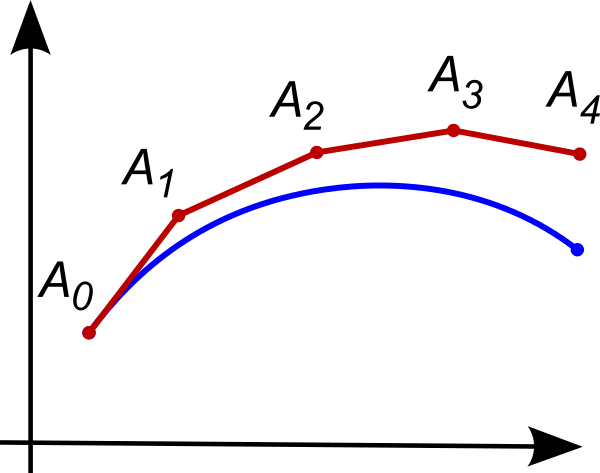
## Chapter 2 - Euler method

1. Introduction

In this chapter we will introduce the method used for solving the differential equations used in the SIR model. We will start by familiarizing you with the Euler’s method and then explaining in detail how we used it to compute the epidemic spread.

Euler’s method is one of the simplest numerical procedures used for solving ordinary differential equations given an initial value. It is also first order method, which means that the global error is proportional to the step size. The method is named after Leonhard Euler and is often used in the construction of more complex and accurate procedures.

Figure 1

2. About the method

The basic idea is that knowing an initial value you can approximate the curve of the function satisfying the equation. You start by calculating the gradient with the initial value and making a step (dt) using the slope. This gives you a second, approximated point that is on the curve of the solution. You can now using this new point calculate the next. Doing this iteratively you accumulate set of points approximating the curve you are looking for.

The main problem of the method is the level of approximation. Figure 1 shows how the error is accumulated as the step size and the number of iterations are increasing.

The smaller the step you are using the greater the accuracy but as the steps is getting smaller the computing power you need to invest to calculate the same interval of the curve becomes greater. There is an inherit tradeoff between the level of approximation and the number of iterations you need to run, so you need to find a balance there.

3. Illustrating the Euler method with an example

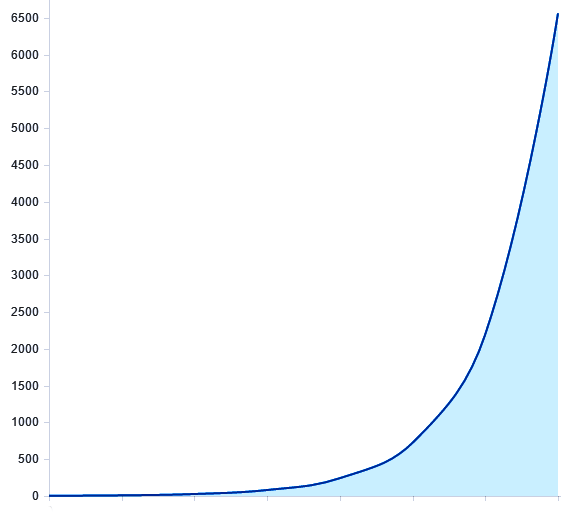
Let’s say we have a differential equation dF = 2\*dF\*dt and we have an initial point F(0) = A0. We can now find an approximate second point by calculating A1 = A0 + 2\*A0\*eps, where eps is the level of approximation. After computing A1 we can calculate A2 = A1 + 2\*A1\*eps. Now we see the pattern and generalize the idea with the following recursive relationship which can then be plotted (see Figure 2).  
  
A(n) = A(n-1) + 2\*A(n-1)\*eps  
A(0) = 1

Figure 2

4. Euler’s method on the SIR Model

In the case of the SIR model we programmed the following recursive relationship we deduced using the system of Kermack-McKendrick’s model:

S(n) = S(n-1) – beta \* S(n-1) \* I(n-1) \* dt

I(n) = I(n-1) + beta \* (S(n-1) \* I(n-1) – alpha \* I(n-1)) \* dt

R(n) = R(n-1) + alpha \* I(n-1) \* dt

This recursive system comes directly from the way we solve differential equations using the Euler method. Using this relationship between the S, I and R groups we were able to iteratively evaluate the number of people in each of them and make a prediction for each day of haw many people were susceptible, infected and removed.

5. Euler’s method on the SAIC Model

Analogously to the SIR Model for the SAIC model we implemented the following recursive system.

S(n) = S(n-1) – beta \* S(n-1) \* I(n-1) \* dt

A(n) = I(n-1) + beta \* (S(n-1) \* I(n-1) – alpha \* I(n-1)) \* dt

I(n) = R(n-1) + alpha \* I(n-1) \* dt

C(n) = R(n-1) + alpha \* I(n-1) \* dt

## Chapter 3 – The modeling software

### In the following part of the report we will introduce you to the software for modeling we wrote.

1. Introduction

As we were researching different papers wrote on the subject of modeling an epidemic spread we noticed that a big number of them were following a particular pattern. What we saw was that in almost all of the cases there were groups of people separated by a status and equations modeling the way the people in the different groups interact and how they transfer from one group to another. This made us realize that this level of abstraction can be programmed and wrapping it in a user friendly GUI would make the process of testing and trying an idea for a model much more approachable.

2. Description of the software

With the software you can easily create nodes or groups of whatever and assign those names and initial quantities. Then you can add parameters to your model. Those are just key value pairs that you can then use in the descriptions of the relationships between the nodes. They can also be changed as the model is running so you can closely observe the way they influence your system and how important are they for the model.

After that you can connect the nodes with oriented edges and for each of those edges you can write a formula describing the interaction between the nodes. Every time the whole system is run the values of these formulas are calculated and the results are transferred according to the directions of the connections.

3. Example with the SIR Model

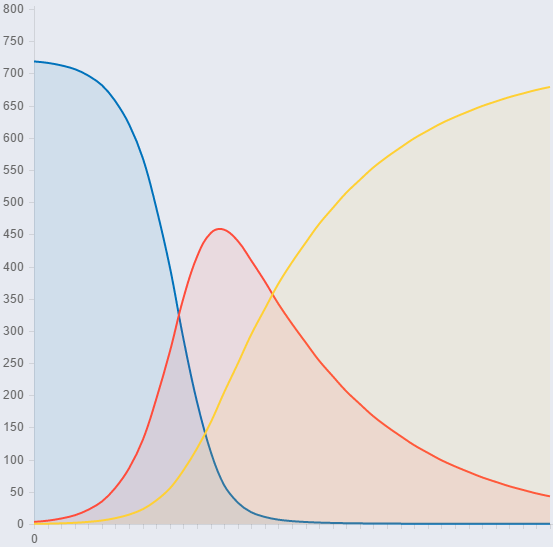
Let’s take the SIR Model for example. If you want to implement it using our software you should create three node with names S, I, and R and quantities 720, 2, 0. Then you should create the parameters alpha and beta with values 0.001 and 0.1. After that you should connect S to I and I to R. The formulas in the edges you created are accordingly S\*I\*alpha and I\*beta.

Figure 3

Running this configuration will give you the well-known graphic we get when we plot the SIR Model.   
(See Figure 3)

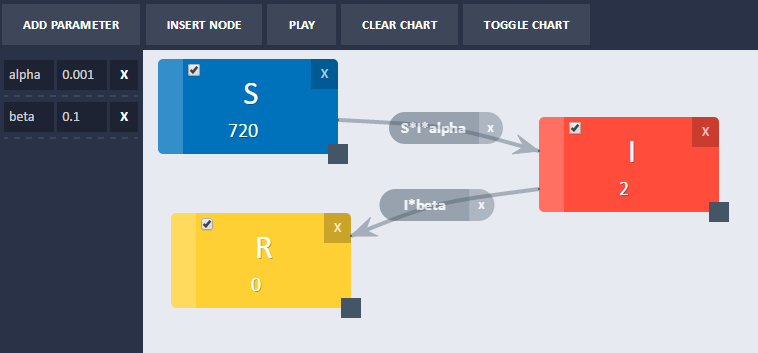
Figure 4 shows the interface of the software and the configuration we used to implement the SIR Model. 

Figure 4

4. Problems along the way

One thing that took us a quite a while to implement was the user interface. For the interactivity of the nodes and the rendering of the graphic we used two libraries that we researched – JSPlumb and ChartJs.

In later development we added the ability to exclude some nodes from the graphic, because of a problem we stumbled upon as we were trying to implement a more sophisticated model.

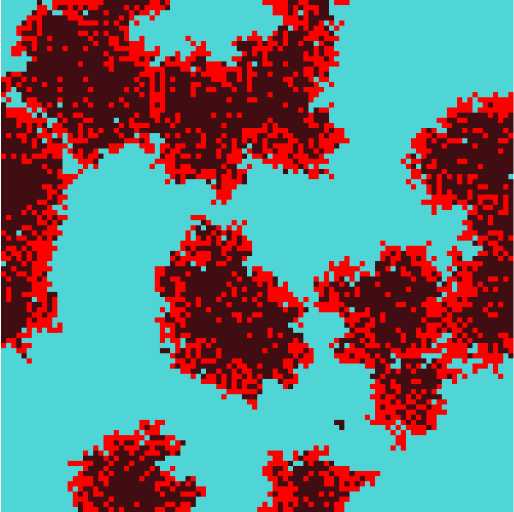
The problematic model was an upgrade to the SIR model. The only difference is that it adds birthrate and mortality to the system. You can easily implement birthrate by adding an influx node and setting its quantity to Infinity which is something the software is supporting but if you then try to run the model the plot is rendered empty, because it is trying to scale and plot points at infinity.

You can try the software by following this link - <https://goo.gl/rN1l2h>.

## Chapter 4 – Cellular Automata

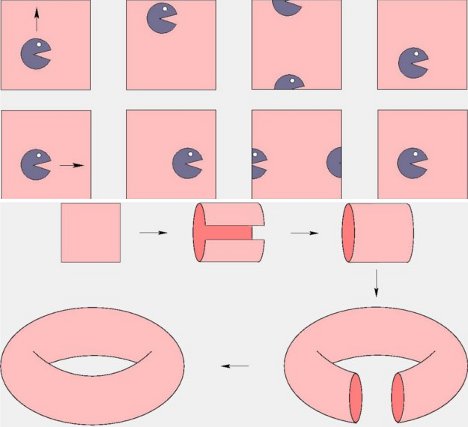
### In this chapter we will introduce you to the idea of modeling an epidemic spread using a cellular automaton.

1. Introduction

Cellular automaton models consists of cells on a regular grid and simple rules governing the states of these cells. These types of models can be used in all kinds of fields and are heavily studied by mathematicians, physicians, theoretical biologists and others. One common example of a cellular automaton you are probably familiar with is the famous [Conway’s Game of Life](https://en.wikipedia.org/wiki/Conway%27s_Game_of_Life).

What cellular automatons give us more than the models we introduced so far is the neighborhood effect, which is the way the neighbors of a cell can influence the state of this cell. This type of model can give us more accurate visual representation of the spread of a disease and help us betted understand and prevent outbreaks.

2. The neighborhood

In the model we are rendering each cell on a square grid and the color of the cell represents its state. The “world” of the cells is flat torus. That is to say that the neighbors of the cells on the edges of the grid are the cells on the other side of the grid (See Figure 5).

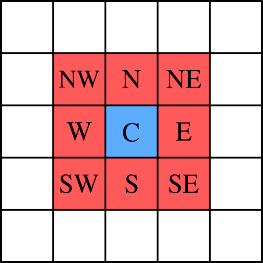
The neighbors of each cell are the eight cells around it, which is also known as [Moore neighborhood](https://en.wikipedia.org/wiki/Moore_neighborhood) (See Figure 6).

Figure 6

This means that every iteration of the model the cells status depends only of the current status it has and the status of the eight cells in its neighborhood.

Figure 5  
Source - http://gajitz.com/this-is-an-impossible-to-visualize-3d-mathematical-shape/

3. The model  
Source - <http://csc.ucdavis.edu/~chaos/courses/nlp/Projects2008/SharonChang/Report.pdf>

The model we implemented is adaptation of the SIR Model for cellular automatons based on probabilities. In each step for every cell is calculated probability for the cell to become infected (Pi) if the cell is susceptible and probability for the cell to become removed (Pr) if the cell is infected. Then we compare this probability to a pseudo random number and change the status of the cell accordingly.

Pi = 1 – (1 – p) R  
Pr = q

Where p (rate of infection) and q (rate of removal) are the parameters of the model between 0 and 1. R is the number of infected cells in the Moore neighborhood of a particular cell. The formula for Pr is pretty much self-explanatory whereas the explanation for the value of Pi is as follows – as the rate of infection increases, 1 - p decreases and the bigger R the smaller 1 - p is which makes Pi grow as the number of infected cells around it grow.

The model has a third parameter n (immunity period), which is the number of iterations removed cell goes through before becoming susceptible again.

4. Implementation

With the software we wrote you can run the mode and change the parameters in real time. You can reset it and run it step by step. There is also a graphic showing the number of people in the different groups (See Figure 7). You can play with the software [HERE](https://rawgit.com/ichko/SirModel/master/src/cellulara-automata/index.html).

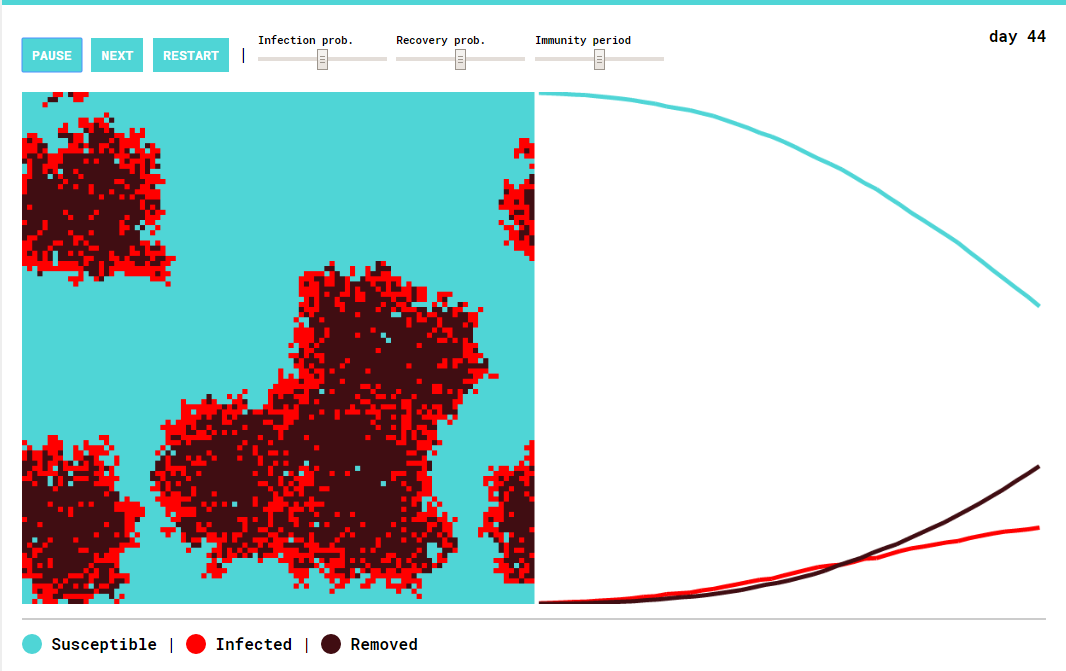


Figure 7

4. Results

Running the model several times with different properties we made some conclusions.

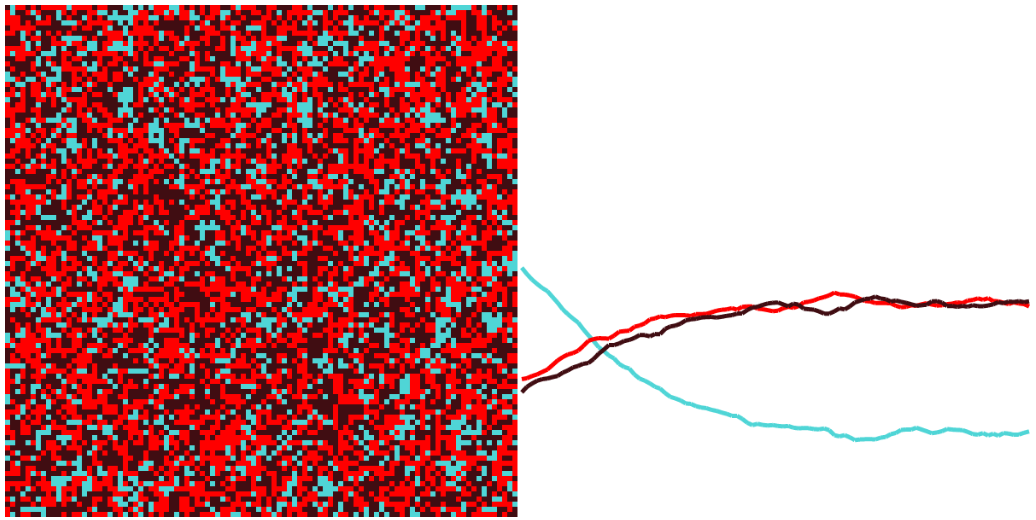
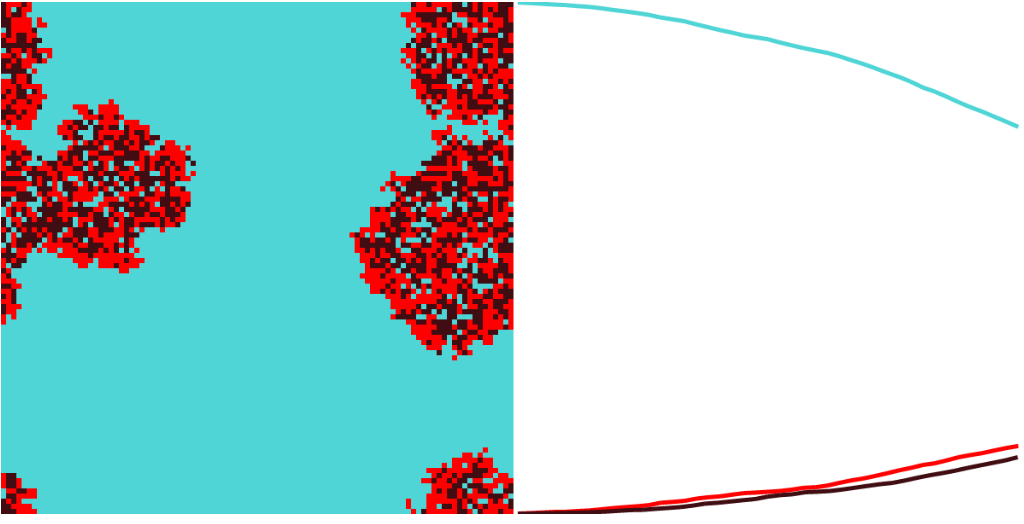
4.1 Running the model with low immunity period (10 days) lead to a system ending in an equilibrium. The number of people in the different groups stabilizes after a certain point and remains relatively the same (See Figure 8).

Figure 8 – 160 days after the initial state



30 days after the initial state

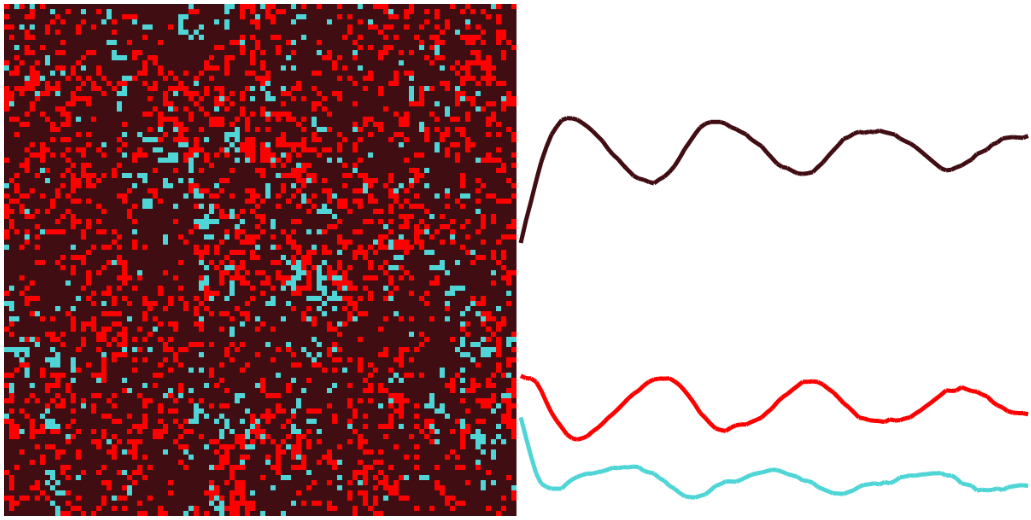
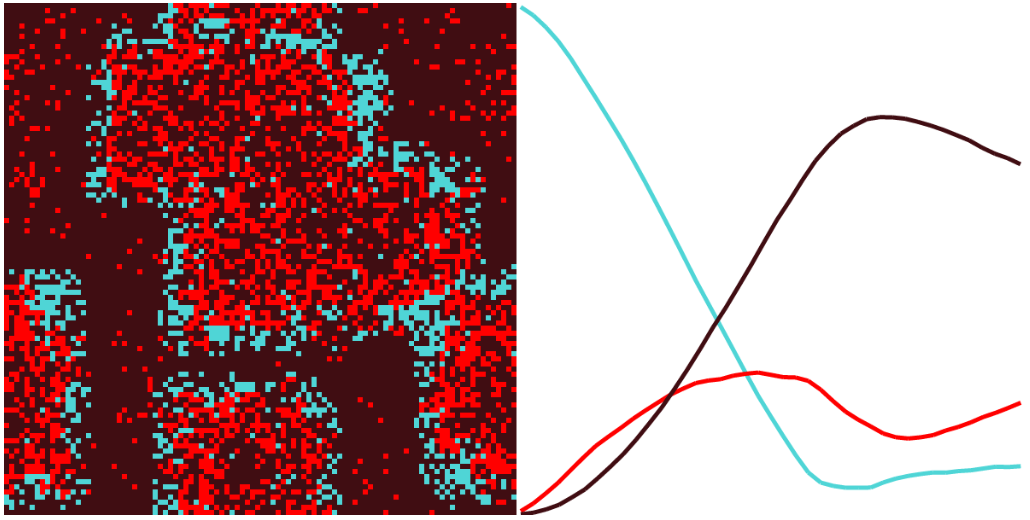
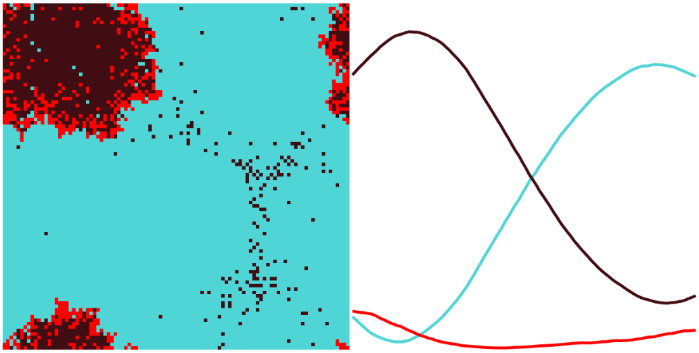
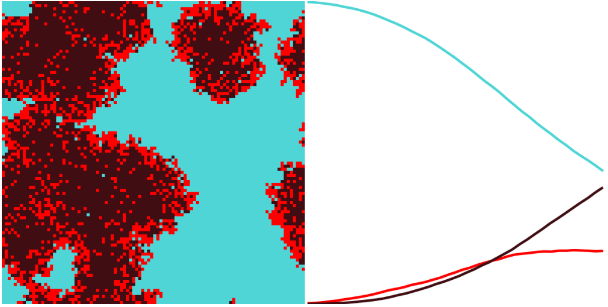
4.2 Running the model with high infection rate (0.5), relatively low removal (0.1) rate and average immunity period (25 days) leads to short term seasonality, which you can see on Figure 9.

Figure 9 – Long after the initial state

4.3 Running the model with low infection rate (0.1), low removal rate (0.1) and high immunity period (70 days) leads to a system with long seasonality.

30 days after the initial state

160 days after the initial state

Short after the initial state