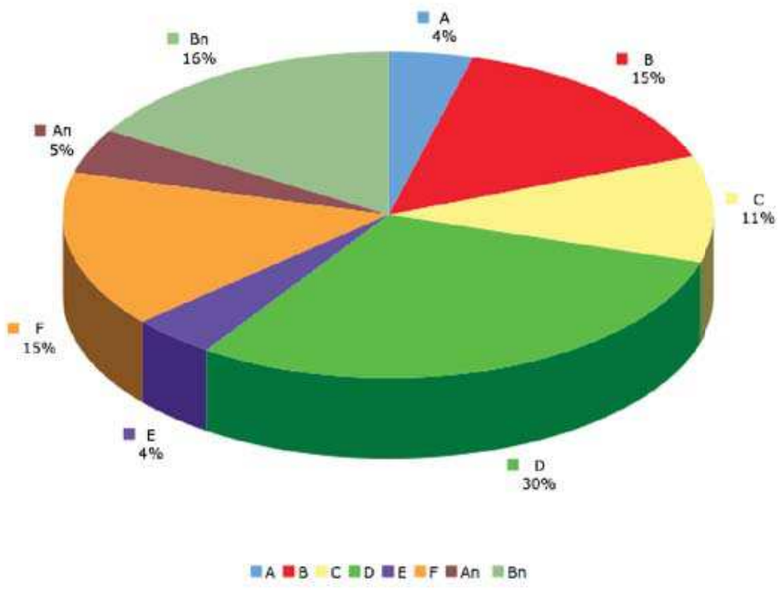


Fig. 22. Percential distribution of fatal cases per day computed by using mean value.

2-radial or von Neumann neighborhood the graphical representation is smoother and ringlike compared to the presented ones, but the overall behavior is almost equal.
Figure 23 depicts the spatial results of the disease spread for scenario A. When analyzing the spreading, one can see that the geographical properties and the different densities in the valleys of Tyrol are responsible for slowing down the outbreak. However, the geographical conditions are unable to stop the outbreak completely.

3.5.3.3 Computing time

The run time of the simulation for the state Tyrol simulation was $\mu = 51.87\ [min]$ minutes ($\sigma = 1.87\ [min]$) per scenario (simulation of 365 time steps per scenario) with four scenarios started parallel on the same machine. The run time of the simulation for Austria was 5.4 $[h]$ hours per scenario. The simulation was performed using an Apple X-Serve 1.1 OS X Server Version 10.4.10 with 2 x 2 GHz Dual Core Intel Xeon processor and 2 GB of RAM installed. The mean storage per scenario was about 10 MB (spatiotemporal snapshots per time step and overall information). The maximum capacity of the framework is only limited by the available memory. A simulation with 10 billion individuals, which is enough to simulate a spread of a disease over the whole globe, would be possible, however, the simulation time would be high. Since the insights, which wanted to be obtained on mechanisms and procedures of disease spread, are based on extensive and complex simulations, it is of great importance to have the possibility to run a large number of simulations or simulations on fine-grained models in a short period without the worry of long simulation periods. It is vital to have the opportunity to experiment with different parameters of the models, in terms of vaccination strategies, behavioral patterns of individuals, model variations and so on. At the European Grid Conference in Amsterdam a framework Wurz & Schuldt (n.d.) for seamless parallel execution of various kinds of algorithms was published. The framework can be used to schedule execution of software in parallel without the burden for the application developer

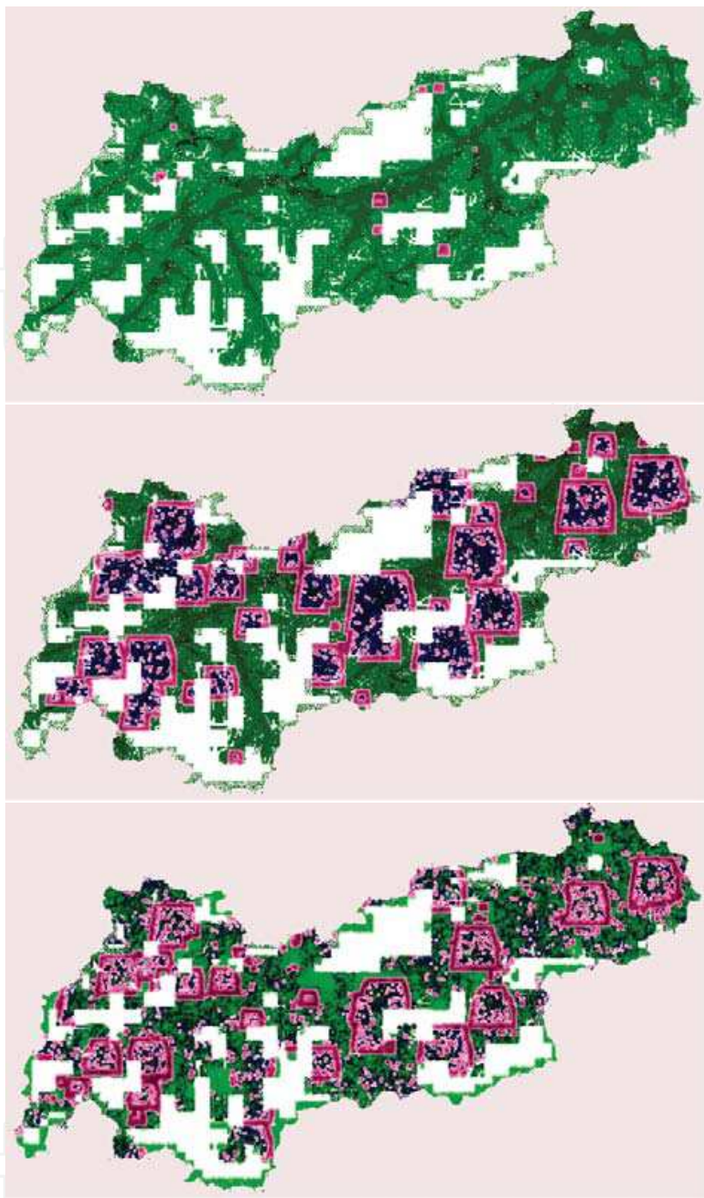


Fig. 23. The colors were used as following: green describes individuals in the state susceptible (S), gentle-pink marks individuals as in state infective (I) and dark blue marks the individuals as to be in state recovered (R). The first graphic depicts the simulation after 20 days. Although the simulation has started in the capital of Tyrol, in Innsbruck, after 20 days there are some outbreaks in the west of the state and in the east, which results form the fact that individuals are moving from cell to cell. Furthermore, unknown activities, which are denoted as spontaneous infection, are responsible for these characteristics. The second image shows a snapshot 80 days later at the time point 100 days after the infection started. It is clearly visible that in the capital where the disease spread started most people are in state recovered. Individual who did not die from the disease do have a temporary immunity. More than 50% of the individuals are in state infected. The last image shows a snapshot at time point 200 days after outbreak where individuals may get infected again. This represents the second outbreak wave with smaller amplitude.

to know details about the computational resources available, and that the framework, acting as a middleware, allows for a dynamic adaptation of the scheduling process.The framework is designed to cope with heterogeneous resources in a dynamic and rather instable network, so that it can be used to utilize computation power available on the Internet or the universities network to speed up simulations of the CA Framework or add additional facets like visualization Ôon the flyÕ. The usage of grid computing in CA model simulation lends itself to be perfect as CA models can be parallelized perfectly. In the first simulation, an Apple Xgrid environment was used. Apple's Xgrid technology enables to use ad hoc groups of Macintosh system into low-cost supercomputer.

3.6 Summary

The framework enables the simulation of different communicable diseases by specifying the disease parameters and demographic characteristics. Furthermore, the population can be divided into subgroups, which enables to simulated different impacts of the disease on each individual. The described scenarios demonstrated that CA and agent based models can be used for simulating and visualizing the spread and the adherent impact of infectious diseases. Furthermore, the simulation environment allows the access to any individual parameter at any point in time of the simulation, which enables detailed statistical analysis. Although the connections and the affiliated behavior between the individuals can be modeled using different neighborhood relations, the behavior of any individual in these models depends on functions using random numbers. For upgrading the behavior algorithms, social behavior approaches should be used and the computation of the economic impact should be computed for better creation of public health strategy plans for managing fatal diseases. The natural manner Ð let us call it "Groundhog Day" - that most individuals do have a way of living caused by their daily workflow, can not be modeled correctly, using such functions. To solve this problem, virtual worlds could be helping. The well-known Second Life, where millions of people do live an additional life and do also have a behavior, which is very similar compared to their own, should be observed for simulations. One has to keep in mind, that building a population model from census and demographic data statistically equal to one in the real world would be very complex but also a deep impact to privacy, and therefore, afflicted with many of problems.

Recapitulatory one can state, that the proposed CA framework is able to support public health offices by providing them with information for creating plans to manage such situations and to prevent serious long-term economic repercussions.

4. Imaging of the cardiac electrical function using cellular automaton approach

4.1 Introduction

Simulation of the electrocardiogram (ECG) and of the body surface potential (BSP) have been a research topic during the last decades. Nowadays, because of the enormous computer power available and because of extensive knowledge about cardiac electrophysiology from the cellular to the tissue level, sophisticated three-dimensional approaches have been developed. Although, the models became very attractive during the last years, there are still extensive problems in making them useful for cardiovascular diagnosis and therapy. First, the individual parameters like fibre architecture, conductivities and others are not available to that extension needed. Second, today, the used cell membrane models have to consider molecular function as well. Finally, the models should be validated based on human data from the cellular to the organ level. This will imply further research necessary in the upcoming two

decades. The paper presented deals with an anisotropic ventricular and an isotropic atrial model developed by our research group during the last 15 years. This in silico approach is used in the electrocardiographic forward and inverse approach. Validation has been done for the inverse formulation in 45 patients only. The whole-heart model presented uses the bidomain source-field formulation. A detailed anatomical model of the atrium is considered as well. The geometrical data is derived from individual magnetic resonance images. Fibre architecture in the ventricle and anatomical features in the atrium, like Bachmann bundle or others are considered based on literature data. The whole-heart model allows the calculation of the de- and repolarization and of the three-dimensional potential pattern throughout the entire heart muscle and volume conductor. Based on this simulated potential data, the 12-lead standard ECG or different BSP maps can be visualized. Today, this in silico whole-heart model environment is used for enhancing the understanding of the nature of the ECG in the normal beat, for different arrhythmias, and for ischemia and infarction. A user-friendly software environment allows interactive model generation, parameter adjustment, simulation and visualization.

4.2 Methods: the forward problem

4.2.1 Volume conductor model

The volume conductor model (VCM) with the embedded cardiac source volume is the basis for the electrocardiographic forward problem. The VCM consists of the compartments chest, lungs, atrial and ventricular myocardium, and of the blood masses from an individual patient. The morphological imaging data were acquired using a Magnetom Vision Plus 1.5 Tesla scanner, Siemens Medical Solutions, Erlangen, Germany. For the lung and torso shape extraction a T1 flash, non-contrasted axial data set during breath-hold (expiration, 10 mm spacing) was used. The cardiac geometry (atrial and ventricular models) was acquired in ECG-gated cine mode during breath-hold (expiration, oblique short-axis scans) with 4 and 6 mm spacing. The segmentation of the compartments was performed using a recently developed VCM segmentation pipeline. The resulting labelsets were triangulated using a standard marching cubes algorithm and optimized using Hammer B. (2001). In the next VCM assembling step the tetrahedral mesh was created using the software package Hypermesh (Altair Eng.), which allows to produce optimized tetrahedral meshes on the basis of the high quality surface mesh. The whole heart VCM consists of 104,001 tetrahedrons and 18,170 nodes, respectively. The volume conductor (whole heart) model with its compartments is depicted in Fig. 24.

4.2.2 Computation of cardiac activation sequences - the cellular automaton

The CA used in this study was developed Fuchsberger M. (1993); Killmann R. (1987; 1990); Rosian M. (1991), modified and implemented in amiraDev 3.0™ Hayn D. (2002). Briefly, after segmentation, triangulation and tetrahedral mesh generation different types of tissue were assigned to the atria and ventricles with corresponding parameter setting for each tissue type. Further, the CA needs the fiber structure assigned to each node of the tetrahedral model as well as the refractory periods assigned to each tissue type.

In the models the types of tissue were the endocardia, epicardium and myocardium. Furthermore, the sinus node, crista terminalis, Bachmann bundle, fossa ovalis, pectinate muscles, coronary sinus, isthmus, the bundle of His, left and right bundle branch and the Purkinje fibers were defined. Each type of tissue has its own set of parameters needed for the computation of the activation times and the time dependent transmembrane potential distribution. The fiber geometry was chosen such, that it fitted qualitatively well with the

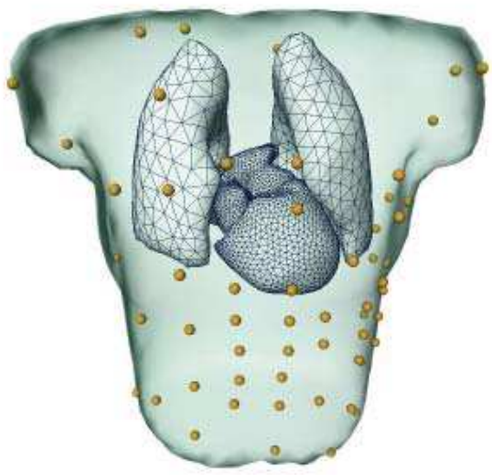


Fig. 24. VCM of an individual patient comprising chest surface, atria and ventricles with the cavitary blood masses and both lungs displayed in an anterior-posterior view.

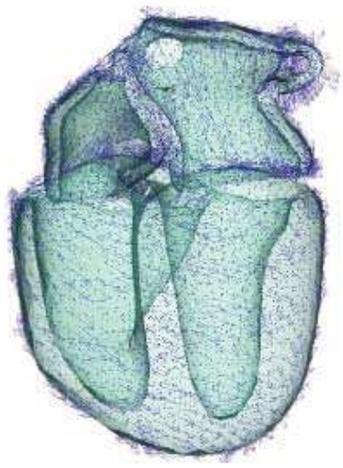


Fig. 25. Fiber orientation assigned to each node of the tetrahedral ventricular model for four different views. The atria and ventricles are displayed in a transparent style.

findings described in literature Greenbaum R. A. et al. (1981); Mc Veigh E. et al. (2001); Nielsen P. M. F. et al. (1991); Rijcken J. et al. (1999); Streeter Jr. D. D. et al. (1969). Figure 25 gives an impression of the ventricular fiber structure used in this study.
Another essential input for the CA are the effective refractory periods *ERP* defined for a heart cycle length (CL) of 700 ms (termed ERP_{700}). These can be adjusted for each tetrahedron individually or for tetrahedrons belonging to one type of tissue. The effective refractory period is defined as the time during which the cell cannot be excited by a stimulus of such magnitude, which is twice as high as a stimulus (diastolic threshold) capable to excite a cell in

resting status. The ERP is computed during run time, as the values depend on each preceding diastolic interval and on the current CL (detailed explanation can be found in Killmann R. (1990); Wach P. et al. (1989)). It should be stated that the CL in this context is the time distance between two proceeding excitation processes in one tetrahedron and does not necessarily refer to a CL provoked by any pacemaker (e. g., sinus node, atrioventricular node or paced rhythms). The run time ERP at the CL is calculated by

$$ERP(CL) = ERP_{700} + A + B \cdot \frac{CL - 700}{1000} - D \cdot \left(\frac{C - CL}{C}\right)^2$$

if $CL < C$ and

$$ERP(CL) = ERP_{700} + A + B \cdot \frac{CL - 700}{1000} \tag{10}$$

if $CL \geq C$, respectively. Parameters A (offset value), B (slope of the function), C (a predefined CL), and D (coefficient for the quadratical decrease) have to be defined for each type of tissue individually. The values for these parameters were chosen for each tissue type according to Killmann R. (1990). ERP_{700} was set to 210 ms for the bundle of His, 290 ms for the conjunction points, 280 ms for the left and right bundle branches, 320 ms for the Purkinje fiber network and 260 ms for the ventricular myocardium. Parameter A is introduced for enabling a simple change from physiological to pathological conditions for each tissue type individually. The value for the slope parameter B was obtained by averaging data reported for human hearts. Parameters C and D were set unequal zero for the ventricular epi-, endo- and myocardium only as the relationship between CL and refractory periods shows up to have a quadratically decreasing shape for these ventricular tissue types Killmann R. (1990) for the case $CL < C$. If $CL \geq C$, the $ERP(CL)$ is assumed to increase linearly with the CL.

Computation of activation times

After selecting an arbitrary number of tetrahedrons assigned with an arbitrary time instant for starting the excitation process the activation sequence is computed as follows. Every tetrahedron of the cardiac model can take up three different states: *excitable* (***e***) , *refractory* (***R***; i. e., excited) or *waiting* (***W***; i. e., awaiting excitation). The last state has no physiological meaning but was introduced due to algorithmic needs.

The simulation starts by selecting that tetrahedron with earliest excitation time and its status is changed from ***e*** to ***R***, i. e., the number of this tetrahedron is written into table ***R***. The 'possible excitation times' of the excited tetrahedron's neighbors are calculated according to the distance between the center of masses and conduction velocities for each connection and stored in table ***W***. In case no conduction between two neighboring tetrahedrons can occur the conduction velocity is set to zero. Whether a conduction between the tissues the tetrahedra are belonging to can or cannot occur is stored in an additional parameter set. In the next step the two tables ***S*** and ***W*** are searched for the tetrahedron with the lowest excitation time. This is the next tetrahedron to be excited and stored in table ***R*** and the corresponding table ***W*** is cleared. If two or more tetrahedrons have the same starting or possible excitation time one of them is arbitrarily chosen and the other one becomes/the other ones become the source tetrahedron of the subsequent calculations. The propagation is then calculated as described above, but now three cases may occur: The neighboring tetrahedron is

- in status ***e*** $\rightarrow$ the possible excitation time is stored in table ***W***, i. e., the tetrahedrons's status is set to 'awaiting excitation',
- already in waiting status and has been assigned a possible excitation time with

  - a lower value than the one calculated this time → no changes occur,
  - a higher value than the one calculated this time → the possible excitation time is changed to the lower value,
- in status ***R*** → no value is then stored for this point in table ***W***.

Then the tables ***W*** and ***S*** are searched again and the sequence of instructions is performed for the next source tetrahedron. The duration of status ***R*** – the period of time a tetrahedron cannot be set into status ***W*** or ***e*** – is evaluated employing (10) each time a possible excitation is to be assigned to that tetrahedron. The computation is finished, when no more starting tetrahedrons in table ***S*** are left and ***W*** is empty or when the excitation time of the current source tetrahedron is higher than the chosen upper limit for simulation duration predefined by the user.

Computation of transmembrane potentials

For modeling different shapes of ventricular and/or atrial action potentials the extended Wohlfahrt formula described in Rosian M. (1991) is used:

$$\varphi_m(t-\tau) = \alpha(t-\tau) \cdot \beta(t-\tau) \cdot \gamma(t-\tau) + K_{10} \quad [\text{mV}]. \tag{11}$$

Parameter t is the current simulation time interval, τ represents the computed activation time, $\alpha(t-\tau)$ [–] describes the shape of the depolarization, $\beta(t-\tau)$ [mV] the phase from the beginning repolarization (enables modeling of a notch right after onset of depolarization) to the plateau shape, and $\gamma(t-\tau)$ [–] characterizes the repolarization process. The value is 0 for all parameters of (11), when $(t-\tau) < 0$. Time $t = 0$ is considered as onset of depolarization (time instant, when half of the depolarization amplitude is reached). With the extended Wohlfahrt formula (11) the time dependent transmembrane potential shape for each node of a cardiac tetrahedron can be calculated based on parameters specifically adjusted for the types of cardiac tissue and based on the 'history' of the preceding activation sequence (relaxing phase, diastolic interval) and the computed activation times.

4.2.3 Extracellular potential computation

Based on the quasi static approximation of Maxwell's equations for electromagnetic field calculation and employing the bidomain theory Geselowitz D. B. & Miller 3rd W. T. (1983) (therefore the subheading extracellular potential computation), the resulting differential equations to be solved are

$$div\left[\boldsymbol{\kappa}_b \, grad\left(\varphi\right)\right] = -div\left[\tilde{\boldsymbol{\sigma}}_{in} \, grad\left(\varphi_m\right)\right] \tag{12}$$

for the cardiac region and

$$div\left[\boldsymbol{\kappa}_c \, grad\left(\varphi\right)\right] = 0 \tag{13}$$

for all other compartments of the VCM. The tensor $\boldsymbol{\kappa}_b$ is the bulk conductivity, i. e., the sum of the electrical effective extracellular and effective intracellular conductivity $\tilde{\boldsymbol{\sigma}}_{in}$. The potential φ describes the extracellular, φ_m the transmembrane potential. The tensor $\boldsymbol{\kappa}_c$ holds the electrical conductivities for all other compartments c. For the ventricular electrical anisotropic conductivities, for the other compartments isotropic conductivities were assumed Bradley et al. (2000).

Applying the FEM, considering the volume conductor model and the related boundary conditions Seger M. et al. (2005) the equations (12) and (13) form together a system of algebraic equations

$$\boldsymbol{R}\boldsymbol{\phi} = \boldsymbol{S}\boldsymbol{\phi}_m. \tag{14}$$

The matrices $\boldsymbol{R}$ and $\boldsymbol{S}$ are the so-called *stiffness matrices*, the matrix ϕ describes the potentials in all nodes of the tetrahedral mesh of the volume conductor, the matrix ϕ_m contains the transmembrane potentials in all source nodes of the ventricles. For forward simulation, the matrix ϕ_m is computed by the CA for discrete time steps. For inverting $\boldsymbol{R}$ on the left hand side of equation (14), the matrix $\boldsymbol{R}$ has to be modified as this matrix is positive *semi*-definite due to the *Neuman's boundary condition* on the torso surface Fischer G. et al. (2002). Therefore, the *Wilson central terminal* is used to define the reference potential on the torso surface Fischer G. et al. (2002) to reveal a positive definite matrix $\tilde{\boldsymbol{R}}$. The inversion of $\tilde{\boldsymbol{R}}$ is performed by a solver based on the *conjugated gradient* method. This consequently leads to the desired potentials in all nodes of the volume conductor model

$$\phi = \tilde{\boldsymbol{R}}^{-1}\boldsymbol{S}\phi_m. \tag{15}$$

Apart from scaling of matrix $\tilde{\boldsymbol{R}}$ no additional preconditioning was performed.

4.3 Results

The environment for simulating the potential data and for visualizing the ECG or BSP maps is based on amiraDev™ (TGS Europe Inc.), which was extended implementing the described functionality using the plugin concept. Thus, a homogenous and user-friendly simulation toolbox could be implemented.
Figure 26 depicts a normal sinus rhythm, whereas in figure 27 an extra stimulus between the right lower and upper pulmonary vein was simulated.

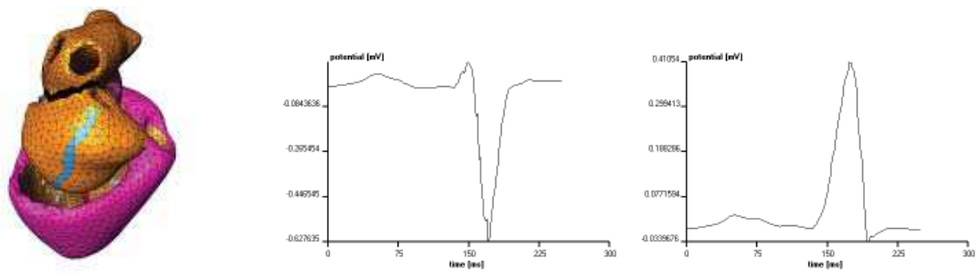


Fig. 26. ECG pattern simulation of a normal sinus rhythm. The gray boxes in the green area (sinus node) mark the tetrahedron where the stimulation starts from. The second figure shows the V3 and the third figure the V5 lead.

By varying the compartment specific parameters (intra- and extracellular conductivities, transmembrane shape) and by specifying the starting point and time any simulation can be performed and used for better understanding the ECG. Furthermore, the potential can be visualized, which also contributes in a deeper understanding of electrical propagation in the heart and the composition of the ECG patterns.

4.4 Discussion

We presented an in silico model environment for the simulation of cardiac de- and repolarization and of the three-dimensional potential pattern throughout the entire volume conductor. A cellular automaton and a bidomain-theory based source-field numerics are the fundamental basics. Only a few simulation scenarios have been presented in this paper. Limitations are given because of the relative course spatial discretization and because of not considering heart muscle contraction. Hence, microscopic cardiac propagation effects can not be simulated. Because we mostly had interest in investigating the macroscopic source-field

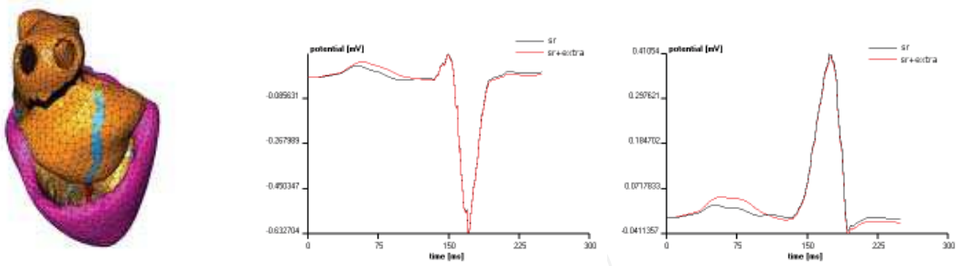


Fig. 27. Simulation of a sinus rhythm with an extra stimulus (staring at 0 ms) between the right lower and upper pulmonary vein. As expected when stimulating the atria at an ectopic focus, the P wave differs compared to the one of the sinus rhythm. This can be seen when comparing the normal ECG (black) with the extra stimulated one (red).

relationship, this limitation can be argued quite well. Because not modelling contraction the simulated T-wave patterns have to be considered as fully synthetic. The simulation of more realistic T-wave patterns have to consider contraction because of the electrical anisotropy and the associated movement of the electrical sources during contraction. Also, we did not consider the very complex fibre architecture in the atrium. Instead of that, for simplicity, we considered electrical isotropy throughout the atrial myocardium. Beside these various limitations, the presented in silico cardiac modelling solution enables various applications for the study of the nature of the ECG pattern in space and time.

5. References

Barrett, C., Eubank, S. & Smith, J. (2005). If Smallpox strikes Portland, *Scientific American* .

Beauchemin, C., Samuel, J. & Tuszynski, J. (2004). A simple cellular automaton model for influenza a viral infections., *J. Theor. Biol.* 232(2): 223–34.

Bossel, H. (1992). Modellbildung und Simulation, *Vieweg Verlag* .

Bradley, C., Pullan, A. & Hunter, P. (2000). Effects of material properties and geometry on electrocardiographic forward simulations, *Annals of Biomedical Engineering* 28(7): p 721–741.

Castiglione, F., Duca, K., Jarrah, A., Laubenbacher, R., Hochberg, D. & Thorley-Lawson, D. (2007). Simulating epstein-barr virus infection with c-immsim., *J Theor Biol* .
URL: *http://bioinformatics.oxfordjournals.org/cgi/content/abstract/btm044v1*

Eubank, S., Guclu, H., Kumar, V. A., Marathe, M. & et al. (2004). Modelling disease outbreaks in realistic urban social networks, *Nature* .
URL: *http://www.mcc.uiuc.edu/nsfitr04Rev/presentations/0113049_Modelling_disease.pdf*

Fischer, G. (2006). In silico cardiac modeling, *Habilitation* .

Fischer G., Tilg B., Wach P., Modre R., Hanser F. & Messnarz B. (2002). On modelling the wilson terminal in the boundary and finite element method, *IEEE Transactions on Biomedical Engineering* 49(3): 217–224.

Fuchsberger M. (1993). *Modellierung der Fasergeometrie in einem numerischen Herzmodell*, Master's thesis, Graz University of Technology. (in German).

Gamma, E. (1994). Design Patterns, *Addison-Wesley Professional - ISBN 0201633612* .

Geselowitz D. B. & Miller 3rd W. T. (1983). A bidomain model for anisotropic cardiac muscle, *Annals of Biomedical Engineering* 11(3–4): 191–206.

Greenbaum R. A., Ho S. Y., Gibson D. G., Becker A. E. & Anderson R. H. (1981). Left ventricular fibre architecture in man, *British Heart Journal* 45(3): 248–263.

Hammer B. (2001). *Optimisation of surface triangulations for image reconstruction from ecg mapping data*, Master's thesis, Graz University of Technology. (in German).

Hayn D. (2002). *Ein finite Elemente Herz Modell*, Master's thesis, Graz University of Technology. (in German).

John von Neumann (1966). The theory of self-reproducing automata, *University of Illinois Press* .

Killmann R. (1987). *Numerisches Herzmodell*, Master's thesis, Graz University of Technology. (in German).

Killmann R. (1990). *Three-dimensional numerical simulation of the excitation and repolarisation process in the entire human heart with special emphasis on reentrant tachycardias*, PhD thesis, Graz University of Technology.

Mc Veigh E., Faris O., Ennis D., Helm P. & Evans F. (2001). Measurement of ventricular wall motion, epicardial electrical mapping and myocardial fibre angles in the same heart, *in* Katila T. & Neonen J. (eds), *Functional Imaging and Modeling of the Heart*, pp. 76–82.

Nielsen P. M. F., LeGrice I. J., Smaill B. H. & Hunter P. J. (1991). Mathematical model of geometry and fibrous structure of the heart, *American Journal of Physiology* 260(4.2): H1365–H1378.

NY (1977). Numerical methods for partial differential equations., *New York: Academic Press* .

Rijcken J., M., B., Schoofs A. J., van Campen D. H. & Arts T. (1999). Optimization of cardiac fiber orientation for homogeneous fiber strain during ejection, *Annals of Biomedical Engineering* 27(3): 289–297.

Rosian M. (1991). *Modellierung der Aktionspotentialformen im menschlichen Herzen*, Master's thesis, Graz University of Technology. (in German).

Seger M., Fischer G., Modre R., Messnarz B., Hanser F. & Tilg B. (2005). Lead field computation for the electrocardiographic inverse problem – finite elements versus boundary elements, *Computer Methods and Programs in Biomedicine* 77(3): 241–252.

Shannon, C. E. (1948). A Mathematical Theory of Communication, *Bell System Technical Journal* vol. 27: 379–423.

Stephen Wolfram (2002). A new kind of science, *B & T* vol. 1.

Streeter Jr. D. D., Spotnitz H. M., Patel D. P., Ross Jr. J. & Sonnenblick E. H. (1969). Fiber orientation in the canine left ventricle during diastole and systole, *Circulation Research* 24(3): 339–347.

Volterra, V. (1926). Fluctuations in the abundance of a species considered mathematically, *Nature* 118: 558–560.

Wach P., Killmann R., Dienstl F. & Eichtinger C. (1989). A computer model of human ventricular myocardium for simulation of ecg, mcg, and activation sequence including reentry rhythms, *Basic Research in Cardiology* 84(4): 404–413.

Wurz, M. & Schuldt, H. (n.d.). Dynamic Parallelization of Grid-Enabled Web Services, *European Grid Conference, Amsterdam* .

Xiao, X., Shao, S.-H. & Chou, K.-C. (2006). A probability cellular automaton model for hepatitis b viral infections., *Biochem. Biophys. Res. Commun.* 342(2): 605–10.

Cellular Automata - Simplicity Behind Complexity

Edited by Dr. Alejandro Salcido

ISBN 978-953-307-230-2
Hard cover, 566 pages
Publisher InTech
Published online 11, April, 2011
Published in print edition April, 2011

Cellular automata make up a class of completely discrete dynamical systems, which have became a core subject in the sciences of complexity due to their conceptual simplicity, easiness of implementation for computer simulation, and their ability to exhibit a wide variety of amazingly complex behavior. The feature of simplicity behind complexity of cellular automata has attracted the researchers' attention from a wide range of divergent fields of study of science, which extend from the exact disciplines of mathematical physics up to the social ones, and beyond. Numerous complex systems containing many discrete elements with local interactions have been and are being conveniently modelled as cellular automata. In this book, the versatility of cellular automata as models for a wide diversity of complex systems is underlined through the study of a number of outstanding problems using these innovative techniques for modelling and simulation.

How to reference

In order to correctly reference this scholarly work, feel free to copy and paste the following:

Bernhard Pfeifer (2011). Biophysical Modeling using Cellular Automata, Cellular Automata - Simplicity Behind Complexity, Dr. Alejandro Salcido (Ed.), ISBN: 978-953-307-230-2, InTech, Available from: http://www.intechopen.com/books/cellular-automata-simplicity-behind-complexity/biophysical-modeling-using-cellular-automata

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen