Teaching Mathematics and Statistics in Sciences - IPA HU-SRB/0901/221/088
Interesting Mathematical Problems in Sciences and Everyday Life - 2011

Mathematical and Simulation Models in the *AnyLogic* program

***Arpad Takači*[1], *Dušan Mijatović*[1],**

[1] ***Department of Mathematics and Informatics,***

Faculty of Sciences,

University of Novi Sad

takaci@dmi.uns.ac.rs

1. Lorenz's weather model

Lorenz's mathematical model of the weather, made in 1963, describes the motion of the air through the atmosphere. More precisely, Lorenz observed the hot air rising up into the atmosphere, then cooling down and eventually falling to the ground. While observing the weather conditions, Lorenz noticed that the weather does not always behave in a foreseen manner. In his attempt of introducing a mathematical model, by following the patterns of the weather conditions, he discovered that the obtained model is of *chaotic behavior*. Actually, the model he introduced was a nonlinear system of differential equations with three unknown variables.

At the beginning, Lorenz was studying a system of 12 equations, and soon he learned that small changes of the variables' initial conditions make large changes in the behaviour of the model. This sensitivity of the initial conditions of the model nowadays is called the *Butterfly effect*. One of the conclusions from this analysis is that all weather forecasts, for a period longer than a week, most often prove themselves wrong.

From the technical point of view, the Lorenz's oscillator is nonlinear, treedimensional and deterministic. Let us remark that still today this system is used for the demontsration of a system with chaotic behavior. Next we give Lorenz's system of ODEs with the unknown functions x, y i z, depending on time t

$$\begin{aligned}\frac{dx}{dt} &= ay - ax \\ \frac{dy}{dt} &= rx - y - xz \\ \frac{dz}{dt} &= xy + bz\end{aligned} \tag{1.1}$$

The physical explanation of the unknowns is as follows. The function x represents the proportional speed of the air motion as a result of the convection. The function y represents the value of the temperature difference between the hot air moving up, and the cold air falling down.The function z is the value of the vertical temperature difference in the system from down to up.

The parameters a. b and r in (1.1) represent the essential part of the system, in view of its big influence on the system. The parameter a corresponds to Prandtl's number, which was obtained from the basic property of the observed air; its usual value is 10. The parameter b represents the observed area in the model; Lorenz took for b the value 8/3, i.e., 2.666.

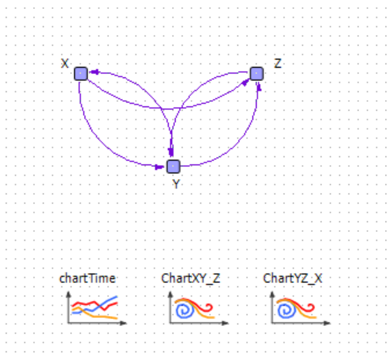
The parameter r is the Raylog's number, which defines the moment when the air convection starts in the system. This number, whose usual value is 28, is essential in the process of change in the system from stable into chaotic.

Depending on the value of the parameter r, the model has the following behaviors. If a=10, and b=8/3, then for $0<r<1$ the system is stable, while for $r>1$ it is unstable. If $1<r<24.74$, then the equilibrium points, denoted by c_1 and c_2 , given by,

$$c_1 = (\sqrt{b(r-1)}, \sqrt{b(r-1)}, r-1) \quad , \quad c_2 = (-\sqrt{b(r-1)}, -\sqrt{b(r-1)}, r-1)$$

are stable, while if $r>24.74$, then the points c_1 and c_2 become unstable, which means that the behavior of the whole system becomes unstable.

The model structure in the *AnyLogic* programme is very simple, since the mathematical expressions can be easily put into the DEs. In our case, the structure in *AnyLogic* gets the following form:



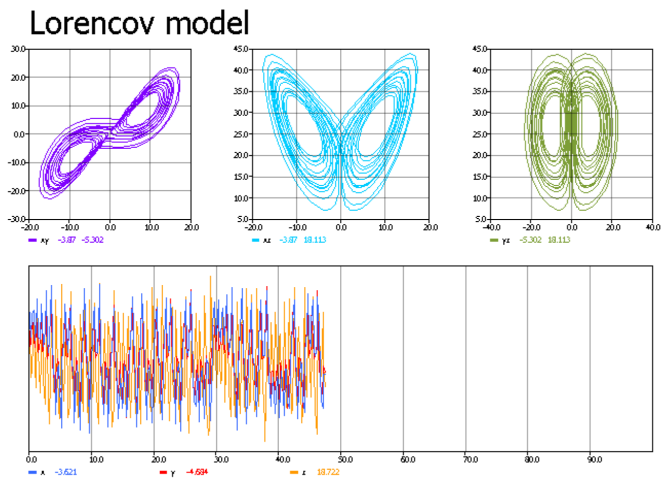


Figure 1.1. Results of the running the Lorenz model in the *AnyLogic* programme

2. The Kermack-McKendrick model (SIR model)

2.1. Introduction

The epidemic model, introduced by Kermack and McKendrick in the early years of the 20th century, is a practical and reliable system of equations for monitoring the spread of infectious diseases. This model was used for examining the spread of dangerous diseases such as plague. Although simple, the model is both complete and efficient.

The model is based on three variables representing a population of people divided into three groups: uninfected (*suspected*), infected, recovered and immune/dead (recovered, immune, and removed). It is important to note that the model was basically designed to monitor a limited population, where the number of persons in the course of the epidemic changes only slightly. In general, the model reffers to diseases that rarely have a fatal outcome, but, with minor modifications, it may be included in a model which allows changes in the size of the analyzed population.

Kermack-McKendrick model allows the inclusion of many features and characteristics of epidemic diseases by using appropriate parameters. Actually, it is these parameters that determine the behavior of the model, provided that the values for the parameters have been carefully selected and thoroughly tested.

2.2. The system of equations used to describe the epidemic

The system uses parameters that contain a variety of influences that essentially affect the progress or prevent an epidemic. We denote with a the group of parameters that influence positively the epidemic, and with b the group of parameters that reduce the epidemic. Further on, with S we denote the uninfected group, with I the first group of infected individuals, and with *R* the recovered ones. The original system of equations of the Kermack-McKendrick model (see [10]), is given in equation (2.1):

$$\begin{aligned}\frac{dS}{dt} &= -aSI \\ \frac{dI}{dt} &= aSI - bI \\ \frac{dR}{dt} &= bI\end{aligned} \tag{2.1}$$

2.3. Analysis of the system (2.1)

From the first two equations one can conclude that the number of uninfected individuals that will become infected is proportional to the number of contacts between people from the group *S* and group *I*, both under the assumption that the number of contacts depends only on the number of persons in each group, i.e., there is a uniform mixing in the population. The third equation is based on the assumption that the number of recovered persons is proportional to the number of infected ones, which represents an average time that people spend in a state of infection until you cross the state when they can neither transmit the infection, nor can they become infected.

The assumption that there is a constant number of persons in the population, implies the following simple equation:

$$S(t)+I(t)+R(t)=N(t),$$

where *N=N(t)* is the total number of the population. Note that the rate of increase of infected individuals is given by $R_0 = a/b$. Firstly, we start with the equation

$$S(0)+I(0)=N(0).$$

Of course, there must be at least a few infected people, i.e, $I(0)>0$. We are interested only in those solutions, where *S*, *I* and *R* are non-negative, which means that it holds $\frac{dS}{dt}<0$ when there are people who are infected and those that were infected. Since *S* decreases as time goes by:

$$S(t)<S(t_1)<S(0), \text{ for } t>t_1>0.$$

In other words, the value of *S* is constantly decreasing and *S* must be nonnegative, meaning that when $t\to\infty$, *S* must have a threshold, which may be 0, i.e,

$$S(\infty)=\lim_{t\to\infty} S(t).$$

From the second equation we have $\frac{dI}{dt}<0$, privided that *aS*<*b*. Since *S* decreases as time passes, it follows that *aS*(0)<*b* and $\frac{dI}{dt}<0$ for each *t*>0, implying that in this case the epidemic disappears. There will be no epidemic, unless a critical value, *b/a,* is reached by the initial population of uninfected. This value is small if *b*<<*a*, which means that the immunity stops the spread of the disease.

The number of recovered, *R=R(t),* from the third equation, monotonically increases, but since $R(t) \le N$, the limit $R(\infty) = \lim_{t\to\infty} R(t)$ exists. Next, the limit $I(\infty) = \lim_{t\to\infty} I(t)$, also exists, and the expression (*I*(∞) - *R*(∞)/*N*) shows the *strength* of the epidemic that effected the population. It is now necessary to determine these limits.

From the first and the third equation in (2.1) it follows

$$\frac{dS}{dR} = -aS/b\,,$$

Hence

$$S = S(0)\exp(-aR/b)\,.$$

From $R \le N$ it follows that $S \ge S(0)\exp(-aN/b)$ and $S(\infty) > 0$. This means that there will always remain a number of uninfected. In fact, several persons in the population will not get sick until the end of the epidemic.

$$\frac{\frac{dI}{dt}}{d\left(\frac{dS}{dt}\right)} = -1 + \frac{b}{aS}\,.$$

From the equality $S(0) + I(0) = N$ (0) it follows that

$$I = N - S + \left(\frac{b}{a}\right)\ln\left\{\frac{S}{S(0)}\right\}$$

Clearly $S(t) - S(\infty) > 0$ and $I(t) \to 0, t \to \infty$. As a consequence of this analysis, we have $S(t) = S(0)\exp(-aRt/b)$ and the equality

$$S(\infty) = S(0)\exp(-a\{N - S(\infty)\}/b). \tag{2.2}$$

Note that equation (2.8) determines $S(\infty)$. The equation is satisfied only for one positive value of $S(\infty)$, less than *b/a*. Once we find $S(\infty)$, the limit $R(\infty)$ can be found from $R(\infty) = N - S(\infty)$ and the spread of the epidemics equals to $R(\infty)/N$.

2.4. A Vaccination Model

So far we have worked with the assumption that only few people have natural immunity, and thus their number can be ignored. We continue to assume this assumption, but we will include consideration of the case when the epidemic is spreading so fast that it is necessary to stop the spread the epidemic by vaccination.

To simplify the model, we assume that vaccination removes a person from *S* or *R* momentarily, and that there is a way to avoid vaccination of those who are infected (group *I*). Now, we have to introduce a new group, *V*, consisting of the vaccinated people. Thus we have the following system:

$$\begin{aligned} \frac{dS}{dt} &= -aSI - \alpha(t) \\ \frac{dV}{dt} &= \alpha(t) \end{aligned} \tag{2.3}$$

The first equation in the system (2.3) changes the first equation of the initial SIR model (see equation (2.1)). Function $\alpha(t)$ is the rate of vaccination.

It is not easy to choose the function representing the rate of vaccination. Namely, any vaccination program includes the cost of personnel commitment to conduct the vaccination, their efficient use of equipment and other resources.

Moreover, we must take into account the damage suffered by the whole society from the epidemic. It is desirable that the vaccination program is limited by the total number of infected at some time, and to keep the number of infected below a desired level. To ensure that no more than N_1 individuals from a population get the disease during the time interval $0 \le t \le T$, it is necessary that

$$R(T) + I(T) \le N_1.$$

To keep the number of infected under a certain value N_2 in the same period of time, we have to assume $\max\limits_{0 \le t \le T} I(t) \le N_2$.

The control of the epidemic in this way implies that $\alpha(t)$ should be chosen small enough in order to meet the limits given in the previous two inequalities, and thereby keep costs at a minimum. Note that we got a problem of dynamic programming.

2.5. Model parameters

The first step is to determine the formulas for the infection factor *a* and the disease withdrawal factor *b*. For simplicity, we use only the following four parameters, namely *d*, *c*, *p* and *n*, which have a significant impact on the system and on the behavior of the epidemic:

d: duration of illness
c: rate of contacts
p: probability of infection
n: total population.

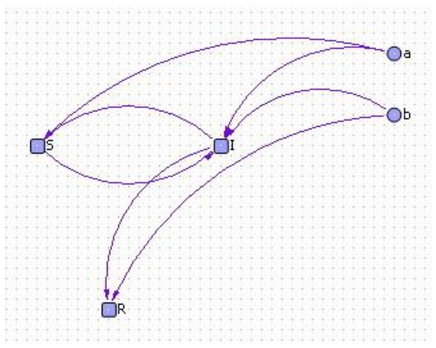
We will use the following formula for the parameter $a=cp/n$ which shows that the spread of the epidemic increases, whenever the rate of contacts occuring between people from the observed population increases, and the likelihood that the disease is transferred to direct contacts between the individuals. Note that the number 1/*n* multiplied by the number of uninfected, *S*, gives the infected part of the population.

The parameter *b* satifies the equality *b=1/d*, which shows that there is an inverse proportion between the softening of the epidemic, and the duration of the disease. In other words, the epidemic disease of shorter duration will end sooner. An example of this is the *Ebola* virus. The reason why this extremely infectious and fatal disease is not spread around the world that it that quickly kills its victims, the epidemic has very short duration, and it is unlikely that the disease will spread to a larger territory.

By testing the parameters, one can determine the parameter values appropriate for the required behavior of the model. For now on, we shall use the following values for the parameters: *c*=5, *p*=0.05, *n*=1000, *d*=15. Entering the equations in *AnyLogic*, one can construct the following simulation model:

$$\begin{aligned} \frac{dS}{dt} &= -aSI \\ \frac{dI}{dt} &= aSI - bI \\ \frac{dR}{dt} &= bI \\ a &= \frac{cp}{n} \\ b &= \frac{1}{d} \end{aligned} \tag{2.4}$$

The corresponding model structure in *AnyLogic* is given below:



By entering the initial values: *S*=999, *I*=1 and *R*=0, and running a simulation we obtain the following graph, where the number of uninfected is shown in blue, the number of infected in red, and the number of recovered in yellow.

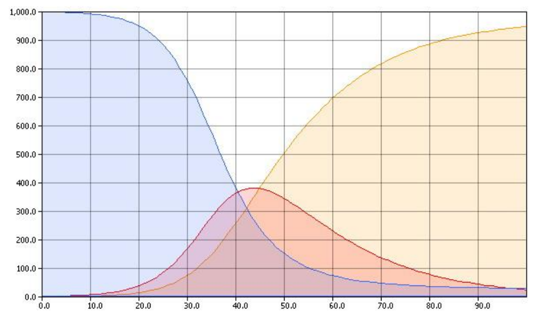
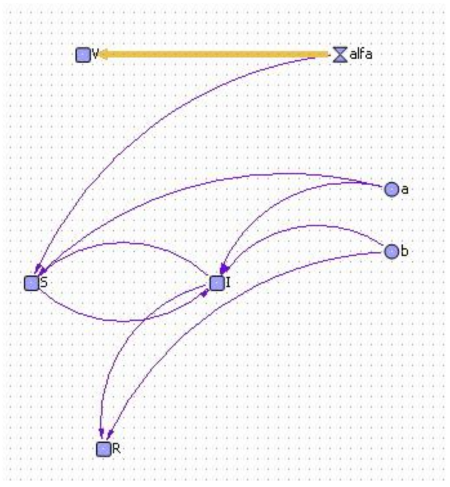


Figure 2.1. SIR model with vaccination

The vaccination should greatly reduce the number of infected in the system. In the model, we include vaccinations by changing the system of equations, compare (2,4) above and (2.5) below.

$$
\begin{aligned}
\frac{dS}{dt} &= -aSI - \alpha \\
\frac{dI}{dt} &= aSI - bI \\
\frac{dR}{dt} &= bI \\
\frac{dV}{dt} &= \alpha \\
a &= \frac{cp}{n} \\
b &= \frac{1}{d}
\end{aligned}
\tag{2.5}
$$

Model structure in *AnyLogic* is now different, as can be seen from the next figure.



Now, by running the simulation, we obtain the following graph:

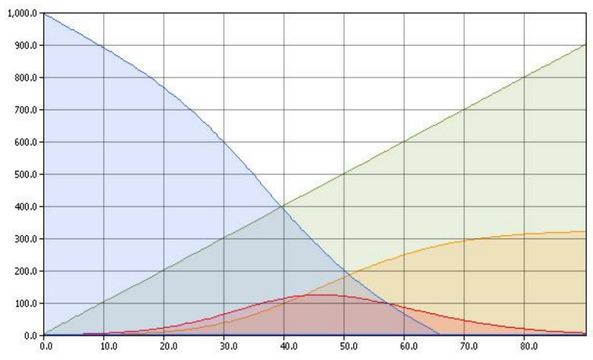


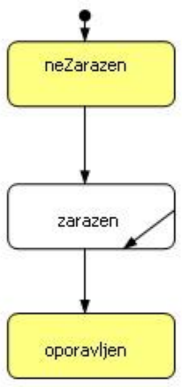
Figure 2.2. A smaller number of infected due to vaccination

Since the parameters in both instances were the same, one can compare the simulation results. Of greatest interest is the number of infected. In the first example, the number of infected has come to almost 400. In the model with vaccination only a small number of infected moved over 100.

2.6. The agent SIR model in AnyLogic

As mentioned in the previous text, each model created in system dynamics can be translated into an agent based model. To that end, first one has to create a new model and a new class of active objects called *Person*. In this class the state diagram should be inserted.

Next, a state diagram describing the behavior of an agent has to be made. Obviously, the diagram must have three states: uninfected, infected and recovered. The next figure shows the state diagram as created in *AnyLogic*.

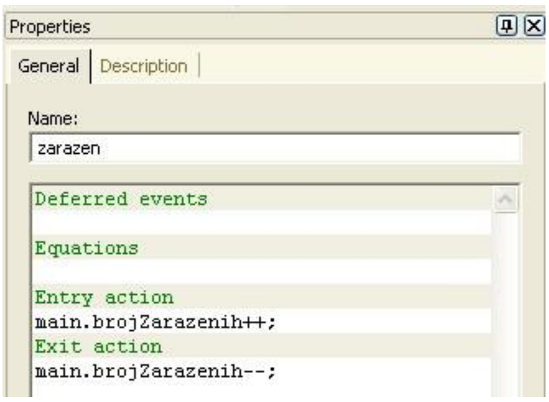


When a diagram is drawn, it should set the conditions for transitions between states, transition between uninfected and infected states will make over the signal events. If an uninfected agent gets the signal "contact", which means that there has been contact with an infected agent, then he switches to the state *infected.* Internal transition from the state *infected* with the ability of an infected agent sends a signal contact. We will assume that the rate of contact is as follows:

```
if (randomTrue(verovatnocaZaraze)) {
```

```
                Osoba neko=main.ljudi.random();
                neko.statechart.fireEvent("kontakt");
        }
```

The transition from *infected* to *recovered* state also happens with a rate. For the transition rates we take the value *1/trajanjeBolesti* (the duration of the disease). We also need to enter the input and output actions for each state. The variables *brojNeZarazenih* (number of suspected), *brojZarazenih* (number of infected) and *brojOporavljenih*(number of recovered) monitor the number of agents who are in these states, so that the input action in the state would be required to increase by one, and the output action to reduce the number of agents in this state. The picture shows how it was done for the state of infection. In the same way one can analyze the status of not infected and recovered.



Name:
zarazen

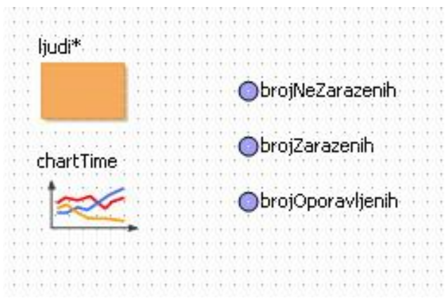
```
Deferred events

Equations

Entry action
main.brojZarazenih++;
Exit action
main.brojZarazenih--;
```

Returning to the diagram structure of class *Person*, there is a need to define the main variable of type *Main* and has an initial value *(Main) getOwner)*.

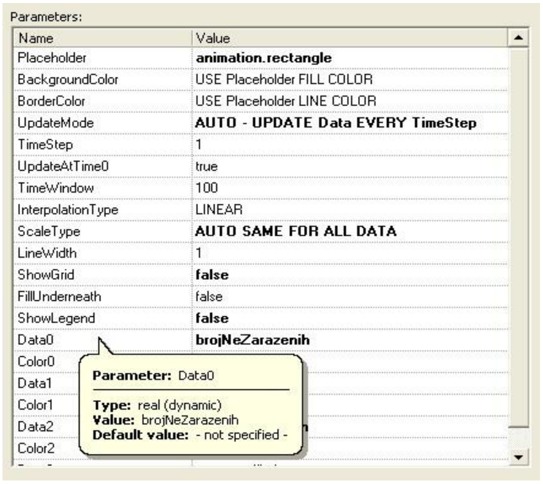
The class of persons should include the three parameters that are features of the agent. It is necessary to define the parameter *trajanjeBolesti* that type double and has an initial value of 15, *verovatnocaZaraze*, that type double and has an initial value of 0.05, and a parameter of type double *stopaKontakata* with an initial value 5. This defines the class *Person.* The diagram of the class structure *Person* has to be inserted into the *Main* class by dragging the mouse. In the diagram of the structure *Main* there will appear an orange rectangle that represents the agent defined over a class of persons. More is needed into the properties of this class can change the name for a class of people in the *Replication* tab 1000. This means that in our model we shall make 1000 agents. Moreover, on the diagram structure there should be the variables *broj NeZarazenih*, *brojZarazenih* and *brojOporavljenih* of the type Integer.



The model is now almost finished, but we still have to do something to start the work.Recall that on the diagrams for the agents each agent will be placed on creating the uninfected state because the state shows the initial indicator. Since there is no infected person, nothing will happen. Thus there is a need for one agent to manually enter a code that will start the epidemic. It is necessary to open the *Code* field in the *Main Class* section of the *Startup Code* and then enter:

```
ljudi.item(0).statechart.fireEvent("kontakt");
```

which ordered to the zeroth agent to send the signal event *contact*. Animation can be kept simple for the time being. Next, we shall draw a rectangle animation to be used for graphical representation of the infected, uninfected and recovered. Graphical display is enabled by using the *Business Graphics Library*, where it is necessary to insert the object *ChartTime* into the class structure diagram *Main*. The properties of *chartTime* are set on the next picture:



Parameters:

Name	Value
Placeholder	**animation.rectangle**
BackgroundColor	USE Placeholder FILL COLOR
BorderColor	USE Placeholder LINE COLOR
UpdateMode	**AUTO - UPDATE Data EVERY TimeStep**
TimeStep	1
UpdateAtTime0	true
TimeWindow	100
InterpolationType	LINEAR
ScaleType	**AUTO SAME FOR ALL DATA**
LineWidth	1
ShowGrid	**false**
FillUnderneath	false
ShowLegend	**false**
Data0	**brojNeZarazenih**
Color0	
Data1	
Color1	
Data2	
Color2	

Parameter: Data0
Type: real (dynamic)
Value: brojNeZarazenih
Default value: - not specified -

Now, running the simulations, we obtain the following graph:

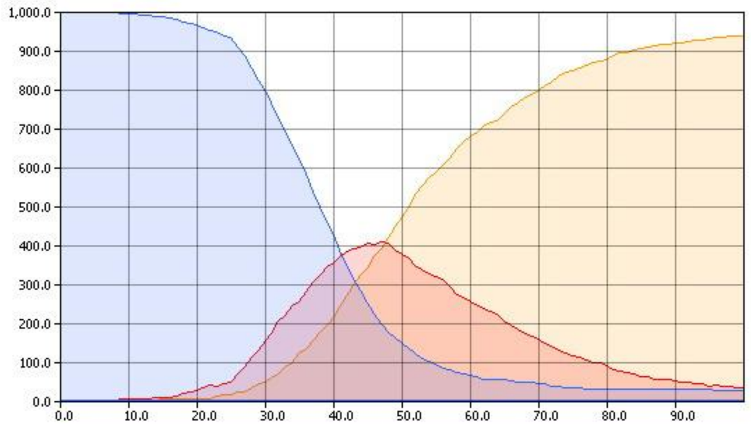


Figure 2.3. System dynamics model

Next, comparing the graphics obtained in the system dynamics model (Figure 2.3) with the same parameters for the Agent based model (Figure 2.4), one can observe the differences.

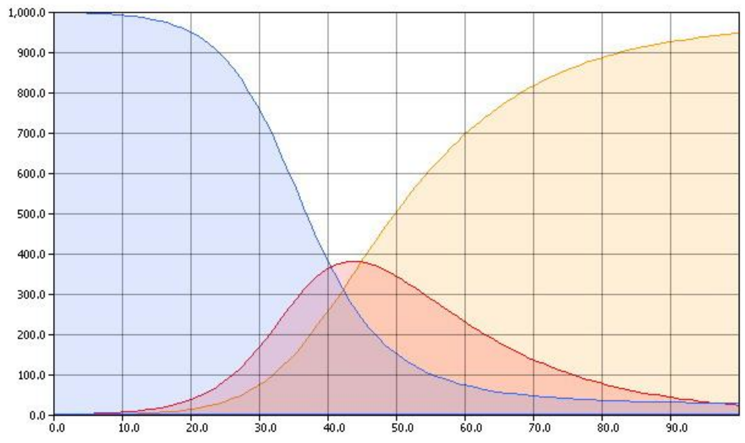


Figure 2.4. The agent based model

From the graph obtained in the agent based model, one can see the stochastic nature of the model, but it is still very similar to results obtained in the system dynamics model. Now, a question naturally arises: Why is it necessary to use agent models and what are its advantages? In the system dynamics model, the duration of the disease was shown by the exponential distribution. It is not a real situation for some diseases. In fact, it would be better if the duration of disease is shown using a triangular distribution.

The transition between the state *infected* and the state *recovered* should be modified so that the rate of transition should be replaced by the *Fire After timeout*, and in the *Timeout field*, the following should be entered:

```
Triangular
        {trajanjeBolesti/2,trajanjeBolesti,trajanjeBolesti*1.5}
```

Now the simulation shows the following graph:

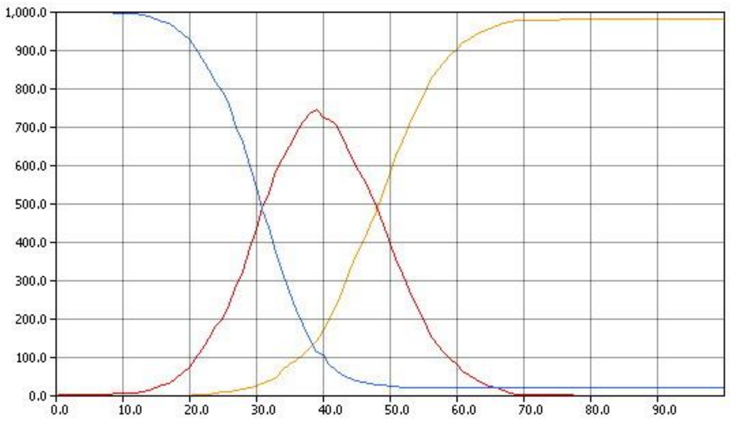


Figure 2.5. Change in a system dynamics model

This change is possible in a system dynamics model. Changing the equation one can try to reproduce this behavior, but it significantly complicates the equation. Even if we achieve to get this behavior in a model of system dynamics, it is just an attempt to reproduce known results, while the agent model is a way to model the actual characteristics of some diseases.
If we recall the system dynamics model previously described, a model is contained in it, and vaccinations. How would it put the process of vaccination in the agentbased model? In view of the possibility of combining paradigms in modeling the *AnyLogic*, we may model the system dynamics through vaccination.

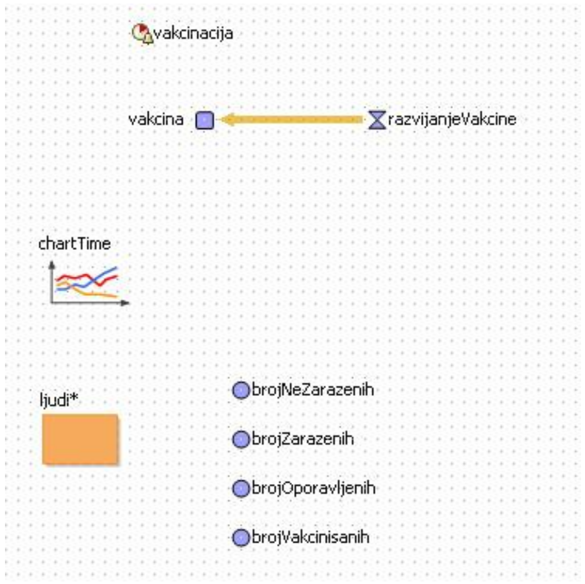
On the *Main* class diagram of the structure will bring the necessary variables for vaccination. One variable should have a name with an initial value of *razvojVakcine* 10 while the other variable is given by the differential equation

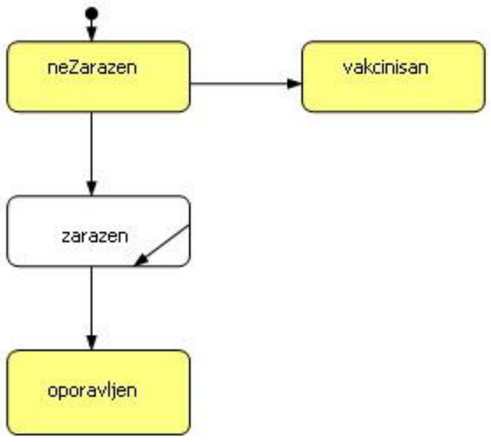
$$\frac{dVakcina}{dt} = razvojVakcine\ .$$

Moreover, the diagram would require a static timer, which should assign the name immunization. The timer will be cyclic, with a timeout and actions to be performed is given by the following code:

```
while (vakcina>1) {
  ljudi.random().statechart.fireEvent("vakcinacija");
  vakcina--;
}
```

This code says that every time the timer comes to its end, perform the vaccinations as long as there is vaccine in stock. Vaccine recipients will be selected in a random way, but only uninfected agents can go into a state of vaccinations.





In the state diagram for the agent, we have to add another state - *vakcinisan*. The starting action for that state will be *main.brojVakcinisanih*++. The transition from the state *neZaražen* into the state *vakcinisan* waits the signal event *„vakcinisan“*. In this way, an agent can move into the state *vakcinisan* only from the state *neZaražen*.

Before starting the animation, it is necessary to put *chartTime* into the *Properties* and show the variable *brojVakcinisanih*. The obtained graph clearly shows the efficiency of the vaccination. Namely, the number of infected

has much decreased, compared with their number obtained in the previous simulation.

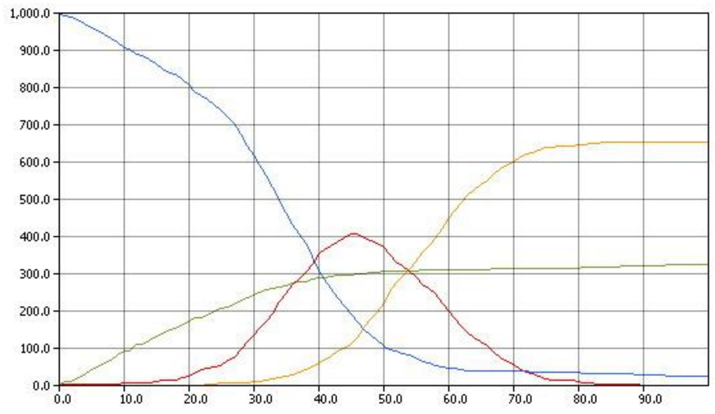


Figure 2.6. The graphs of the *S, I* and *R* functions

Perhaps the most important influence on the spread of the epidemic is the spatial environment in which the epidemic spreads. Introducing the impact of the environment in the system dynamics model implies the work with partial differential equations, and, presumabely such models can take us too far.

However, the introduction of an environmental impact in the agent model is not difficult. Thanks to the agent modeling library, one can easily create an agent with a quite a few different properties. To the diagram of the structure one has to include the object *agentBase* from the library object *AgentBase*. After that, we have to define the

The dimensions of the agent's environment will be 300 x 300. The default network of the agents will be *all in range* and *contact range* should be set to 30. Next, we have to change the code for the internal transition in the state *infected*, as follows:

```
if (randomTrue(verovatnocaZaraze))
agentBase.sendToRandomContact("kontakt");
```

and then add into the property *onReceive* of the object agentBase the following expression:

```
statechart.fireEvent(message)
```

The result of this simulation shows an even greater reduction of the number of infected, because now the environment affects the agents by the rate of contacts. Then the epidemic is spreading slowlier through the population, and thus the vaccination process becomes more efficient.

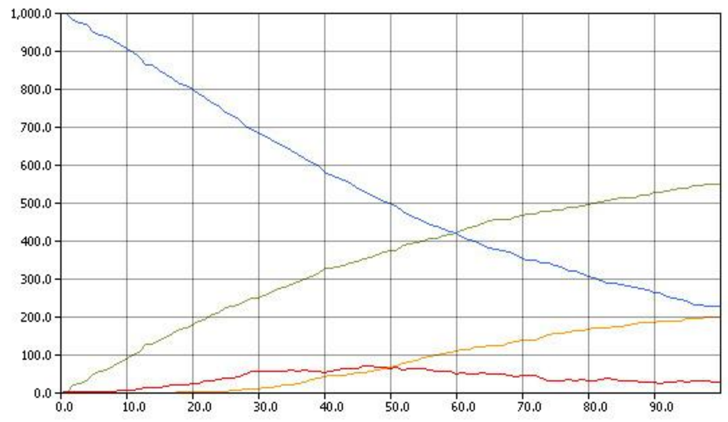


Figure 2.7.

In order to get a better presentation of the environment influence on the spreading of the epidemics, one can add into the model the animation of each agent. Since the agents are multiplied objects, it is enough to make the animation just for one agent, and then in the final animation all of 1000 agents will appear in the model. Next, for better visualization of the epidemic spreading, one should add in the field *Fill color* an appropriate color.

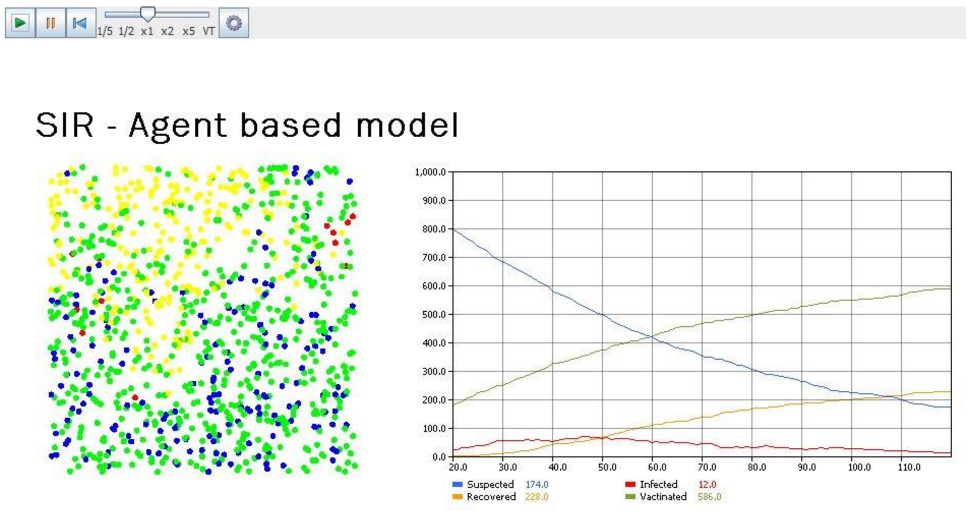


Figure 2.8. The simulation of the model

3. Mathematical pendulum

From a mathematical point of view, the pendulum is an oscillating system consisting of a nonstretchable string of some length but of negligible mass, and a negligibly small suspended ballot with larger mass than that of the string. The problem is to observe the movement of the pendulum under the influence of gravity.

For simplicity, for now we assume that the pendulum's length is l = 1 and that the pendulum's mass is m = 1. The first assumption is that gravity has a constant impact force to the pendulum force *g* acting vertically downward. In order to analyse the movement of the pendulum in a positive or negative direction, we have to measure the angle α compared to the pendulum at rest, and the pendulum angular velocity ω. The equation of the motion of the pendulum is:

$$\frac{d^2\alpha}{dt^2} = \alpha'' = -g\sin\alpha . \qquad (3.1)$$

.

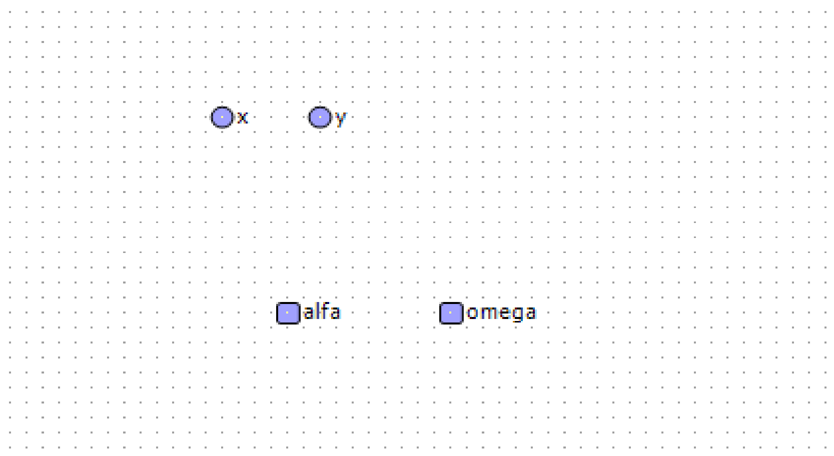
Using the definition of the angular velocity ω, we can put it in the previous term and obtain

$$\begin{aligned} \alpha' &= \omega \\ \omega' &= -g\sin\alpha \end{aligned} \qquad (3.2)$$

By analyzing the behavior of the pendulum, we can conclude that the model of an ideal mathematical pendulum, appears in four states:

- pendulum is at rest vertically down;
- pendulum is moving between two points in which they now rest with the same deflection on both sides of the balanced position;
- pendulum rotates continuously in the same direction and never comes in the steady state;
- pendulum is vertically upright at rest (unstable).

This model of the pendulum is fairly easy to model in the AnyLogic. Namely, it is enough to enter the above equation, and observe the behavior of the pendulum. In the model, we use variables to represent the x and y coordinates of the pendulum in animation, namely the angle α_0 and the angular velocity. To begin with, we introduce two parameters: the initial angle α_0 and the length l of the pendulum and the length *l*



.

By introducing another parameter in *AnyLogic*,one easily adds the resistance at the middle of the pendulum. The parameter *r* represents the resistance of the environment. Just change the equation for the radial velocity:

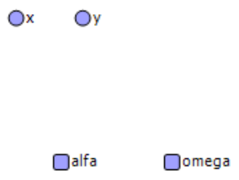
$$\omega' = -g\sin\alpha - r\omega$$

Instead of a detailed description of how the model is made in AnyLogic, we rather use the report generated for each *AnyLogic* model. The report is a good way to get someone to show detailed models of structures.

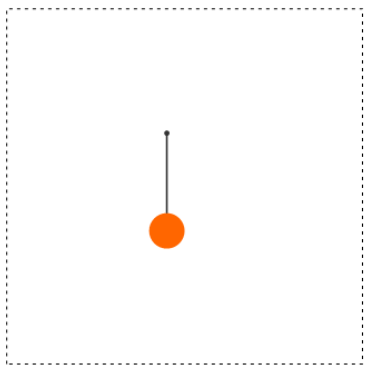
Active Object: Main

Parameters	
Name	alfa0
Type	real
Default value	3.14
Name	l
Type	real
Default value	100
Name	g
Type	real
Default value	9.81

Icon	
Picture	

Structure	
Picture	
Variable	Omega
Variable type	real
Equation type	Integral or Stock
Equation	d(omega)/dt = (-g*sin(alfa))/l
Initial value	0
Variable	alfa
Variable type	real
Equation type	Integral or Stock
Equation	d(alfa)/dt = omega

Initial value	alfa0
Variable	y
Variable type	real
Equation type	Formula
Equation	y = l*cos(alfa)
Variable	x
Variable type	real
Equation type	Formula
Equation	x = l*sin(alfa)

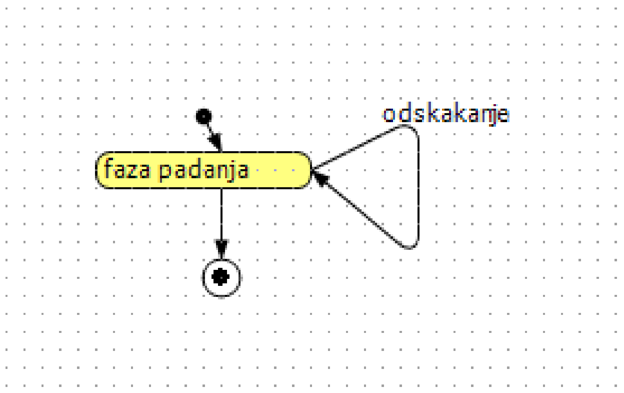
Animation	
Name	animation
Picture	
Oval	Oval1
X	x
Y	y
Line	Line2
End point X	x
End point Y	y

4. The Bouncing Ball

This model is one of the simplest hybrid models, and is thus often used as an example of hybrid modeling. The ball is released from a certain height to a solid base, hitting the ground and moves up and then again falls down. After each shot of the ball on the ground, it loses a certain portion of energy, so that eventually ceases jump. The falling of the ball down is simply described by a system of diferential equations (4.1) given below:

$$\frac{dy}{dt} = V_y$$
$$\frac{dV_y}{dt} = -g, \qquad (4.1)$$

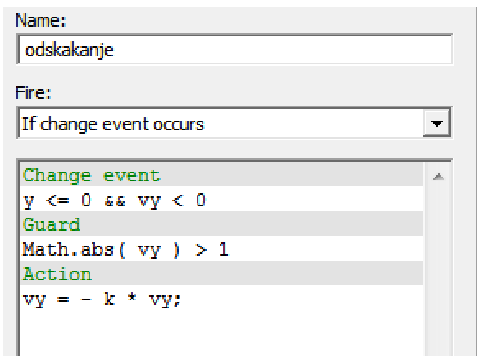
where V_y is the speed of the ball on the y axis, and g is the gravity. The moment when the ball hits the ground is the part when this model becomes hybrid. In *AnyLogic* this is simply modeled using a state diagram.



In the phase of falling, the ball behaves according to the above equations. The transition "bouncing" has to wait for the condition $y \le 1$ and $V_y < 0$, provieded that $|V_y| > 1$. If these conditions are met, then we have

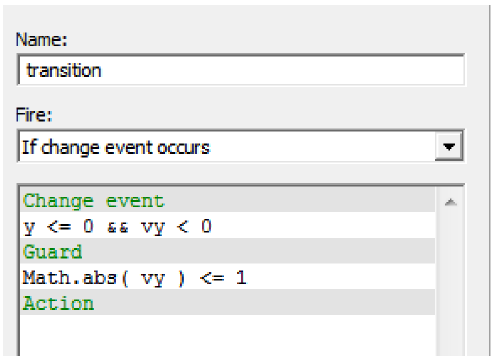
$$V_y' = -kV_y \qquad (4.2)$$

where k in (4.2) is the coefficient representing the loss of energy of the ball. After that, the ball goes back into the "falling" phase.


Name:
odskakanje
Fire:
If change event occurs

```
Change event
y <= 0 && vy < 0
Guard
Math.abs( vy ) > 1
Action
vy = - k * vy;
```

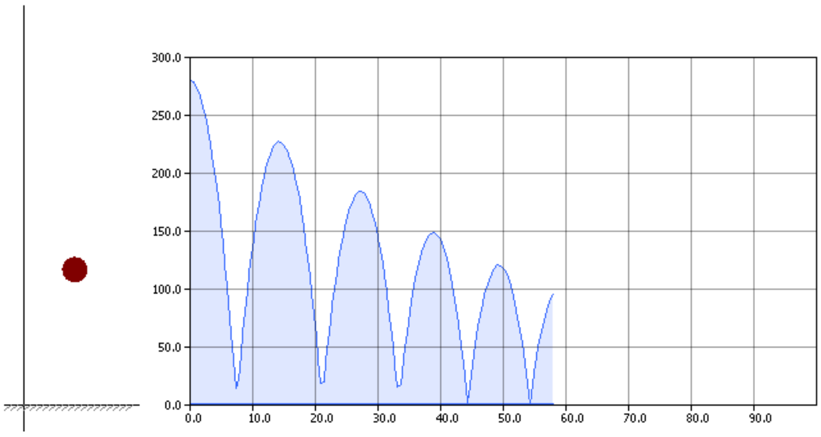
Next, the state diagram enters its final state, which means that the ball does not have enough energy to continue its bouncing, and the system goes to a stand..



Name:
transition
Fire:
If change event occurs

```
Change event
y <= 0 && vy < 0
Guard
Math.abs( vy ) <= 1
Action
```

This example shows how a seemingly simple model can have a hybrid behavior, thus it is essential to have a tool that allows you to easily create hybrid models.



5. The Lotka-Volterra model

The Lotka Volterra model is composed of a pair of differential equations describing the relationship between the predators and preys in the simplest case. The model was developed around 1925, independently by two authors, a biologist Alfred Lotka, and the famous Italian mathematician Vito Volterra. Volterra has developed a model to explain his son in law the behavior of two fish populations in the Adriatic Sea, where one species (the preys) is the food for the other one (the predators, but often in the literature called "sharks").

The previous model is a natural extension of the so-called logistic model, introduced around 1845 by the Belgian scientist Verhulst. In the model, the population of predators and preys oscillates so that the largest population of predators is just below the largest prey population. This model has the following assumptions:

– population of preys will grow exponentially if there are no predators;
– population of hunters will starve (disappear) in the absence of the preys;
– hunter can eat as many preys;
– two sides are moving in a homogeneous, confined space in a random way.

Let us introduce the following variables:

- P: predator population
- N: prey population
- t: time
- r: rate of growth of the prey population
- a: rate of attacks on the prey population
- q: rate of mortality of the predator population
- c: efficiency of conversion of food into descendants

In some models, another assumption is introduced, namely the rate of mortality of the prey population, In general, it is assumed in the model that the prey does not become extinct naturally, but rather gets eaten by the predators. We add that in some models the product of the parameters c and a is called the productivity of predators.

Let us consider what happens to the predator population when the prey population is not present in the observed eco-system. Without predators' food, the number of predators declines exponentially, since then it satisfies the following simple ODE:

$$\frac{dP}{dt} = -qP \tag{5.1}$$

Note that qP is the product of the mortality rate of the predators and the population of predators. The negative sign in the upper equation indicates the decline of the number of predators. We get a more complicated and perhaps a more realistic model in the form od ODE, if we add the product $caPN$ to the right hand side of the upper equation (5.1), see equation (5.2) below. Note that the mentioned product describes the rate of attacks on the preys multiplied by the number of predators and the number of preys.

$$\frac{dP}{dt} = caPN - qP\ . \tag{5.2}$$

Next, the population of preys is expected to grow exponentially in the absence of predators:

$$\frac{dN}{dt} = rN \tag{5.3}$$

The presence of predators prevents the exponential increase of the preys. The product aPN describes the mortality of the preys, implying the following ODE:

$$\frac{dN}{dt} = rN - aPN\ . \tag{5.4}$$

The obtained equations (5.1) – (5.4) describe the predator-prey model, which predicts cyclic dependencies. As the number of predators P grows, so does the product aPN also, which additionally enhances the growth of the number of predators. At the same time, the aPN product reduces the number of preys, N, which indirectly reduces the number P of predators. Now, as the value of aPN in (5.4) decreases, the prey population is recovering and *N* starts to grow. But then the parameter *q* increases and the cycle will start again. The ideal curve is shown on the graph below.

The model was tested also in experimental conditions, by following the number of both populations. In particular, one experiment, done by Huffaker in 1958 can be accepted as valid.

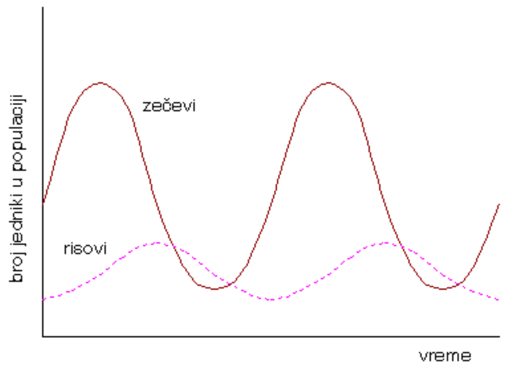


Figure 5.1. Graph of the Lotka-Volterra model

He observed two populations. One was the predator, and the other the prey population. In the observed envionment he put several oranges (prey's food), and covered them ba wax, which enabled Huffaker to control the amount of the eaten food. In this environment he put also few rubber balls. On the graph 5.2 below we can see the result of Huffaker's experiment, obtained for one choice of oranges and balls.

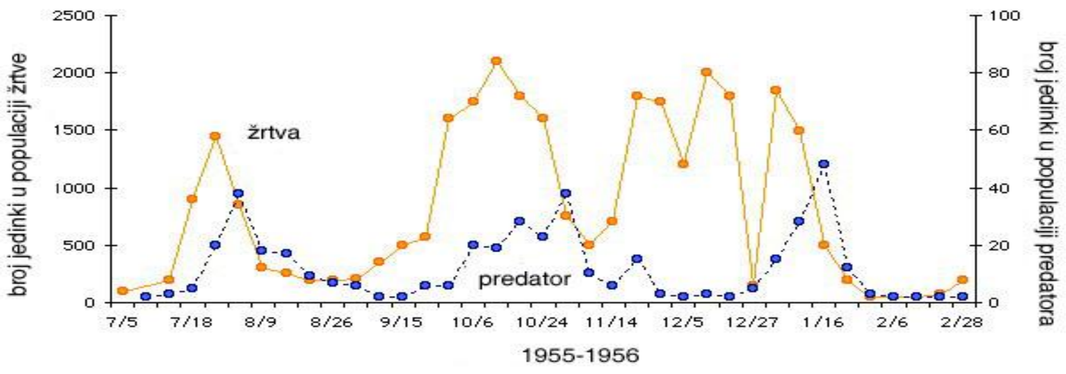


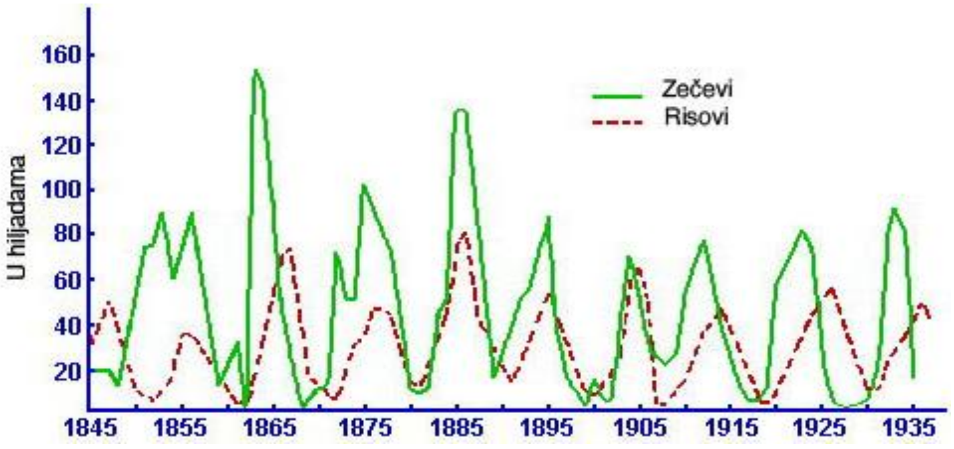
Figure 5.2. Graph of the Lotka-Volterra model obtained by following two populations in laboratory conditions

The graph shows that the populations are exhibiting a cyclic behavior, and that the maximum value for predators behind the maximum value for the preys. In order to conclude that this experiment really reflects the assumptions of the Lotka-Volterra model, we must consider another fact. Actually, in this experiment there is a rather complex environment in which there are also other populations, which is contrary to the assumption made in the model. Thus also the assumption that the predators meet the preys only randomly is questionable.

Another experiment has been based on monitoring the number of animals in nature. The problem in monitoring the population in nature and modeling of the population is that most predators rely on multiple preys. Fortunately, there are several species that are strictly specialized in the course of evolution, and the food they eat is of just one kind. In Canadian forests there is an ideal example of this relationship between the population of lynx and snow rabbits (hares).

The Hudson Bay Company has been closely monitoring the number of lynx and rabbits between 1800 and 1900. It was for a long time noticed that these two populations are very good for observation, because of the frequent attacks of the lynx on the hares, in view of their way of hunting.

The collected data show cyclic oscillations in both populations at approximately every 12 years. The graph 5.3 presents the data obtained.

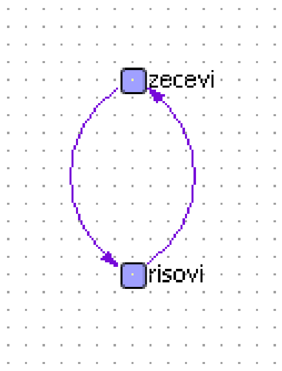


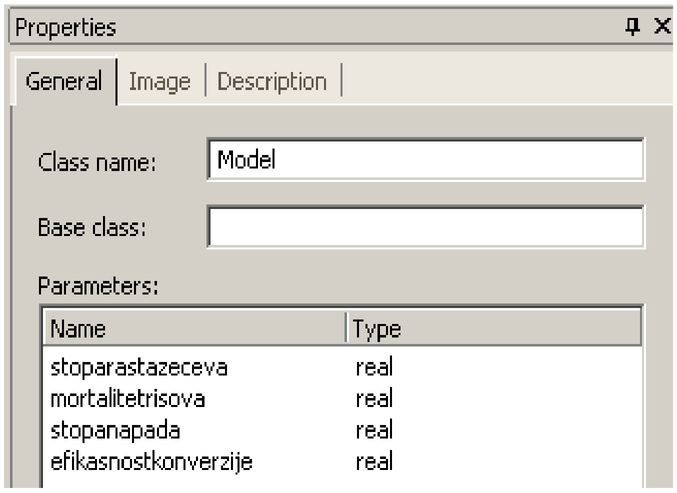
5.3 The graph obtained by observing the populations of lynx and hares in the nature.

A lack of Lotka-Volterra model is relying on unrealistic assumptions. The prey population has a limited amount of food in nature and this affects the number of predators, while, on the other hand, they can eat unlimited amounts of food. From the mathematical point of view, the cyclic behavior we got from a system of differential equations in the Lotka-Volterra model tends to repeat itself endlessly and thus a similar behavior will be obtained for each set of values for the four-parameter model.

Let us conclude that the Lotka-Volterra model is not enough to present the behavior of most populations in the nature. In fact, additional information, specific for the analysed system, has to be inserted in the system.

The presentation of models in *AnyLogic* we start with the simplest model. Since the *AnyLogic* has a good method of solving differential equations, the easiest way is to define two varuables that will represent the populations of hares and lynx. These two variables appear in a system of differential equations for the Lotka-Volterra model given below in this chapter. The variables are placed in the class of an active object in the model. At the class level the following parameters are defined *natalitetrsova*, *stoparastazeceva*, *stopanapada*, *efikasnostkonverzije* with initial values. The animation shows the number of hares and lynx, both graphically and in numbers.





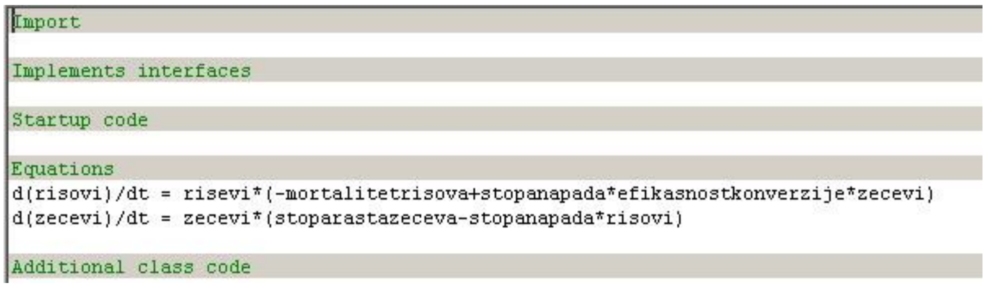
Class name: Model

Base class:

Parameters:

Name	Type
stoparastazeceva	real
mortalitetrisova	real
stopanapada	real
efikasnostkonverzije	real

5.4. The AnyLogic model of the interaction between hares and lynx



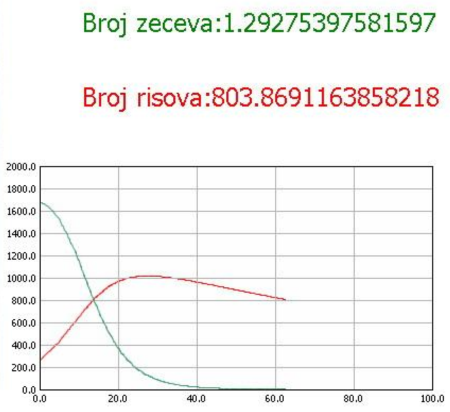
```
Import

Implements interfaces

Startup code

Equations
d(risovi)/dt = risevi*(-mortalitetrisova+stopanapada*efikasnostkonverzije*zecevi)
d(zecevi)/dt = zecevi*(stoparastazeceva-stopanapada*risovi)

Additional class code
```



Broj zeceva:1.29275397581597

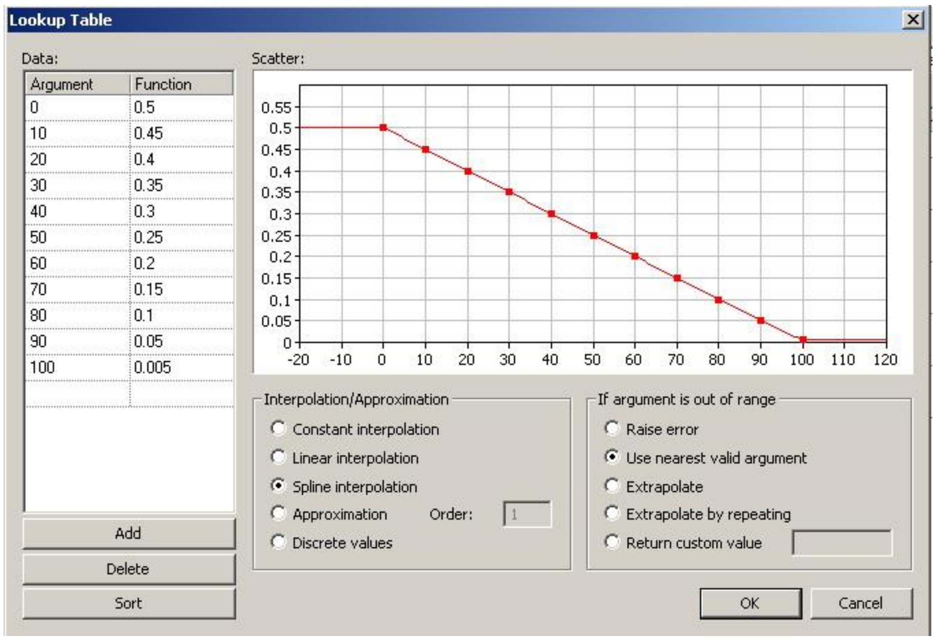
Broj risova:803.8691163858218

Figure 5.5. A linx and its prey interaction in number and the corresponding graph

The following mathematical model represents a shift in terms to some finer structure of the model and better use of opportunities of the *AnyLogic* program, in which we have a classic example of system dynamics. Differential equations appearing in (5.5) are given in a somewhat different way, but they are still in line with the Volterra-Lotka model.

$$
\begin{aligned}
&\frac{d(Zecevi)}{dt} = RodjeniZecevi - UginuliZecevi \\
&\frac{d(Risevi)}{dt} = RodjeniRisevi - UginuliRisevi \\
&RodjeniZecevi = Zecevi \cdot NatalitetZeceva \\
&UginuliZecevi = GustinaZeceva \cdot Risevi \\
&RodjeniRisevi = Risevi \cdot NatalitetRiseva \\
&gustinazeceva = \frac{Zecevi}{PovrsinaOblasti}
\end{aligned}
\tag{5.5}
$$

Lynx mortality was given as a table search on the basis of these data was made using the spline interpolation function.


Lookup Table

Data:

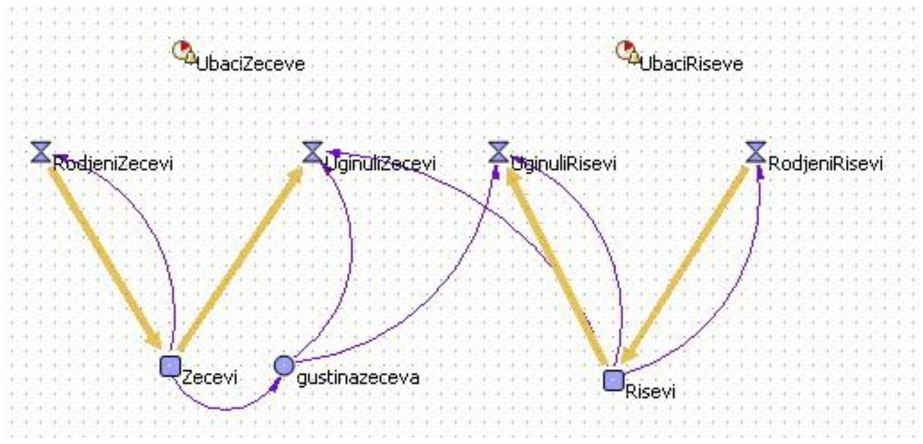
Argument	Function
0	0.5
10	0.45
20	0.4
30	0.35
40	0.3
50	0.25
60	0.2
70	0.15
80	0.1
90	0.05
100	0.005

Interpolation/Approximation: Constant interpolation, Linear interpolation, [✓ Spline interpolation], Approximation (Order: 1), Discrete values

If argument is out of range: Raise error, [✓ Use nearest valid argument], Extrapolate, Extrapolate by repeating, Return custom value

The structure is inserted, and two timers that are used for insertion of lynx and hares into the system during the execution model. For each timer is set to operate in manual mode. In executing a timer *UbaciZeceve* execute the Java command Rabbits + = 1000 Timer *UbaciRiseve* executed Lynx + = 50.

In the active object class, there are three real parameters, namely *povrsina oblasti* (area of the region), *NatalitetZeceva* (natality of the hares) and *NatalitetRiseva* (natality of the lynx) with default values given below.



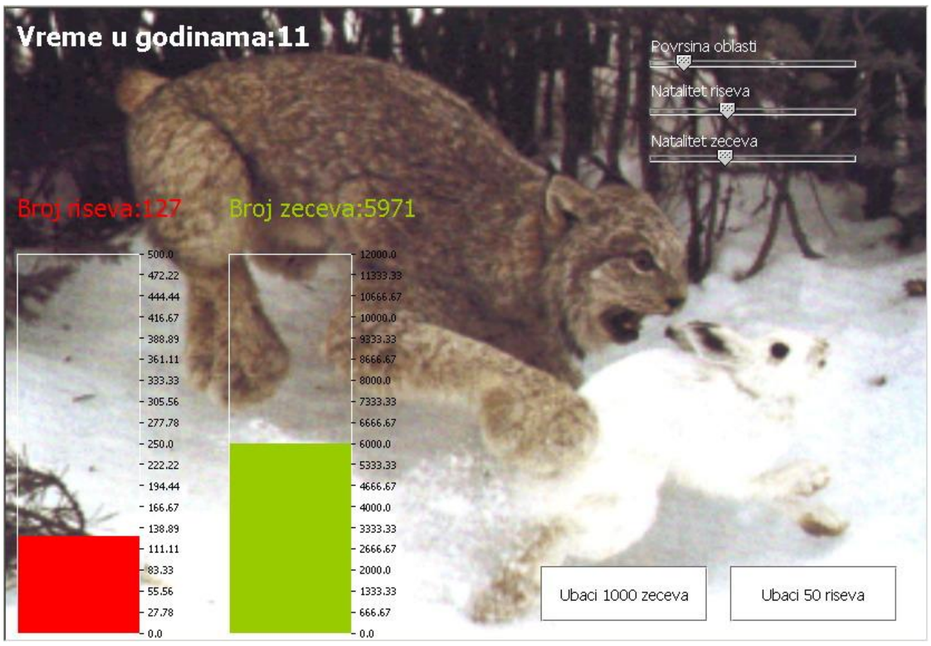
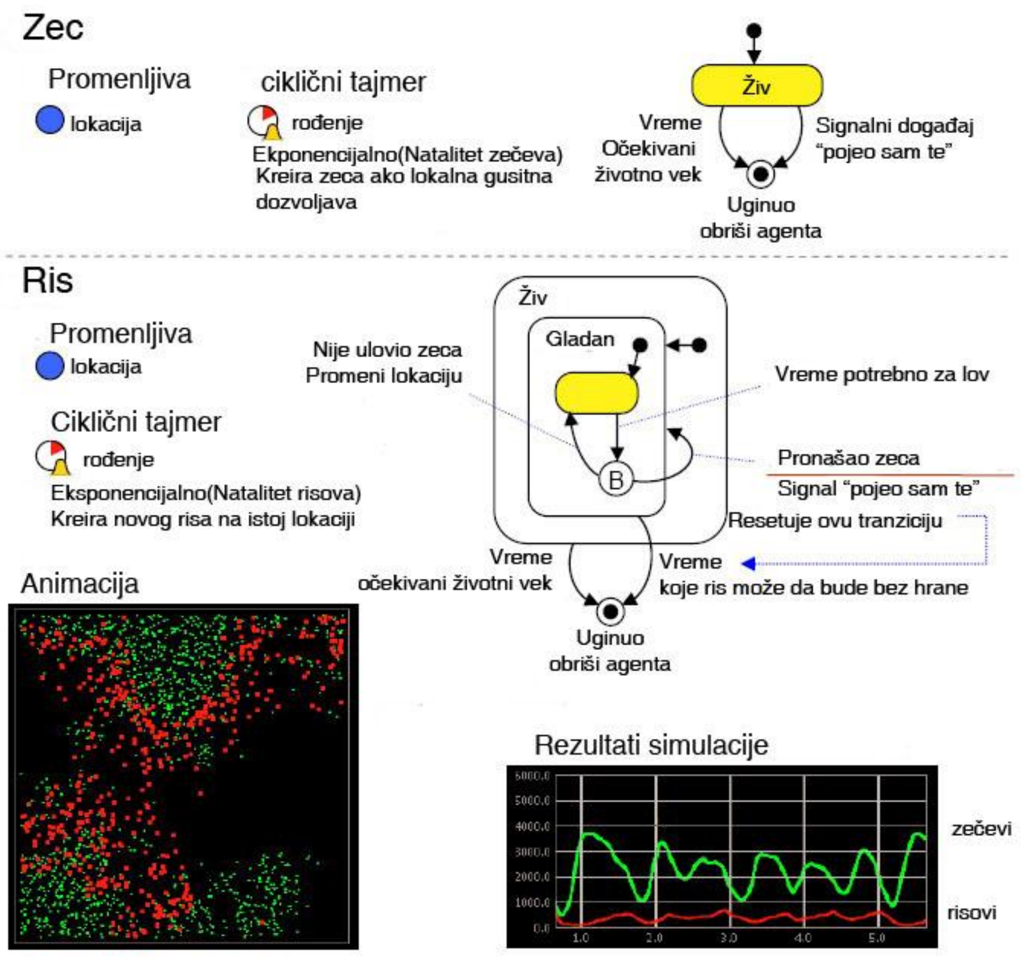


Figure 5.6. Another linx and prey interaction in numbers

The animation is in this model more complex. Moreover, in order to display the number of individuals for the hare and lynx numbers, and by using diagrams, interactive forms have been inserted. Two command buttons are connected to timers in the diagram structures that increase the number of individuals. Three sliders are linked to parameters *povrsinaoblasti*, *NatalitetZeceva* and *NatlitetRisova*. Using the slider in the conduct, we can change the values for these parameters.



5.7. Predator prey model created by agents

The model based agents in the *AnyLogic* program represents rather well the real system. Now, we add the following hypotheses given below.

- Both the hares and lynxes have a certain life expectancy. Their deaths are caused by age, starvation and lynx attacks.
- Hares and lynxes are aware of the two-dimensional space in which they are located.
- The density of the hare population is limited, and they reproduce themselves only if there is enough space around them.
- Lynxes hunt only in the area around them and they do that with certain frequency.
- If the lynx does not catch a hare in the course of hunting, then it changes its position in space.
- If in a certain time period a lynx fails to eat a hare, then it dies.

Next we give some elements of the *AnyLogic* model.

Each agent is assigned the variable *Location*, which contains information on the current position of the agent in space. The initial position is assigned of each agent in a random way. The value of the variable location site is updated when the agent moves in space and that influences its behavior.

The timers in this model are defined both for the birth and death of the lynxes and the hares.The behavior of the timer is cyclic, and creates a new agent by the exponential distribution. Births of the hares depend also on their local density. For the hares, the diagram is simple, and it describes the behavior of the hares. The states in the diagram are *alive* and *dead*.

They are defined by two transition: one for the signal (message) of the lynx "I ate you!". The second transition is the definition of time and a life expectancy of a hare. The state diagram for the lynxes is more complicated. Namely, the lynxes prefer a certain time period for hunt. Whether a lynx finds a hare, depends on the density of hares and the number of lynxes in the area. If a lynx finds and eats a hare, then it sends a signal (message) to the hare "I ate eat you!", leaves the state of hunger, and immediately reenters to the hungry state. If the lynxes had no luck in hunting the hares, and the lynxes do not catch a hare, then they move in space, but remain in a state of hunger.

When a lynx catches a hare, he immediately resets the time that indicates how many lynxes may be without food before they die. The time that the lynxes can spend without enough food is just one day, otherwise he does not survive. The model defines algorithmic functions to determine whether the area around the hares is overcrowded, while this function determines new areas for lynxes in a random way.

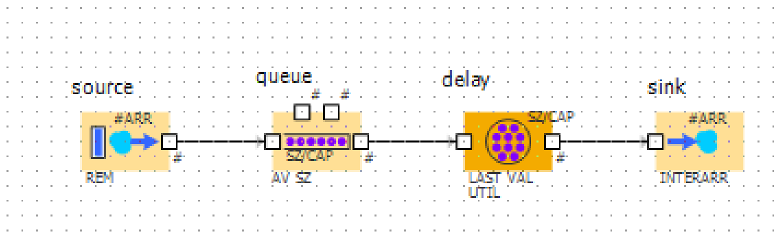
The result obtained in the simulation gives a graph that looks much like the graphs obtained by observing a population in nature. The highest values for lynxes are a little behind the greatest value for hares.

Unlike to the previous models, it can happen that the total population tends to extinction, because of the parameter values. The simulation was added a 2D image of the space where the lynxes and hares are staying.

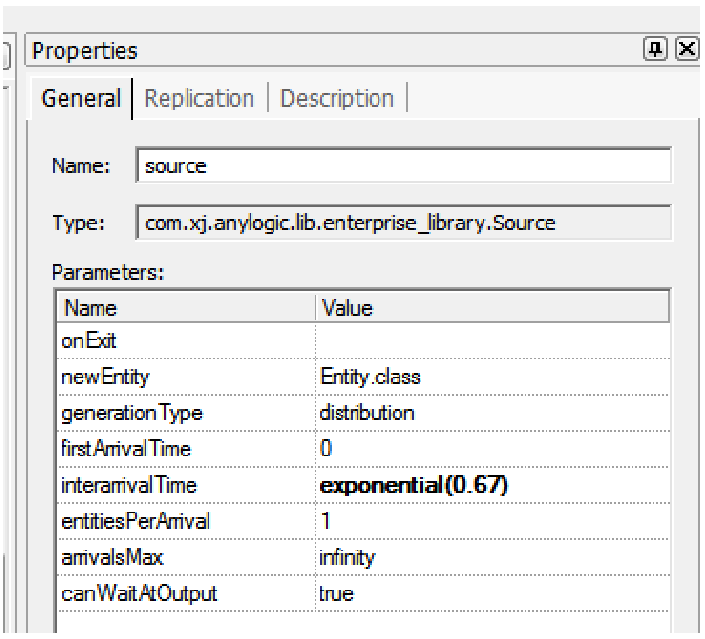
The model is based on an *XJTek* company's model.

6. A universal, simple queuing model

The model to be described can be applied to various types of queuing models. Often we meet the situation when we have to wait in a line to "get" a service, say, visit a doctor, cash a check, or wait in a line in front of an ATM. All these models have in common that the client enters the system, gets in the line, waits for the service, receives the service he needed and then leaves the system. This is a rather simple model, but the main problem in making this model is finding the appropriate random Using the Enterprise Libary, an important part of the *AnyLogic* programme, one can put together a model with individual elements. By adjusting the parameters of each element that are used, one gets the desired behavior of the model. For now on, we only use four elements to make a simple queuing model.



The first element, *source*, is used for the generation of the entities (client, manufacturing parts, etc.,) arriving into the system. The parameters for the *source* are simple, and, most often, they are used for entering the random distributions according to which the entities were generated.



Name: source

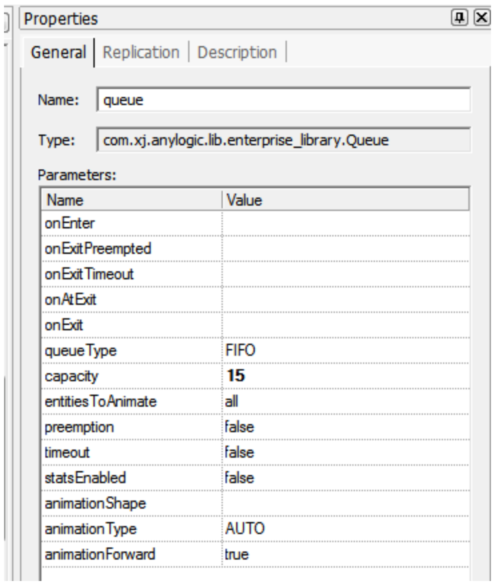
Type: com.xj.anylogic.lib.enterprise_library.Source

Parameters:

Name	Value
onExit	
newEntity	Entity.class
generationType	distribution
firstArrivalTime	0
interarrivalTime	**exponential(0.67)**
entitiesPerArrival	1
arrivalsMax	infinity
canWaitAtOutput	true

In our example, the arrival of entities is accomplished by an exponential distribution with parameter 0.67, and by each generation of new entities in the system exactly one entity is inserted, because in the field *entitiesPerArriva*, when one enters *1.arrivalMax*, defines the maximum number of generated entities, and by entering *infinity*, we get an infinite number of generated entities. More precisely, the entities will be inserted into the system throughout the duration of the simulation model.

The next element in the model is the *queue*, which enables the generated entities to stand in a line. This element is essential for defining the maximum capacity line, which in our case is 1.5.



Name: queue

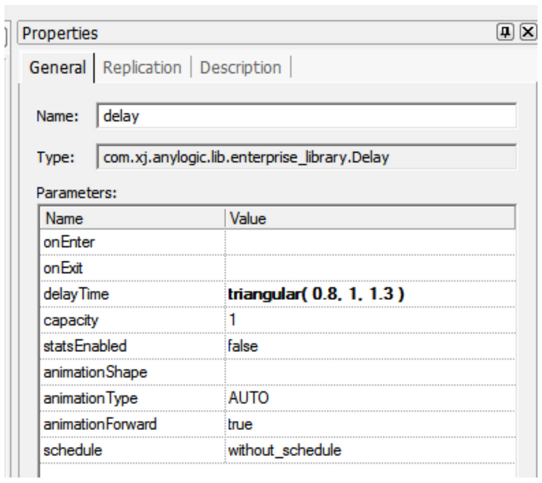
Type: com.xj.anylogic.lib.enterprise_library.Queue

Parameters:

Name	Value
onEnter	
onExitPreempted	
onExitTimeout	
onAtExit	
onExit	
queueType	FIFO
capacity	**15**
entitiesToAnimate	all
preemption	false
timeout	false
statsEnabled	false
animationShape	
animationType	AUTO
animationForward	true

One can also select the type of the queue. In our case, the queue is of FIFO type (first in first out). The fields *preemption* and *timeout* are also very important, since they allow us to model a simple typical customer behavior in such systems. It might happen that some clients will not want to enter and/or wait in a long queue, but will rather leave the system. If the field *preemption* enrolls the *true* value, this option will be activated in the model. What will happen next with this entity can be defined so that the output from the queue element corresponding to the option *preemption*, connects a new element from the *Enterprise Library*, e.g., with the *sink* entity.

By the same principle it can also be used to represent timeout options of those customers who stood in the line, but their waiting times turned out to be too long, and thus they wanted to give up. The d*elay* element is a "service delivery", or, for example, the time the client spent waiting for his cashier to take or deposit some cash.



Name: delay

Type: com.xj.anylogic.lib.enterprise_library.Delay

Parameters:

Name	Value
onEnter	
onExit	
delayTime	**triangular(0.8, 1, 1.3)**
capacity	1
statsEnabled	false
animationShape	
animationType	AUTO
animationForward	true
schedule	without_schedule

In the *delay time* one has to place a random distribution, which corresponds to the duration of the service (in our case it is a triangular distribution with parameters 0.8, 1, 1.3, which means that the customer is retained, for example on the bank desk, at least

0.8 time units, a maximum of 1.3 with an average time of a time unit). Field capacity determines how many clients can be serviced simultaneously.

The element sink has nothing to adjust for the behavior of the model, since it simply destroys the generated entities.

This model is a starting point for learning and understanding the queuing models. By extending this model with other elements of the Enterprise Library, we can get very complex models such as models of airports, emergency services, manufacturing, etc.

It is noteworthy that, in these models, the behavior of the models associated with the elements in the model and not the entities situated in the model as is the case with agent models where the behavior is defined for the agent (the entity). It is also important to note that this model does not include the use of resources, which also represents a step towards a "more realistic" model, for example if you model a doctor's surgery with a doctor, a nurse and say an ECG machine.

These are the resources that the patient will need to use in the course of receiving the service. Sharing of resources is actually the biggest problem to be addressed by the mathematical model. Let us add that a natural question is whether the doctor's office will pay off purchasing an ECG device in order to reduce the waiting time for patients and enable more medical examinations.

The correct answer on this important question (which means more revenue and more satisfied patients) can be easily checked by the queuing model.

REFERENCES

[1] J. Caldwell and J. Douglas K. S. Ng, Mathematical Modeling, Kluwer Academic Publishers, 2004.

[2] H. Caswell, *Matrix Population Models: construction, analysis, and interpretation,*

[3] Sinauer Associates, Inc. Publishers, 2001.

[4] V. Čerić, *Simulacijsko modeliranje*, Školska knjiga, Zagreb, 1993.

[5] N. D. Fawkes, J. J. Mahony, *An Introduction to Mathematical Modelling*, John Willey and Sons, Second Edition, 1996.

[6] R. Haberman, *Mathematical Models*, Fourth Edition, Prentice-Hall 1977. S. Ross, *Simulation*, Third Edition, Academic Press, 2002.

[7] M. Haddin, *Modelling and Quantitative Methods in Fisheries*, Chapman & Hall/CRC, 2001.

[8] J. A. Hamilton, Jr., D. A. Nash, U. W. Pooch. *Distributed Simulation*, CRC Press, 1997.

[9] Mark M. Meerschaert, *Mathematical Modeling*, Second Edition, Academic Press, 1998.

[10] D. Mooney, R. Swift, *A Course in Mathematical Modeling*, Mathematoical Association of America, 1999.

[11] A. Takači (with L. Juhas and D. Mijatović), *Skripta za matematičko modeliranje* (in Serbian), WUS, Belgrade, and Faculty of Sciences, Novi Sad, 2006.

[12] B. P. Zeigler, H. Praehofer, T. G. Kim, *Theory of Modelling and Simulation*, Second Edition, Academic Press, 2000.

[13] *AnyLogic* User Manual, XJ Technologies, 2005.