

**Multi-Agent Systems (MAS)**

**Discrete-Event Simulation**

Tatiana Makhalova

Clermont-Ferrand, France 2015 - 2016

**Contents**

1. [Model description 3](#_TOC_250019)
   1. [Concept 3](#_TOC_250018)
   2. [Inhabitants 3](#_TOC_250017)
   3. [Viruses 4](#_TOC_250016)
   4. [An Expert 4](#_TOC_250015)
   5. [Objects: summary 4](#_TOC_250014)
2. [Implementation 5](#_TOC_250013)
   1. [Software Engineering Tools 5](#_TOC_250012)
      1. Profiling tools 5
      2. [UML tools 6](#_TOC_250011)
   2. [Implementation of classes of agents 6](#_TOC_250010)
      1. [Class Virus 7](#_TOC_250009)
      2. [Class Inhabitant 7](#_TOC_250008)
   3. [GUI 8](#_TOC_250007)
   4. [Statistics 10](#_TOC_250006)
3. [Analysis of results 11](#_TOC_250005)

[3.0.1 Ways of the model evolution 11](#_TOC_250004)

1. [Conclusion 13](#_TOC_250003)
2. [Python code (fragments) 14](#_TOC_250002)
3. Fragments of code 15
   1. [Class ”Virus” 15](#_TOC_250001)
   2. [Class ”Inhabitant” 20](#_TOC_250000)

**Introduction**

A cellular automata is a powerful tool for different kind of simulations. In this project we consider the SIS-like (susceptible-infectious-susceptible) model of epidemics, some details of the implementation and the results of simulations with respect to various parameters, such as the probability of infection and recovery, the density of inhabitants and viruses populations, lifetime of viruses and strategies of inhabitants movements.

In the first section we consider the concept of the model, types of agents, their properties and parameters. Details of the implementation are represented in the second section. In the third section we discuss results of model evolution and the simulations result depending on different parameters.

# Model description

## Concept

The simple SIS-like model of epidemics is described in the project. There are two type of objects (agents) here: *viruses* and *inhabitants*. Inhabitants move inside an rectangle area situated at a 2-dimensional torus. Each inhabitant moves randomly or by its own intentions within an area. If an inhabitant fall into a surrounding of a virus it begins to ache sick with a probability *P* (*inf ection*) and will recover in the feature with probability *P* (*recovery*). The probability of infection is defined by a virus (each virus can have its own value or common values for all instances of the *Virus* class), while the probability of recovery is be defined for inhabitant (the value may be different or the same for all instances of the *Inhabitant* class). Viruses can arise and vanish, inhabitants can only move and change their states.

It should be noted that proposed model is not the classical SIS model, since the network changes dynam- ically (due to moving of inhabitants) and the spreading of an epidemic goes from a virus to an inhabitant (whereas in the classical model it spreads from an infected inhabitant to susceptible ones). Thus, it is impossible to get an analytical solution and the equations, derived by Kermack and McKendrick, are not applicable in this case. We can get some statistic and investigate the properties of the model by simulations. Further, we will consider each class of objects, their properties (parameters) and actions.

## Inhabitants

Inhabitants have its own or the same for all inhabitants probability of recovery (*recoveryRate* field). The number of inhabitants *N* is constant, they cannot die, but they can change their states (susceptible *S* or infected *I* - the property *Infected* ), at each time *t* we have *St* +*It* = *N* . Inhabitants go from being susceptible to being infected and to being susceptible again (fig. 1).

We defined two types of movement: random and knowledge-based. The last type of movements takes the best cell within a given neighborhood (determined by *radius*). The best position corresponds to the cell with the smallest density of inhabitants and viruses (weights for density of inhabitants and viruses are used to manage the importance of parameters and therefore to change the strategy of movements). An inhabitant will retake the cell on an iteration, if the more healthy inhabitant wish to take the same cell (in other words, we define inhabitants priority for resolving conflicts during the relocation procedure).

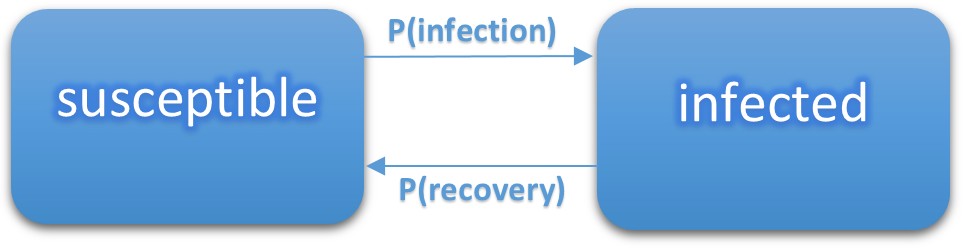


Figure 1: Inhabitants states

## Viruses

**Viruses** arise by chance in a random position and have been living for a certain period of time (*lifetime* is the number of iterations during which an virus is presenting and spreading an infection). Each virus has its own radius (*radius*) of spreading, infection probability (*infectionRate* is the probability that an inhabitant becomes infected after relocation into an area of influence of a virus), on each iteration the random number of viruses arises (the maximal rate of new viruses is a parameter of simulations *virusesRate*). Viruses can spread an infection in the given neighborhood (fig. 2) with a given probability (the same for all objects or random for each virus).

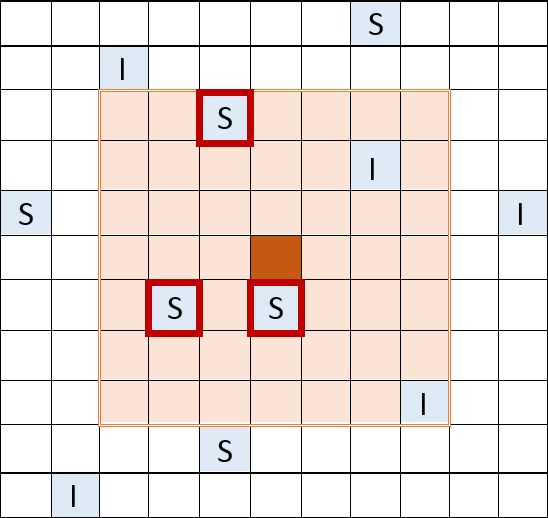


Figure 2: The spreading of an infection: a virus infects susceptible inhabitants in the neighborhood (cells are circled in red) with a certain probabilrty

## An Expert

An expert was introduced for collecting statistic, generating of random numbers and giving some information (for example, a density of objects in a cell). Creating of viruses and inhabitants, supplying information to a view (GUI), collecting of statistic (including data that is given by deleted viruses) fall to an expert.

## Objects: summary

Table 1: The description of viruses and inhabitants

|  |  |
| --- | --- |
| **Viruses (objects)** | **Inhabitants (cognitive agents)** |
| Become activated on each iteration  (activity - spreading disease)    Don’t organize: appear in a random cell Don’t communicate with each other | Analyze the environment (how much viruses and  inhabitants live in the given area)    Select a cell with the most favorable conditions Social intelligence: enter into negotiations in the case of a conflict,  an inhabitant may give place to more healthy one    Don’t map out long-term strategy, since the en- vironment changes on each iteration |

# Implementation

## Software Engineering Tools

Visual Studio 2015 is used as a programming environment, a programming language is C#. To make the analysis of simulations result the following libraries have been used: *pandas*, *numpy* and *matplotlib* (Python).

* + 1. **Proftling tools**

For the performance analysis Microsoft Visual Studio Profiling Tools was choosen. The table 2 represents available tools for scrutiny of the program. Next, we briefly look at some of them.

Table 2: The profiling methods are available in *VS 2015*

|  |  |
| --- | --- |
| **The method** | **Description** |
| CPU Sampling    Instrumentation  .NET memory allocation Concurrency | Collects a call stack of the functions that are ex- ecuting on the processor, memory allocation in- formation is also available in this mode    Collects detailed timing information about each function call including memory allocation    Tracks a lifetime of objects (starts on resource allocation, ends after garbage collecting)    It is used for multi-threading applications |

**CPU sampling** (fig.3) provides information on functions which do the most of work. To analyze the

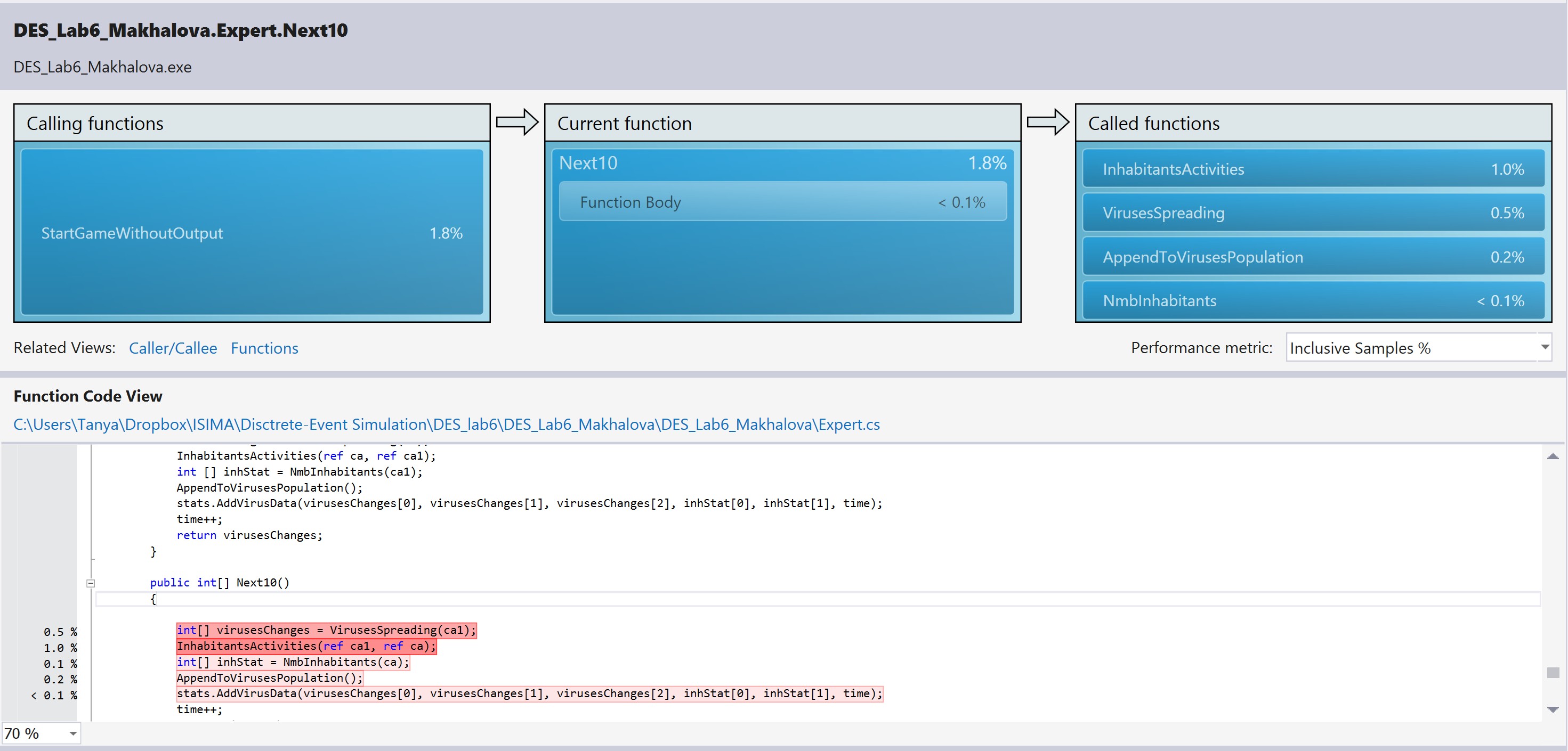


Figure 3: A performance report by using sampling (an analysis of exclusive sampling)

weakness of ”lite” functions the inclusive analysis was performed. The comparative analysis the execution time of the target function and child function was made.

**Instrumentation** (fig.4) provides tools for understanding the impact of input and output operations on application performance, since it gathers detailed timing information about a section of analyzed code. Inclusive sampling provides statistics of the target function and other functions that are called by the target function. Is is used to analyze and optimize such function as spreading of viruses, moving inhabitants and the ”heaviest” sub-function of the target one.

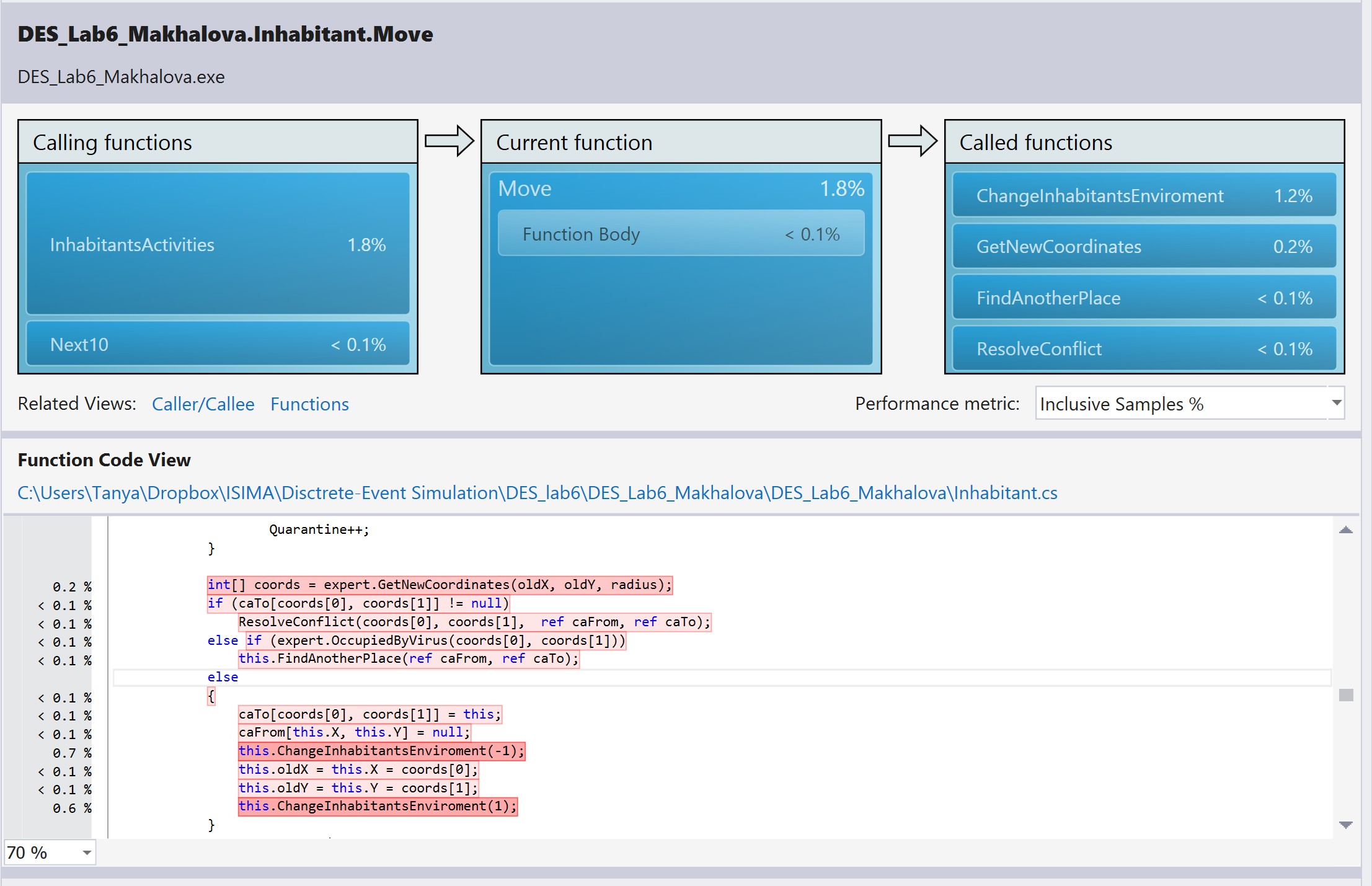


Figure 4: Performance report by using instrumentation

### UML tools

The UML diagrams were constructed with Visual Studio 2015 (manual mode). Standard tools of VS allow to create several types of diagrams (fig.5), a diagram of the process of cognitive inhabitant movements will be represented further (fig.9).

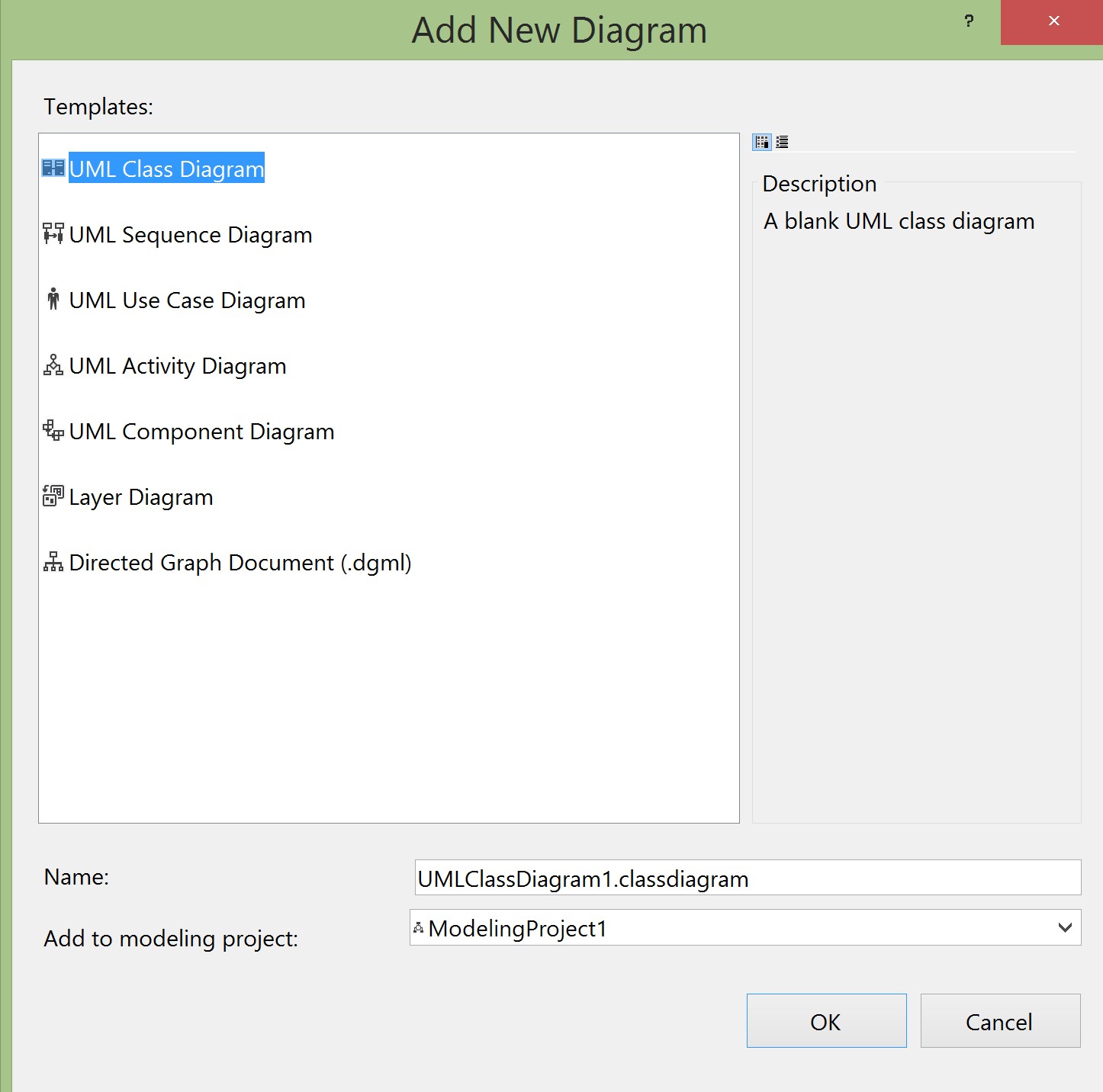


Figure 5: The type of diagrams available to use

## Implementation of classes of agents

The structure of the class diagram is represented on fig.6. Viruses and inhabitants inherit from the same class *MASObject* that contains the following properties: *coordinates of object*, *radius of spreading*, *id*.

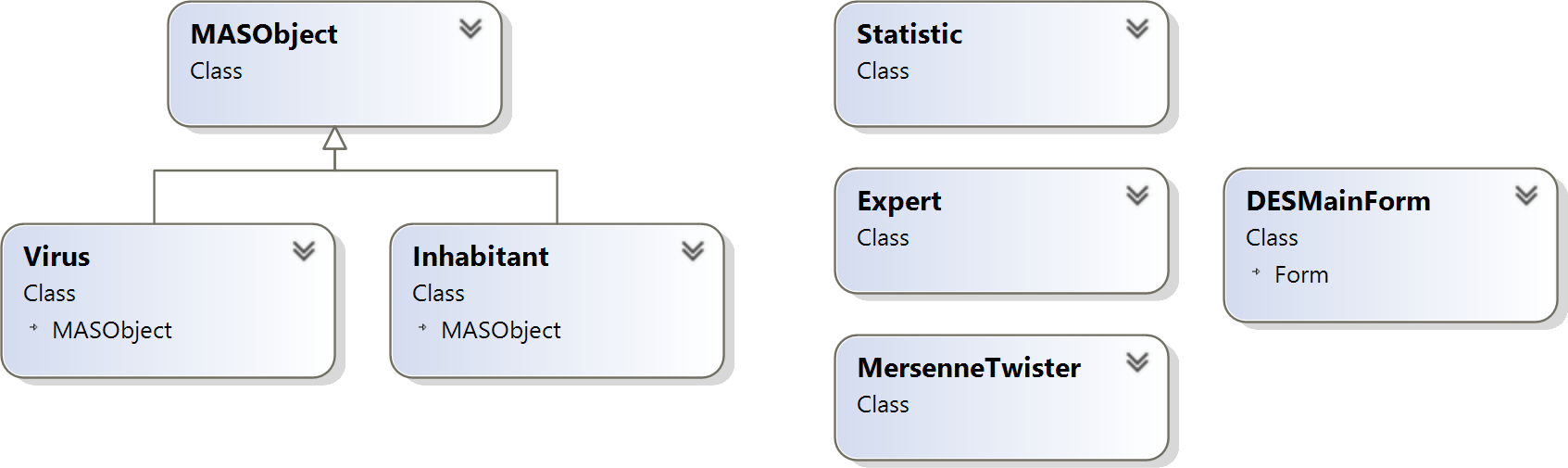
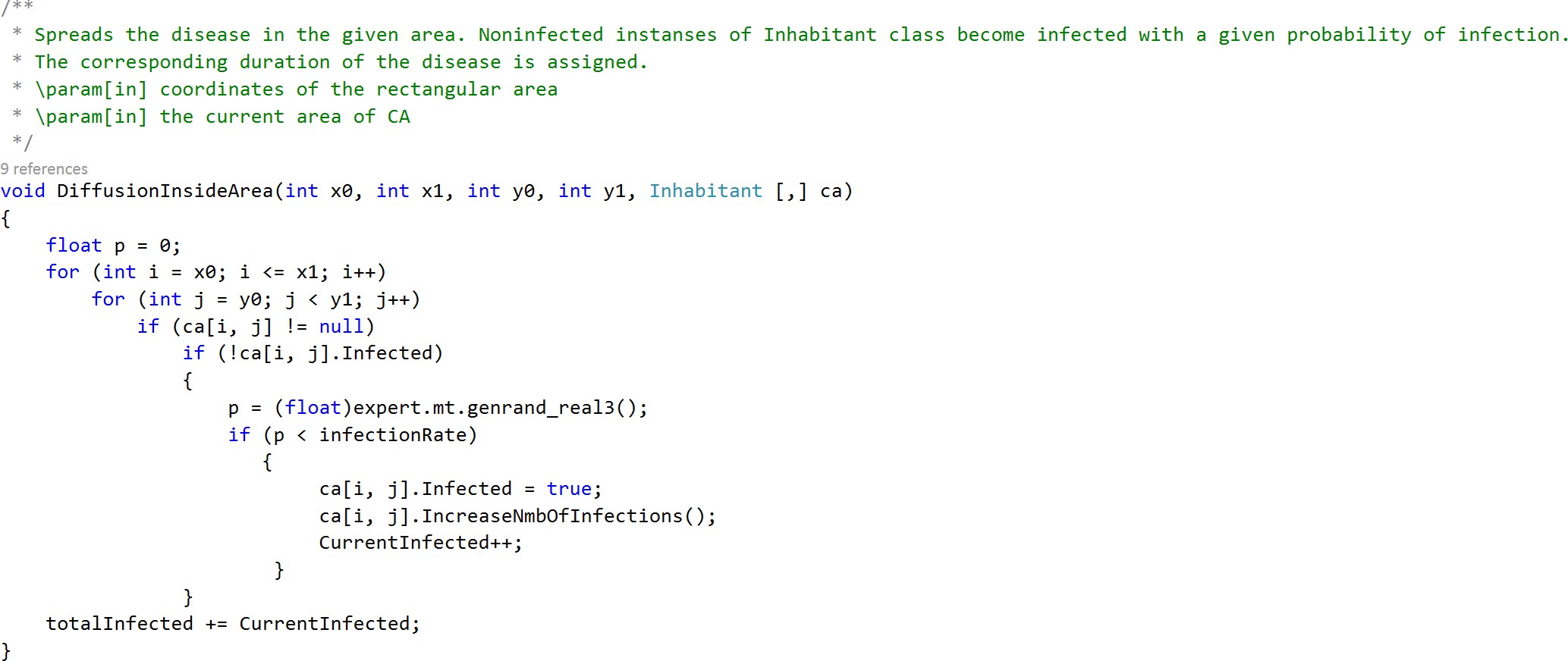
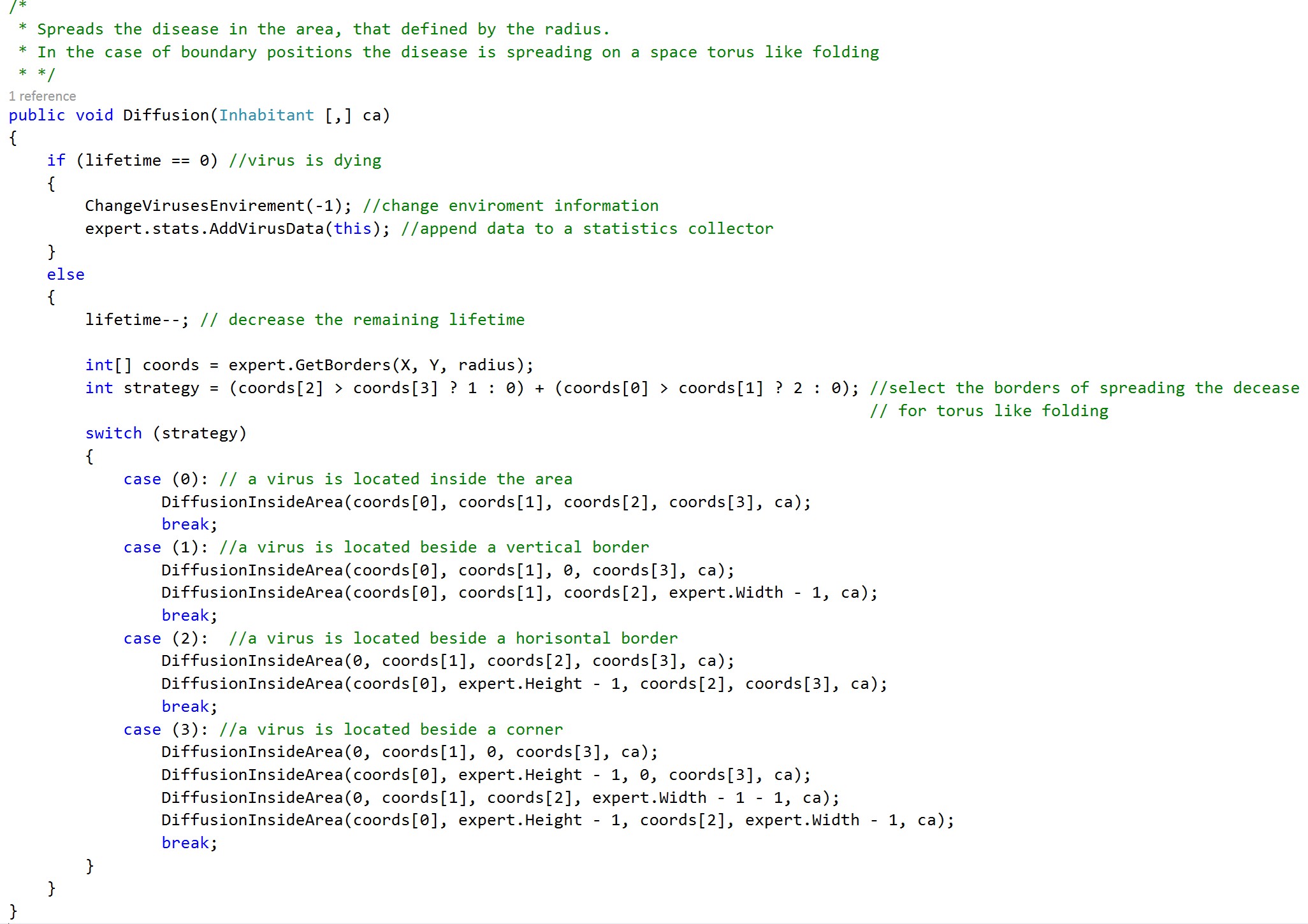


Figure 6: The class diagram of the project

### Class Virus

Viruses are created on each iteration by an *Expert*. During a lifetime each virus is collecting the amount of infected inhabitants on each iteration and sends this information to an *Expert* on its (virus’s) disappearance. For a virus only one action is defined: spreading of an infection. Since we use a torus-like space we have a special spreading procedure on the borders and in the corners (fig.7, a - the spreading are is divided on several parts), the spreading inside an rectangle is identical (fig.7, b).



* + - 1. The main spreading function (b) The function of spreading inside a rectangle area

Figure 7: The function of spreading of the infection

### Class Inhabitant

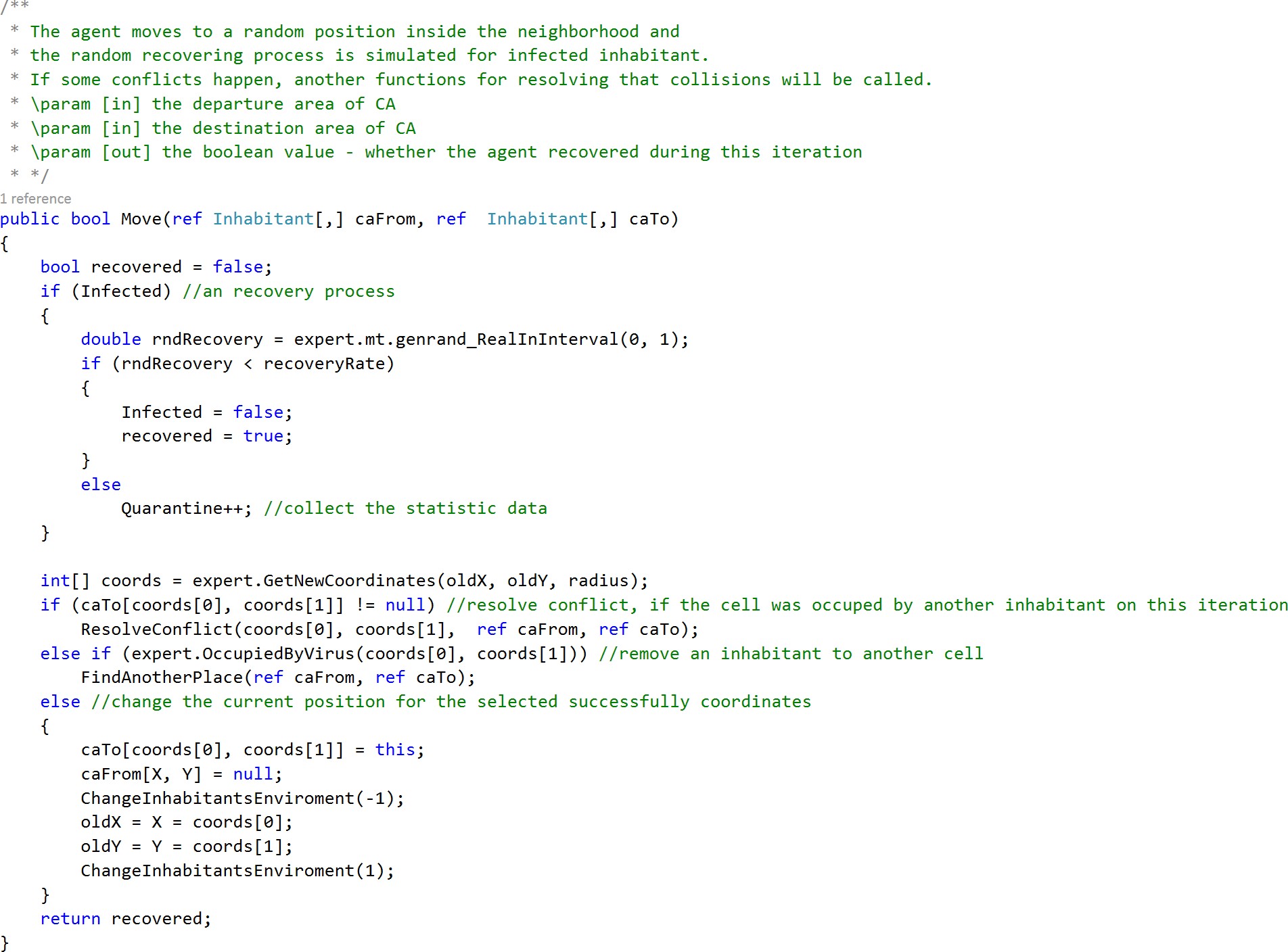
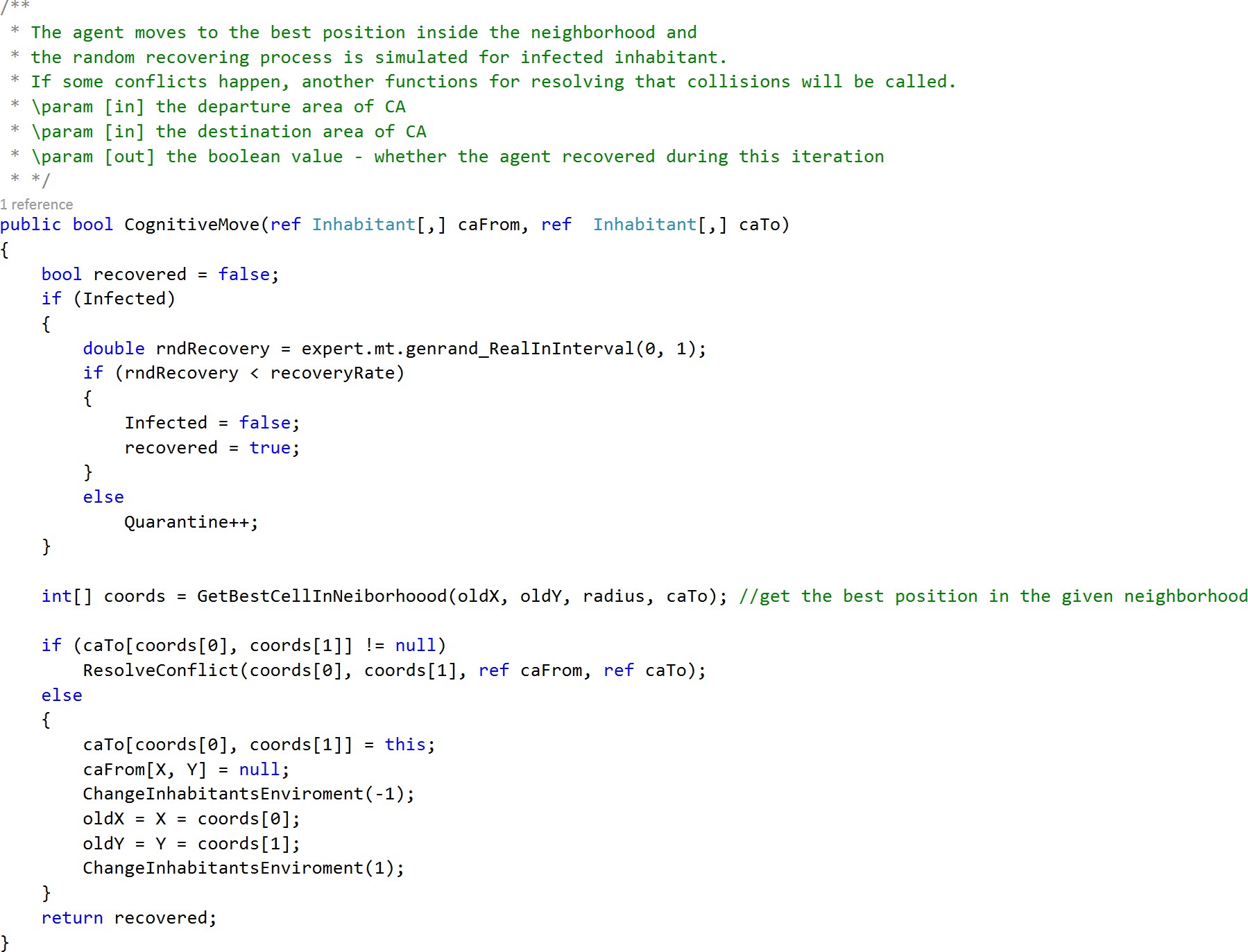
*An Expert* creates the population of inhabitants once, during a simulation process inhabitants can change their state (susceptible and infected) and move within a space. For inhabitants two types of moving are defined: random (fig.8,a) and intelligent (based on knowledge about an environment and social intentions, fig.8, b).

**The knowledge-based movements function** For intelligent movements of inhabitants (an UML dia- gram is represented on fig.9) the following options were determined:

* Movement to the best location

Giving up the place (which was taken on the current iteration) to the more healthy inhabitant in the case of a conflict

*•*

* + - 1. Random (b) Intelligent

Figure 8: The relocation functions of an inhabitant inside a neighborhood of the radius *radius*

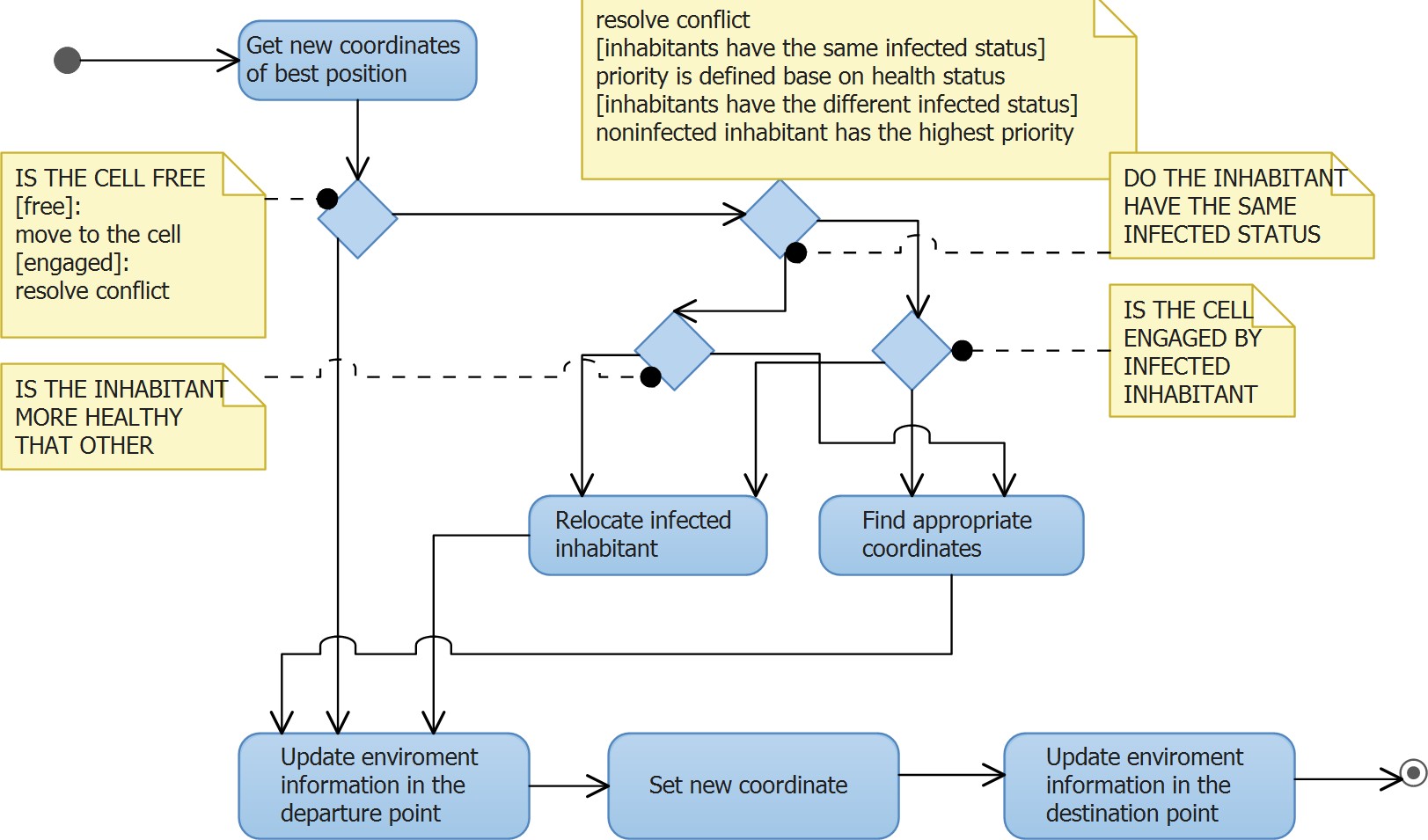


Figure 9: UML - Activity diagram of inhabitants movements

**Movement to the best location** in the neighborhood is a knowledge-based function. It is based on density of inhabitants and density of viruses. The function *CognitiveMove* (fig.8, b), the child function *GetBestCell* (fig.10) and *ResolveConflict* (fig. 11) make up the knowledge-based movement.

**Dislocation** Since the density of objects can be high, the expansion of neighborhood was provided in the case of a multitude of unsuccessful attempts of relocation.

## GUI

GUI (fig.13) makes provision for the variation the following parameters:

1. The output space;
2. The size of a cellular automata: *Width* and *Height* ;
3. The duration of a simulation: *Time*;

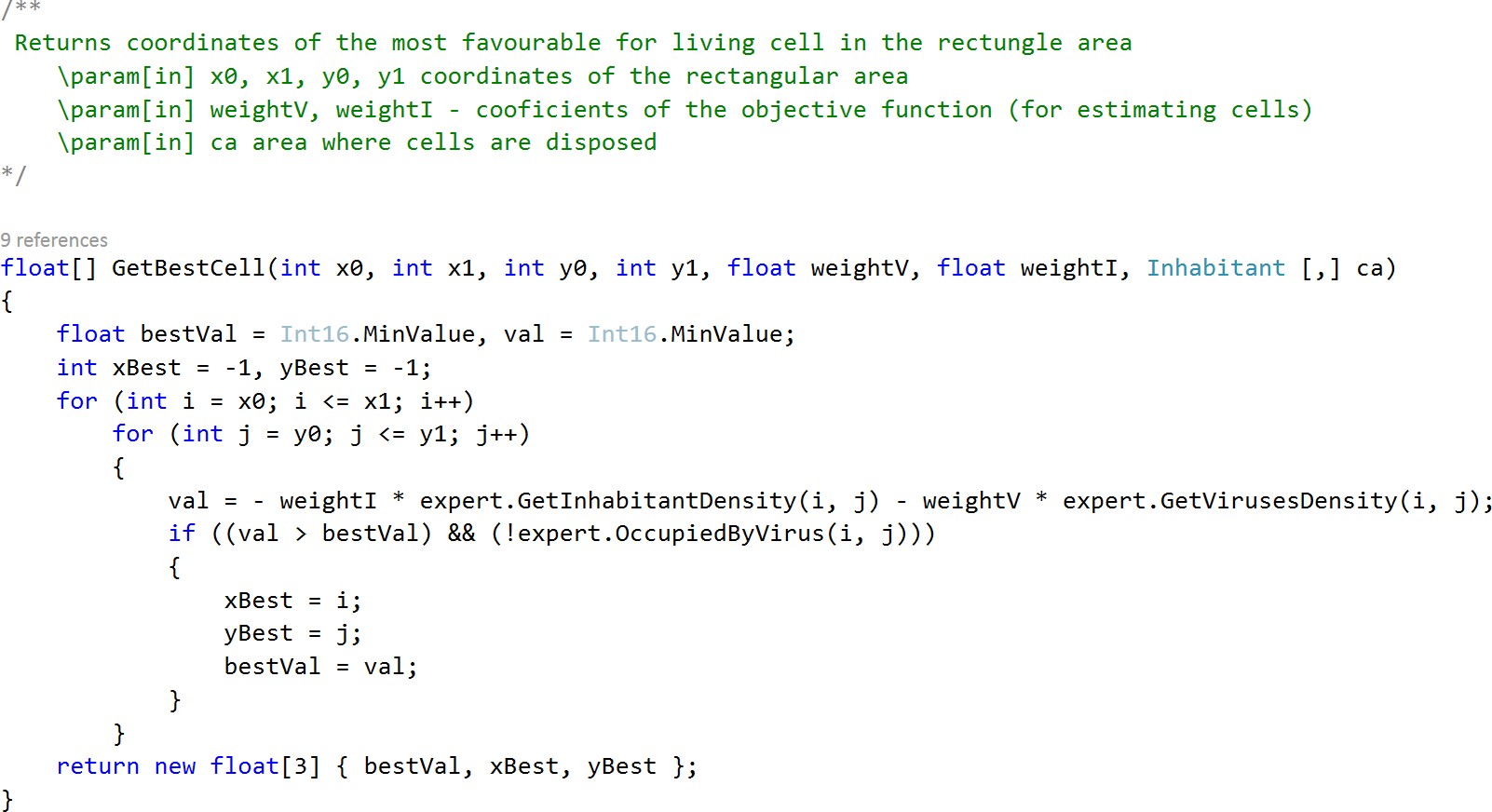


Figure 10: The function for selecting the best position in the neighborhood based on density of inhabitants (*Get- InhabitantsDensity*) and density of viruses (*GetVirusesDensity* ). Weights *weightI* and *weightV* are used to determine the importance of densities

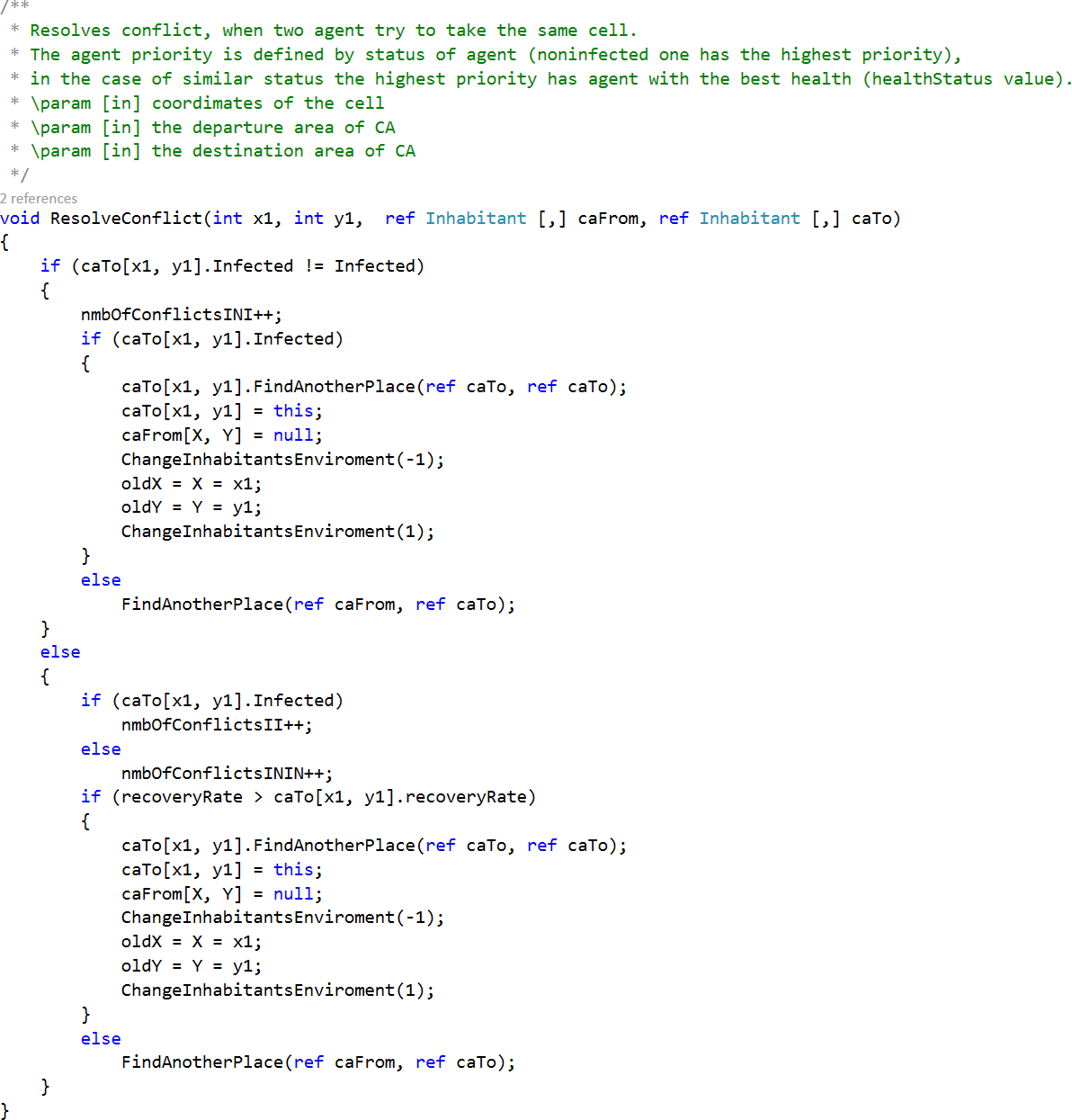


Figure 11: The function for resolving conflicts

1. Checkbox *Cognitive moving* defines the type of inhabitants movements;
2. Checkbox *Intermediate output* is used for tracking states of a cellular automata during a simulation;
3. *The rate of viruses* (defines the maximal share of cells, which will be taken by new viruses on each iteration);
4. The probability of becoming infected after falling into a neighborhood of a virus:*The infection proba- bility* or random for each virus - checkbox *Random infection rate for each virus*;
5. A virus lifetime (in iterations);
6. *The rate of inhabitants* (the share of cellular automata’s cells);

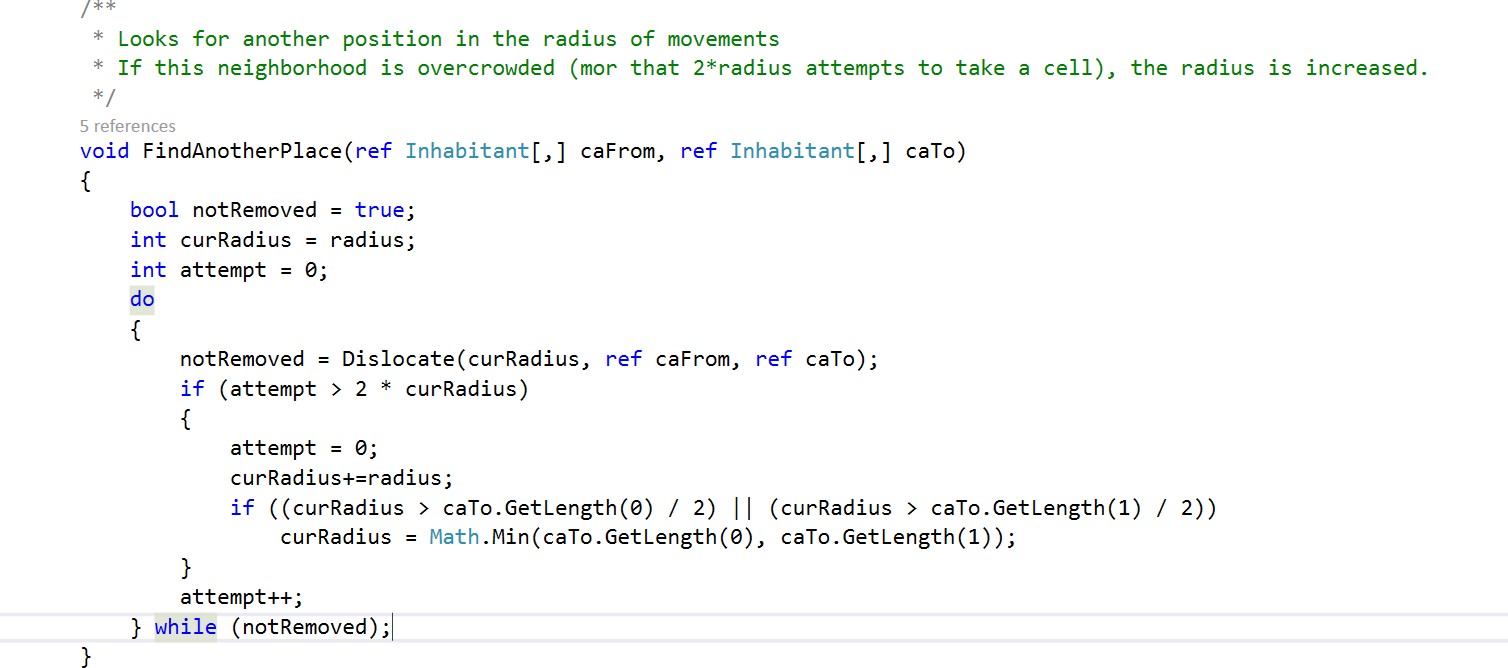


Figure 12: The function of change a position

1. The probability of becoming recovered on an iteration (being infected): *The recovery probability* or random for each inhabitant - checkbox *Random recovery rate for each virus* ;
2. The button *Start* is used to start simulation;
3. The button *Clear the output box* is used for clearing the output space.

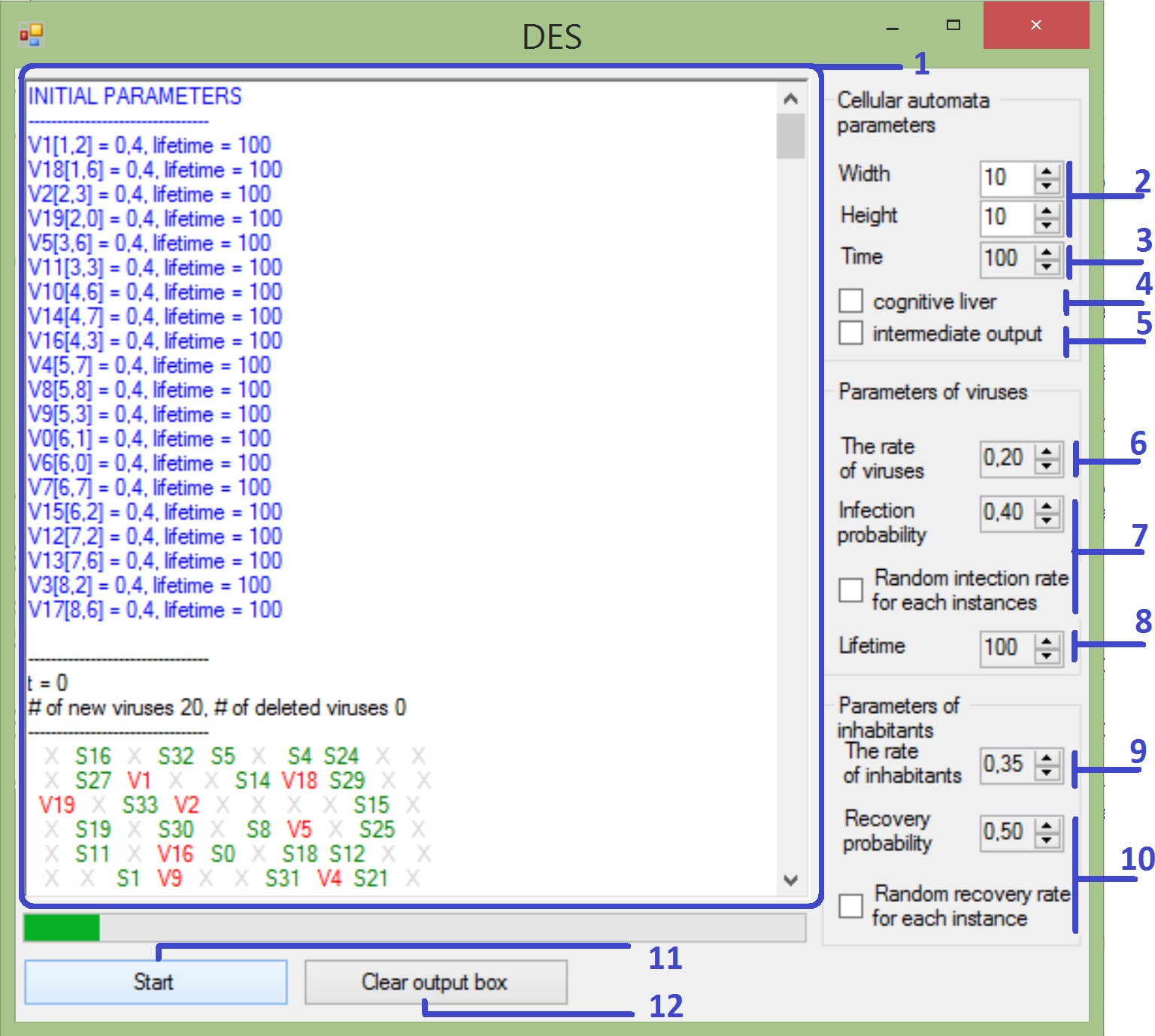


Figure 13: The main window of the application

## Statistics

We collect raw data, which is processed by Python data analysis tools. For inhabitants the following information is collected:

* Duration of disease
* The number of relocation for infected/susceptible inhabitants

The number of conflicts between infected / infected, susceptible / susceptible and infected / susceptible inhabitants

*•*

* The number of viruses (total / average / on each iteration)
* The amount infected and sustained inhabitants ( on average / on each iteration / by each virus)

# Analysis of results

The 10000 simulations for each set of parameters were used to investigate properties of the model. Futher, the following questions will be considered:

* The model evolution depending on various parameters
* The rate of infected and susceptible inhabitants w.r.t. different parameters of the model
* Time of simulations for two types of relocation strategies
* The number conflicts
* The average duration of the disease for an inhabitant

### Ways of the model evolution

There are several ways of the evolution:

The number of infected and susceptible inhabitants remain the same during the long period of time (a stable state from the beginning)

*•*

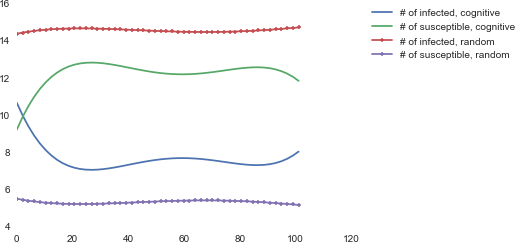
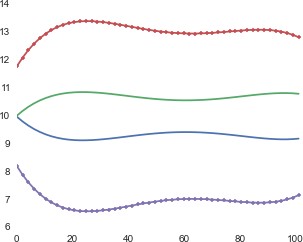
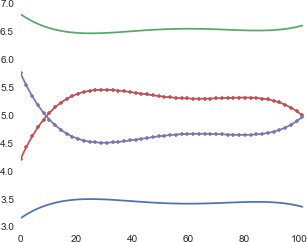
The proportion of infected and susceptible inhabitants is unstable and changes significantly from iter- ation to iteration (an unstable state)

*•*

The number of inhabitants of a certain type decreases/increases for the beginning and becomes stable after several iteration (the saturation process)

*•*

We can see from the fig.14 that the type of inhabitant movement has significant influence on the simulation results. Having the same probability of infection and recovery, but a small variation of density, the model evolution is mostly determined by the way of inhabitants moving. The relatively stable state of an epidemic with the random movements (c) opposes the state of halting of an epidemic during the simulation. The relatively stable ”no epidemic” state with cognitive moving (a) opposes the state of halting of epidemic with random moving.

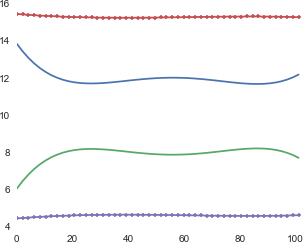
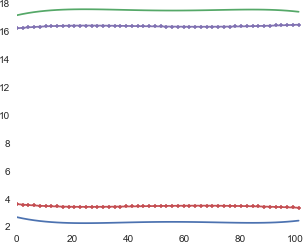
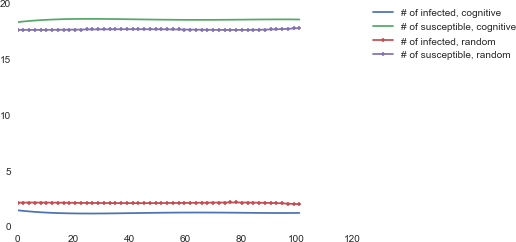


* + - 1. inhabitants rate = 0.1,(b) inhabitants rate = 0.2, (c) inhabitants rate = 0.2, viruses rate = 0.2

viruses rate = 0.05 viruses rate = 0.1

Figure 14: The averaged number of infected and susceptible inhabitants during 100 iteration, probability of recovery = 0.2, probability of infection = 0.2

The evolution of the model is also changed, when the rates of viruses and inhabitants remain the same, but values of infection and recovery probabilities are changed. In this case the type of movement affects the results in the same manner (fig.15).

(a) probability of recovery = 0.2,(b) probability of recovery = 0.8,(c) probability of recovery = 0.8, probability of infection

probability of infection = 0.8

probability of infection = 0.8

= 0.2

Figure 15: The averaged number of infected and susceptible inhabitants during 100 iteration, inhabitants rate = 0.2, viruses rate = 0.1, the number of viruses is constant and they are nonmovable

As the rate of occupied cells increases, the number of relocation (conflicts) increases. It happens due to strive of inhabitants to occupy the best location. As the rate of inhabitants and viruses increases, the share of the same best position for several inhabitants increases and hence, the number of conflicts for the same cell (in the case of knowledge-based movements) rises faster, then for random moving strategy.

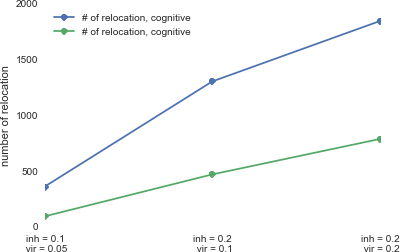


Figure 16: The number of relocations (conflicts) for different strategies of the cell selection

The time of simulations increases as well (as the number of relocations increases). The average time of simulation for random and intellegent cell selection is represented on the fig. 17.

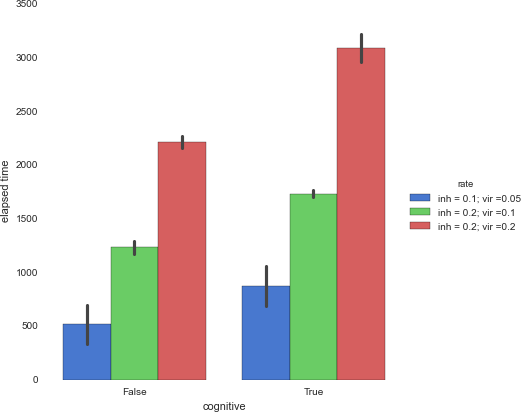
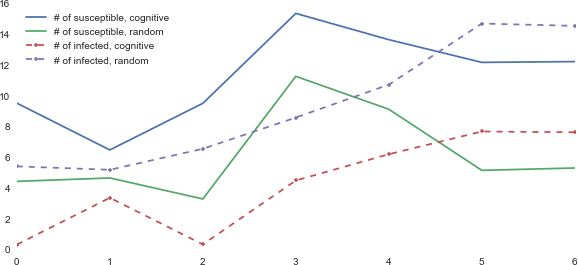
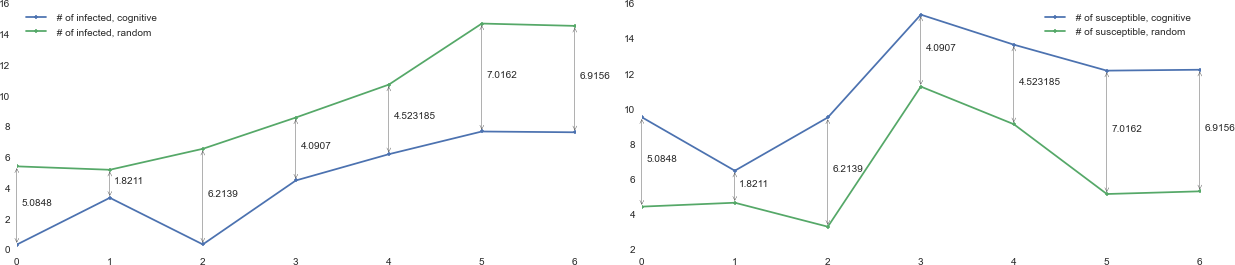


Figure 17: The average time of simulation for different density and two strategies of the cell selection

Since we consider SIS models of epidemic, each inhabitant can be infected and recovers several time.

**Long-distance relocation effect** As it was mentioned early, an inhabitant that cannot find good cell in its neighborhood removes to out of the neighborhood. As the rate of inhabitants increases such overcrowding



1. The disparity in infected inhabitants (b) The disparity in susceptible inhabi-(c) The summary disparity in inhabitants

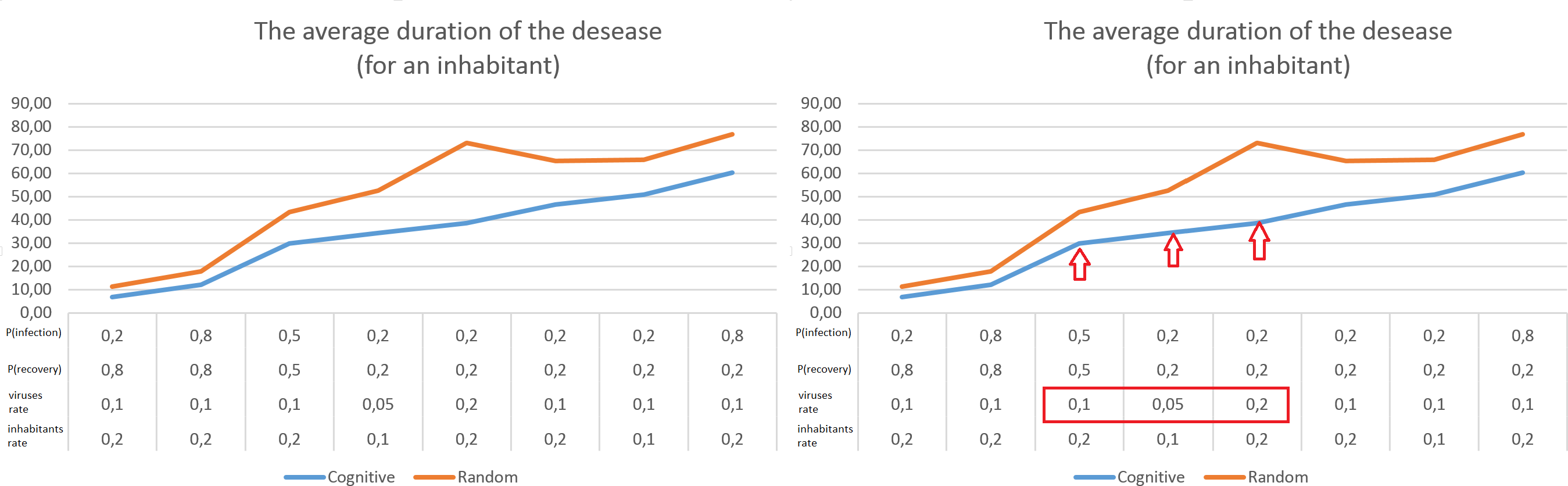
tants states

Figure 18: The difference of the average number of infected and susceptible inhabitants (averaged among 10000 simulation) with respect to different parameters (table 3)

Table 3: The parameters of simulations represented on fig. 18, the rows are correspond to the points on x-axis

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Number**  **of a point** | **inhabitants**  **rate** | **viruses**  **rate** | **viruses**  **lifetime** | **knowledge-based**  **mean** | **random**  **mean** |
| 0 | 0.10 | 0.05 | 1 | 0.40 | 5.49 |
| 1 | 0.10 | 0.05 | 101 | 3.44 | 5.27 |
| 2 | 0.10 | 0.10 | 1 | 0.42 | 6.63 |
| 3 | 0.20 | 0.10 | 1 | 4.57 | 8.66 |
| 4 | 0.20 | 0.10 | 101 | 6.28 | 10.80 |
| 5 | 0.20 | 0.20 | 1 | 7.76 | 14.77 |
| 6 | 0.20 | 0.20 | 101 | 7.70 | 14.62 |

on condition nonmovable viruses may lead to decreasing the average duration of disease of an inhabitant due to relocation takes a place beyond the virus’s scope. It means that there is some point of the population density, such that exceeding this value leads to decreasing the duration of disease (the number of infections) because of inhabitants relocations into the area free of viruses (fig.19).



* 1. (b) The effect of long-distance movements

Figure 19: The average duration of the disease for an inhabitant

# Conclusion

In the report the SIS-like model has been considered. Unlike the classical SIS model, accordingly to the introduced model the spreading of a disease is permitted from a virus to an inhabitant, while in the original model a disease spreads from an inhabitant to another one. More than that, our model allows dynamic modification a neighborhood of agents (that is equivalent to changing the graph structure - edges between ”an inhabitant node” and ”a virus node” in the case of graph representation of agents).

In the report we presented the concepts of agent (inhabitants and viruses) and described the implemen- tation details.

Performed series of simulations with the introduced model were analyzed and summarized in this report:

the evolution of the model, the rate of susceptible and infected inhabitants, the number of conflicts and relocations with respect to different parameters were shown. Some characteristic properties were discovered and represented in the last chapter of the report.

# Python code (fragments)

Polynomial fit

i m p o r t numpy as np i m p o r t p a n da s as pd

i m p o r t m a t p l o t l i b . p y p l o t as plt i m p o r t s e a b o r n as sns

def g e t \_ a p p r o x ( vals ) :

y = vals . a s t y p e ( float )

x = np . array ( range ( y . shape [0]) ) r e t u r n np . p o l y 1 d ( np . p o l y f i t ( x , y , 5) )

Plot of aggregated parameters of the model (the number of infected and susceptible inhabitants) during simulations

for key in d a t a \_ s t a m p . keys () : print " key . = . " , key

print d i f f e r e n t \_ p a r a m e t e r s [ key : key +1]

c r i t e r i o n = logs . Stamp . map ( l a m b d a x : x in d a t a \_ s t a m p [ key ]) d f \_ l o g s \_ s a m e \_ p a r a m e t e r s = logs [ c r i t e r i o n ]

a g g \_ d f = d f \_ l o g s \_ s a m e \_ p a r a m e t e r s . g r o u p b y ( by = " D a t a T y p e " ) . mean () . r e s e t \_ i n d e x ()

val = a g g \_ d f [ a g g \_ d f . D a t a T y p e == " c n t I n f e c t e d " ]. v a l u e s . T [ 2:]. T + a g g \_ d f [ a g g \_ d f . D a t a T y p e == " c n t N o n I n f e c t e d " ]. v a l u e s . T [ 2:]. T

plt . plot ( a g g \_ d f [ a g g \_ d f . D a t a T y p e == " c n t C u r I n f e c t e d " ]. T [2:] , label = " c n t C u r I n f e c t e d " )

plt . plot ( a g g \_ d f [ a g g \_ d f . D a t a T y p e == " c n t I n f e c t e d " ]. T [2:] , label = " c n t I n f e c t e d " )

plt . plot ( a g g \_ d f [ a g g \_ d f . D a t a T y p e == " c n t N o n I n f e c t e d " ]. T [2:] , label = " c n t N o n I n f e c t e d " )

plt . l e g e n d ( b b o x \_ t o \_ a n c h o r =(1.05 , 1) , loc =2 , b o r d e r a x e s p a d =0.) plt . show ()

Average simulation time for different strategies of selecting new cells and densities of viruses and inhab- itants (fig.17).

a g g \_ d f [ " rate " ] = pd . S e r i e s ( data = l , index = a g g \_ d f . index )

g = sns . f a c t o r p l o t ( x = " c o g n i t i v e " , y = " . e l a p s e d . time " , hue = " rate " , data = agg\_df , size =6 , kind = " bar " , p a l e t t e = " muted " )

g . d e s p i n e ( left = True )

Comparative graphs of inhabitants states with different parameters (corresponds to fig.18)

v a l \_ c o g \_ i n f = list ( res [ res . D a t a T y p e == " c n t I n f e c t e d " ]. n m b \_ c o g ) v a l \_ n o t \_ c o g \_ i n f = list ( res [ res . D a t a T y p e == " c n t I n f e c t e d " ]. n m b \_ n o n c o g )

v a l \_ c o g \_ n o n I n f = list ( res [ res . D a t a T y p e == " c n t N o n I n f e c t e d " ]. n m b \_ c o g ) v a l \_ n o t \_ c o g \_ n o n I n f = list ( res [ res . D a t a T y p e == " c n t N o n I n f e c t e d " ]. n m b \_ n o n c o g

)

fig = plt . f i g u r e (1 , f i g s i z e =(10 , 15) )

ax = fig . a d d \_ s u b p l o t (311 , a u t o s c a l e \_ o n = True )

x \_ c o o r d s = range ( len ( v a l \_ n o t \_ c o g \_ i n f ) )

ax . plot ( val\_ cog\_ inf , ’. - ’ , label = " . # . of . infected , . c o g n i t i v e " ) ax . plot ( v a l \_ n o t \_ c o g \_ i n f , ’. - ’ , label = " . # . of . infected , . r a n d o m " ) ax . l e g e n d ( loc = 0)

for i in x r a n g e ( len ( v a l \_ n o t \_ c o g \_ i n f ) ) :

dif = abs ( v a l \_ c o g \_ i n f [ i ] - v a l \_ n o t \_ c o g \_ i n f [ i ])

plt . a n n o t a t e ( ’ ’ , xy =( i , v a l \_ c o g \_ i n f [ i ]) , x y c o o r d s = ’ data ’ , x y t e x t =( i , v a l \_ n o t \_ c o g \_ i n f [ i ]) , t e x t c o o r d s = ’ data ’ ,

a r r o w p r o p s ={ ’ arrowstyle ’: ’ < - > ’})

plt . a n n o t a t e (

dif , xy =( i , max ( v a l \_ c o g \_ i n f [ i ] , v a l \_ n o t \_ c o g \_ i n f [ i ]) - dif /2) , x y c o o r d s = ’ data ’ ,

x y t e x t =(5 , 0) , t e x t c o o r d s = ’ o f f s e t points ’)

fig = plt . f i g u r e (2 , f i g s i z e =(10 , 15) )

ax = fig . a d d \_ s u b p l o t (312 , a u t o s c a l e \_ o n = True )

x \_ c o o r d s = range ( len ( v a l \_ c o g \_ n o n I n f ) )

ax . plot ( v a l \_ c o g \_ n o n I n f , ’. - ’ , label = " . # . of . susceptible , . c o g n i t i v e " ) ax . plot ( v a l \_ n o t \_ c o g \_ n o n I n f , ’. - ’ , label = " . # . of . susceptible , . r a n d o m " ) ax . l e g e n d ( loc = 0)

for i in x r a n g e ( len ( v a l \_ c o g \_ n o n I n f ) ) :

dif = abs ( v a l \_ c o g \_ n o n I n f [ i ] - v a l \_ n o t \_ c o g \_ n o n I n f [ i ])

plt . a n n o t a t e ( ’ ’ , xy =( i , v a l \_ c o g \_ n o n I n f [ i ]) , x y c o o r d s = ’ data ’ , x y t e x t =( i , v a l \_ n o t \_ c o g \_ n o n I n f [ i ]) , t e x t c o o r d s = ’ data ’ ,

a r r o w p r o p s ={ ’ arrowstyle ’: ’ < - > ’})

plt . a n n o t a t e ( dif , xy =( i , max ( v a l \_ c o g \_ n o n I n f [ i ] , v a l \_ n o t \_ c o g \_ n o n I n f [ i ])

- dif /2) , x y c o o r d s = ’ data ’ ,

x y t e x t =(5 , 0) , t e x t c o o r d s = ’ o f f s e t points ’)

fig = plt . f i g u r e (3 , f i g s i z e =(10 , 15) )

ax = fig . a d d \_ s u b p l o t (313 , a u t o s c a l e \_ o n = True )

ax . plot ( v a l \_ c o g \_ n o n I n f , label = " # . of . susceptible , . c o g n i t i v e " ) ax . plot ( v a l \_ n o t \_ c o g \_ n o n I n f , label = " # . of . susceptible , . r a n d o m " ) ax . plot ( val\_ cog\_ inf , ’. - - ’ , label = " # . of . infected , . c o g n i t i v e " ) ax . plot ( v a l \_ n o t \_ c o g \_ i n f , ’. - - ’ , label = " # . of . infected , . r a n d o m " ) ax . l e g e n d ( loc = 0)

1. **Source code (fragments)**

## Class ”Virus”

using S y s t e m ;

using S y s t e m . C o l l e c t i o n s . G e n e r i c ; using S y s t e m . Linq ;

using S y s t e m . Text ;

using S y s t e m . T h r e a d i n g . Tasks ;

n a m e s p a c e D E S \_ L a b 6

{

/\* \*

* V i r u s e s are s t a t i o n a r y objects , s p r e a d i n g an i n f e c t i o n
* among i n s t a n c e s of the I n h a b i t a n t class

\*

* I n s t a n c e s of this class may d i f f e r the r a d i u s of spreading ,
* the p r o b a b i l i t y of the i n f e c t i o n by a c o n t a c t with n o n i n f e c t e d inhabitants , d u r a t i o n of life .
* The p a r e n t class is M A S O b j e c t

\*/

class Virus : M A S O b j e c t

{

// / < summary >

// / P r o b a b i l i t y of the i n f e c t i o n

// / </ summary >

float i n f e c t i o n R a t e ;

// / < summary >

// / The n u m b e r of i t e r a t i n g s d u r i n g the virus is e x i s t i n g

// / </ summary > int l i f e t i m e = 0;

// / < summary >

// / The total n u m b er of i n f e c t e d a g e n t s by the virus

// / </ summary >

int t o t a l I n f e c t e d = 0;

// / < summary >

// / The n u m b e r of i n f e c t e d a g e n t s by the virus on c u r r e n t i t e r a t i o n ( s t a t i s t i c v a r i a b l e )

// / </ summary >

p u b l i c int C u r r e n t I n f e c t e d { get ; set ; }

p u b l i c float L i f e t i m e

{

get { r e t u r n l i f e t i m e ; }

}

p u b l i c float C o n t a g i o u s n e s s

{

get { r e t u r n i n f e c t i o n R a t e ; }

}

/\* \*

* + C r e a t e s the i n s t a n c e of the Virus class
  + in the r a n d o m p o s i t i o n at u n i q u e place for each i n s t a n c e
  + \ param [ in ] e x p e rt an i n s t a n c e of the E x p e r t class
  + \ param [ in ] id the u n i q u e n u m b e r of a new - g e n e r a t e d virus

\*/

p u b l i c Virus ( E x p e r t expert , int id )

: base ( expert , id )

{

this . e x p e r t = e x p e r t ;

l i f e t i m e = e x p e r t . V i r u s e s L i f e t i m e ; i n f e c t i o n R a t e = e x p e r t . I n f e c t i o n R a t e ; if ( i n f e c t i o n R a t e == -1)

i n f e c t i o n R a t e = base . G e t R e a l N u m b e r (0 , ( float ) 0.6) ; bool c o l l i s i o n = false ;

int [] c o o r d s ; do

{

c o o r d s = e x p e r t . G e t S o m e C o o r d i n a t e s () ;

if ( this . e x p e r t . ca [ c o o r d s [0] , c o o r d s [1]] == null ) c o l l i s i o n = e x p e r t . D e t e c t V i r u s C o l l i s i o n ( c o o r d s [0] ,

c o o r d s [1] , ref e x p e r t . ca ) ;

} while ( c o l l i s i o n ) ; X = c o o r d s [0];

Y = c o o r d s [1]; C h a n g e V i r u s e s E n v i r e m e n t (1) ;

}

/\* \*

* + S p r e a d s the d i s e a s e in the given area . N o n i n f e c t e d i n s t a n s e s of I n h a b i t a n t class b e c o m e i n f e c t e d with a given p r o b a b i l i t y of

i n f e c t i o n .

* + The c o r r e s p o n d i n g d u r a t i o n of the d i s e a s e is a s s i g n e d .
  + \ param [ in ] x0 , x1 , y0 , y1 c o o r d i n a t e s of a r e c t a n g u l a r area
  + \ param [ in ] ca an area of a c e l l u l a r a u t o m a t a

\*/

void D i f f u s i o n I n s i d e A r e a ( int x0 , int x1 , int y0 , int y1 , I n h a b i t a n t [ ,] ca )

{

float p = 0;

for ( int i = x0 ; i <= x1 ; i ++) for ( int j = y0 ; j < y1 ; j ++)

if ( ca [ i , j ] != null )

if (! ca [ i , j ]. I n f e c t e d )

{

p = ( float ) e x p e r t . mt . g e n r a n d \_ r e a l 3 () ; if ( p < i n f e c t i o n R a t e )

{

ca [ i , j ]. I n f e c t e d = true ;

ca [ i , j ]. I n c r e a s e N m b O f I n f e c t i o n s () ; C u r r e n t I n f e c t e d ++;

}

}

t o t a l I n f e c t e d += C u r r e n t I n f e c t e d ;

}

/\*

* + S p r e a d s the d i s e a s e in the area , that d e f i n e d by the r ad i u s .
  + In the case of b o u n d a r y p o s i t i o n s the d i s e a s e is s p r e a d i n g on a space torus like f o l d i n g
  + \ param [ in ] ca an area of a c e l l u l a r a u t o m a t a

\* \*/

p u b l i c void D i f f u s i o n ( I n h a b i t a n t [ ,] ca )

{

if ( l i f e t i m e == 0) // virus is dying

{

C h a n g e V i r u s e s E n v i r e m e n t ( -1) ; // c h a n g e e n v i r o m e n t i n f o r m a t i o n

}

else

{

e x p e r t . stats . A d d V i r u s D a t a ( this ) ; // a p p e n d data to a s t a t i s t i c s c o l l e c t o r

lifetime - -; // d e c r e a s e the r e m a i n i n g l i f e t i m e

int [] c o o r d s = e x p e r t . G e t B o r d e r s ( X , Y , r a di u s ) ;

int s t r a t e g y = ( c o o r d s [2] > c o o r d s [3] ? 1 : 0) + ( c o o r d s [0] > c o o r d s [1] ? 2 : 0) ; // s e l e c t the b o r d e r s of s p r e a d i n g the d e c e a s e

s w i t c h ( s t r a t e g y )

{

case (0) : // a virus is l o c a t e d i n s i d e the area D i f f u s i o n I n s i d e A r e a ( c o o r d s [0] , c o o r d s [1] , c o o r d s

[2] , c o o r d s [3] , ca ) ; break ;

case (1) : // a virus is l o c a t e d b e s i d e a v e r t i c a l b o r d e r

D i f f u s i o n I n s i d e A r e a ( c o o r d s [0] , c o o r d s [1] , 0 , c o o r d s [3] , ca ) ;

D i f f u s i o n I n s i d e A r e a ( c o o r d s [0] , c o o r d s [1] , c o o r d s [2] , e x p e r t . Width - 1 , ca ) ;

break ;

case (2) : // a virus is l o c a t e d b e s i d e a h o r i s o n t a l b o r d e r

D i f f u s i o n I n s i d e A r e a (0 , c o o r d s [1] , c o o r d s [2] , c o o r d s [3] , ca ) ;

D i f f u s i o n I n s i d e A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , c o o r d s [2] , c o o r d s [3] , ca ) ;

break ;

case (3) : // a virus is l o c a t e d b e s i d e a c o r n e r D i f f u s i o n I n s i d e A r e a (0 , c o o r d s [1] , 0 , c o o r d s [3] , ca

) ;

D i f f u s i o n I n s i d e A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , 0 , c o o r d s [3] , ca ) ;

D i f f u s i o n I n s i d e A r e a (0 , c o o r d s [1] , c o o r d s [2] , e x p e r t . Width - 1 - 1 , ca ) ;

D i f f u s i o n I n s i d e A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , c o o r d s [2] , e x p e r t . Width - 1 , ca ) ;

break ;

}

}

}

/\* \*

* U p d a t e s i n f o r m a t i o n about v i r u s e s in a given area
* I n c r e a s e s rate of viruses , since a new virus a p p e a r s
* \ param [ in ] x0 , x1 , y0 , y2 c o o r d i n a t e s of the r e c t a n g u l a r area
* \ param [ in ] sign the sign of c h a n g e s ( i n c r e a s i n g or d e c r e a s i n g )

\* \*/

void C h a n g e V i r u s e s I n A r e a ( int x0 , int x1 , int y0 , int y1 , short sign )

{

for ( int i = x0 ; i <= x1 ; i ++) for ( int j = y0 ; j < y1 ; j ++)

e x p e r t . I n c r e a s e V i r s u s D e n s i t y ( i , j ,( float ) sign / s q u a r e ) ;

}

/\* \*

* + U p d a t e s i n f o r m a t i o n about v i r u s e s in a given n e i g b o r h o o d
  + I n c r e a s e s rate of viruses , since a new virus a p p e a r s
  + \ param [ in ] an area of a c e l l u l a r a u t o m a t a

\* \*/

void C h a n g e V i r u s e s E n v i r e m e n t ( short sign )

{

int [] c o o r d s = e x p e r t . G e t B o r d e r s ( X , Y , r a d i u s ) ;

int s t r a t e g y = ( c o o r d s [2] > c o o r d s [3] ? 1 : 0) + ( c o o r d s [0] > c o o r d s [1] ? 2 : 0) ;

s w i t c h ( s t r a t e g y )

{

case (0) :

C h a n g e V i r u s e s I n A r e a ( c o o r d s [0] , c o o r d s [1] , c o o r d s [2] , c o o r d s [3] , sign ) ;

break ; case (1) :

C h a n g e V i r u s e s I n A r e a ( c o o r d s [0] , c o o r d s [1] , 0 , c o o r d s [3] , sign ) ;

C h a n g e V i r u s e s I n A r e a ( c o o r d s [0] , c o o r d s [1] , c o o r d s [2] , e x p e r t . Width - 1 , sign ) ;

break ; case (2) :

C h a n g e V i r u s e s I n A r e a (0 , c o o r d s [1] , c o o r d s [2] , c o o r d s [3] , sign ) ;

C h a n g e V i r u s e s I n A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , c o o r d s [2] , c o o r d s [3] , sign ) ;

break ; case (3) :

C h a n g e V i r u s e s I n A r e a (0 , c o o r d s [1] , 0 , c o o r d s [3] , sign ) ; C h a n g e V i r u s e s I n A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , 0 ,

c o o r d s [3] , sign ) ;

C h a n g e V i r u s e s I n A r e a (0 , c o o r d s [1] , c o o r d s [2] , e x p e r t .

Width - 1 - 1 , sign ) ;

C h a n g e V i r u s e s I n A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , c o o r d s [2] , e x p e r t . Width - 1 , sign ) ;

break ;

}

}

}

}

## Class ”Inhabitant”

using S y s t e m ;

n a m e s p a c e D E S \_ L a b 6

{

/\* \*

* \ brief I n h a b i t a n t s are m o v i n g agents , s u s c e p t i b l e to i n f e c t i o n .

\*

* I n h a b i t a n t s have a r a d i u s of m o v i n g ( a m a x i m a l d i s t a n c e of a m o v e m e n t ) . Each i n s t a n c e also has a s p e c i f i c h e a l t h state
* If an i n s t a n c e moves into an o c c u p i e d cell , a c o n f l i c t is s o l v e d t a k i n g into a c c o u n t a s t a t u s of the d i s e a s e ( i n f e c t e d / n o n i n f e c t e d )

\* \*/

class I n h a b i t a n t : M A S O b j e c t

{

// / < summary >

// / A p r o b a b i l i t y to r e c o v e r ( d i f u n e h e a l t h state )

// / </ summary >

float r e c o v e r y R a t e =0; //

// / < summary >

// / Share of i n f e c t e d i n h a b i t a n t s in the m o v e m e n t area

// / </ summary > float i n f e c t e d R a t e ;

// / < summary >

// / The n u m b e r of c o n f l i c t e s Infected - I n f e c t e d

// / </ summary >

int n m b O f C o n f l i c t s I I = 0;

// / < summary >

// / The n u m b e r of c o n f l i c t s Infected - N o n i n f e c t e d

// / </ summary >

int n m b O f C o n f l i c t s I N I = 0;

// / < summary >

// / The n u m b e r of c o n f l i c t s Non Infected - N o n I n f e c t e d

// / </ summary >

int n m b O f C o n f l i c t s I N I N = 0;

// / < summary >

// / Last c o o r d i n a t e s of the s u c c e s s f u l m o v e m e n t

// / </ summary > int oldX , oldY ;

// / < summary >

// / The n u m b e r of i n f e c t i o n s

// / </ summary >

int n m b O f I n f e c t i o n s = 0;

// / < summary >

// / The total n u m b er of r e l o c a t i o n d u r i n g the s i m u l a t i o n when i n h a b u t a n t is n o n i n f e c t e d

// / </ summary >

int n m b T o t a l R e l o c a t i o n N o n I n f e c t e d = 0;

// / < summary >

// / W e i g h t s for v i r u s e s and i n h a b i t a n t s density , which are used for s e l e c t i n g the best p o s i t i o n in a n e i g h b o r h o o d

// / </ summary > float wV = 1;

float wI = ( float ) 0.3;

p u b l i c int N m b O f C o n f l i c t s I I

{

get { r e t u r n n m b O f C o n f l i c t s I I ; }

}

p u b l i c int N m b O f C o n f l i c t s I N I

{

get { r e t u r n n m b O f C o n f l i c t s I N I ; }

}

p u b l i c int N m b O f C o n f l i c t s I N I N

{

get { r e t u r n n m b O f C o n f l i c t s I N I N ; }

}

p u b l i c float R e c o v e r y R a t e

{

get { r e t u r n r e c o v e r y R a t e ; }

}

// / < summary >

// / I n d e c t e d s t a t u s

// / </ summary >

p u b l i c bool I n f e c t e d { get ; set ; }

// / < summary >

// / The d u r a t i o n of d i s e a s e ( how long the agent was infected , n u mb e r of i t e r a t i o n s )

// / </ summary >

p u b l i c int Q u a r a n t i n e { get ; set ; }

p u b l i c void I n c r e a s e N m b O f I n f e c t i o n s ()

{

n m b O f I n f e c t i o n s ++;

}

// / < summary >

// / The total n u m b er of r e l o c a t i o n d u r i n g the s i m u l a t i o n when i n h a b u t a n t is i n f e c t e d

// / </ summary >

int n m b T o t a l R e l o c a t i o n I n f e c t e d = 0; p u b l i c int N m b T o t a l R e l o c a t i o n I n f e c t e d

{

get { r e t u r n n m b T o t a l R e l o c a t i o n I n f e c t e d ; }

}

p u b l i c int N m b T o t a l R e l o c a t i o n N o n I n f e c t e d

{

get { r e t u r n n m b T o t a l R e l o c a t i o n N o n I n f e c t e d ; }

}

/\* \*

* + C r e a t e s the i n s t a n c e of the I n s t a n t class
  + in the r a n d o m p o s i t i o n in u n i q u e place for each i n s t a n c e ( w i t h o u t c o l l i s i o n s with other i n s t a n c e s of I n h a b i t a n t and Virus c l a s s e s )
  + \ param [ in ] e x p e rt an i n s t a n c e of an E x p e r t class
  + \ param [ in ] id an id of a new - g e n e r a t e d i n s t a n c e

\*/

p u b l i c I n h a b i t a n t ( E x p e r t expert , int id )

: base ( expert , id )

{

this . e x p e r t = e x p e r t ;

r e c o v e r y R a t e = e x p e r t . R e c o v e r y R a t e ; if ( r e c o v e r y R a t e == -1)

r e c o v e r y R a t e = ( float ) e x p e r t . mt . g e n r a n d \_ R e a l I n I n t e r v a l (0 , 1) ;

int [] c o o r d s ;

bool c o l l i s i o n = false ; do

{

c o o r d s = e x p e r t . G e t S o m e C o o r d i n a t e s () ;

c o l l i s i o n = e x p e r t . D e t e c t I n h a b i t a n t C o l l i s i o n ( c o o r d s [0] , c o o r d s [1]) ;

} while ( c o l l i s i o n ) ;

n m b O f I n f e c t i o n s = 0; oldX = X = c o o r d s [0]; oldY = Y = c o o r d s [1];

I n f e c t e d = false ; S e t I n f e c t e d R a t e ( e x p e r t . ca ) ; C h a n g e I n h a b i t a n t s E n v i r o m e n t (1) ;

}

/\*

* + C a l c u l a t e s the n u m b e r of the i n f e c t e d a g e n t s in the given r e c t a n g u l a r area .
  + \ param [ in ] x0 , x1 , y0 , y1 of a r e c t a n g u l a r area
  + \ param [ in ] ca a c e l l u l a r a u t o m a t a area
  + \ param [ out ] i n f e c t e d R a t e the n u m b e r of i n f e c t e d i n h a b i t a n t s in a given r e c t a n g l e area

\* \*/

int S e t I n f e c t e d I n s i d e A r e a ( int x0 , int x1 , int y0 , int y1 , I n h a b i t a n t [ ,] ca )

{

int p o p u l a t i o n = 0; int i n f e c t e d R a t e = 0;

for ( int i = x0 ; i <= x1 ; i ++)

for ( int j = y0 ; j <= y1 ; j ++) if ( ca [ i , j ] != null )

{

p o p u l a t i o n ++;

if ( ca [ i , j ]. I n f e c t e d ) i n f e c t e d R a t e ++;

}

r e t u r n i n f e c t e d R a t e ;

}

/\* \*

* + C a l c u l a t e s the rate of the i n f e c t e d a g e n t s in a new h e i g h b o r h o o d of d e s t i n a t i o n cell
  + In the case of b o u n d a r y p o s i t i o n s the c a l c u l a t i o n is s p r e a d i n g on a space torus like f o l d i n g

\*\ param [ in ] ca ca a c e l l u l a r a u t o m a t a area

\*/

void S e t I n f e c t e d R a t e ( I n h a b i t a n t [ ,] ca )

{

int [] c o o r d s = e x p e r t . G e t B o r d e r s ( X , Y , this . r a d i u s ) ;

int s t r a t e g y = ( c o o r d s [2] > c o o r d s [3] ? 1 : 0) + ( c o o r d s [0] > c o o r d s [1] ? 2 : 0) ;

i n f e c t e d R a t e = 0; s w i t c h ( s t r a t e g y )

{

case (0) :

i n f e c t e d R a t e = S e t I n f e c t e d I n s i d e A r e a ( c o o r d s [0] , c o o r d s [1] , c o o r d s [2] , co o r d s [3] , ca ) ;

break ; case (1) :

i n f e c t e d R a t e = S e t I n f e c t e d I n s i d e A r e a ( c o o r d s [0] , c o o r d s [1] , 0 , c o o r d s [3] , ca ) +

S e t I n f e c t e d I n s i d e A r e a ( c o o r d s [0] , c o o r d s [1] , c o o r d s [2] , e x p e r t . Width - 1 , ca ) ;

break ; case (2) :

i n f e c t e d R a t e = S e t I n f e c t e d I n s i d e A r e a (0 , c o o r d s [1] , c o o r d s [2] , c o o r d s [3] , ca ) +

S e t I n f e c t e d I n s i d e A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , c o o r d s [2] , c o o r d s [3] , ca ) ;

break ;

case (3) :

i n f e c t e d R a t e = S e t I n f e c t e d I n s i d e A r e a (0 , c o o r d s [1] , 0 , c o o r d s [3] , ca ) +

S e t I n f e c t e d I n s i d e A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , 0 , c o o r d s [3] , ca ) +

S e t I n f e c t e d I n s i d e A r e a (0 , c o o r d s [1] , c o o r d s [2] , e x p e r t . Width - 1 , ca ) +

S e t I n f e c t e d I n s i d e A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , c o o r d s [2] , e x p e r t . Width - 1 , ca ) ;

break ;

}

i n f e c t e d R a t e -= (( ca [ X , Y ] != null ) && ( ca [ X , Y ]. I n f e c t e d ) ) ?

1 : 0;

i n f e c t e d R a t e /= ( s q u a r e - 1) ;

}

/\* \*

* + R e s o l v e s conflict , when two agent try to take the same cell .
  + An agent p r i o r i t y is d e f i n e d by s t a t u s of a g e nt s ( n o n i n f e c t e d one has the h i g h e s t p r i o r i t y ) ,
  + in the case of s i m i l a r s t a t u s e s agent with the best h e a l t h ( h e a l t h S t a t u s value ) has the h i g h e s t p r i o r i t y
  + \ param [ in ] x1 , y1 c o o r d i m a t e s of a cell
  + \ param [ in ] caFrom , caTo d e p a r t u r e and d e s t i n a t i o n aread of a c e l l u l a r a u t o m a t a

\*/

void R e s o l v e C o n f l i c t ( int x1 , int y1 , ref I n h a b i t a n t [ ,] caFrom , ref I n h a b i t a n t [ ,] caTo )

{

if ( caTo [ x1 , y1 ]. I n f e c t e d != I n f e c t e d )

{

n m b O f C o n f l i c t s I N I ++;

if ( caTo [ x1 , y1 ]. I n f e c t e d )

{

}

else

{

}

else

caTo [ x1 , y1 ]. F i n d A n o t h e r P l a c e ( ref caTo , ref caTo ) ; caTo [ x1 , y1 ] = this ;

c a F r o m [ X , Y ] = null ; C h a n g e I n h a b i t a n t s E n v i r o m e n t ( -1) ; oldX = X = x1 ;

oldY = Y = y1 ; C h a n g e I n h a b i t a n t s E n v i r o m e n t (1) ;

F i n d A n o t h e r P l a c e ( ref caFrom , ref caTo ) ;

if ( caTo [ x1 , y1 ]. I n f e c t e d ) n m b O f C o n f l i c t s I I ++;

else

n m b O f C o n f l i c t s I N I N ++;

if ( r e c o v e r y R a t e > caTo [ x1 , y1 ]. r e c o v e r y R a t e )

{

}

else

}

}

caTo [ x1 , y1 ]. F i n d A n o t h e r P l a c e ( ref caTo , ref caTo ) ; caTo [ x1 , y1 ] = this ;

c a F r o m [ X , Y ] = null ; C h a n g e I n h a b i t a n t s E n v i r o m e n t ( -1) ; oldX = X = x1 ;

oldY = Y = y1 ; C h a n g e I n h a b i t a n t s E n v i r o m e n t (1) ;

F i n d A n o t h e r P l a c e ( ref caFrom , ref caTo ) ;

/\* \*

* + Looks for a n o t h e r p o s i t i o n in a r a d i u s of m o v e m e n t s
  + If a c u r r e n t n e i g h b o r h o o d is o v e r c r o w d e d ( more that 2\* r a d i u s a t t e m p t s to take a cell ) , a r a d i u s is i n c r e a s e d .
  + \ param [ in ] caFrom , caTo d e p a r t u r e and d e s t i n a t i o n aread of a c e l l u l a r a u t o m a t a

\*/

void F i n d A n o t h e r P l a c e ( ref I n h a b i t a n t [ ,] caFrom , ref I n h a b i t a n t [ ,] caTo )

{

bool n o t R e m o v e d = true ; int c u r R a d i u s = r a d i u s ; int a t t e m p t = 0;

do

{

n o t R e m o v e d = D i s l o c a t e ( cur Radius , ref caFrom , ref caTo ) ; if ( a t t e m p t > 2 \* c u r R a d i u s )

{

a t t e m p t = 0; c u r R a d i u s += r a d i u s ;

if (( c u r R a d i u s > caTo . G e t L e n g t h (0) / 2) || ( c u r R a d i u s

> caTo . G e t L e n g t h (1) / 2) )

c u r R a d i u s = Math . Min ( caTo . G e t L e n g t h (0) , caTo .

G e t L e n g t h (1) ) ;

}

a t t e m p t ++;

} while ( n o t R e m o v e d ) ;

}

/\* \*

* + Try to shift to r a n d o m p o s i t i o n in the n e i g h b o r h o o d , d e f i n e d by a given r a d i u s
  + \ param [ in ] c u r R a d i u s r a di u s of the n e i g h b o r h o o d , where m o v e m e n t s are p o s s i b l e
  + \ param [ in ] caFrom , caTo d e p a r t u r e and d e s t i n a t i o n aread of a c e l l u l a r a u t o m a t a
  + \ param [ out ] bool w h e t h e r the d i s l o c a t i o n was s u c c e s s f u l

\*/

bool D i s l o c a t e ( int cur Radius , ref I n h a b i t a n t [ ,] caFrom , ref I n h a b i t a n t [ ,] caTo )

{

int [] c o o r d s = e x p e r t . G e t N e w C o o r d i n a t e s ( oldX , oldY , c u r R a d i u s )

;

if (( caTo [ c o o r d s [0] , c o o r d s [1]] == null ) && (! e x p e r t .

O c c u p i e d B y V i r u s ( c o o r d s [0] , c o o r d s [1]) ) )

{

caTo [ c o o r d s [0] , c o o r d s [1]] = this ; c a F r o m [ X , Y ] = null ; C h a n g e I n h a b i t a n t s E n v i r o m e n t ( -1) ; oldX = X = c o o r d s [0];

oldY = Y = c o o r d s [1]; C h a n g e I n h a b i t a n t s E n v i r o m e n t (1) ; if ( I n f e c t e d )

n m b T o t a l R e l o c a t i o n I n f e c t e d ++;

else

}

else

{

n m b T o t a l R e l o c a t i o n N o n I n f e c t e d ++; r e t u r n false ;

if ( I n f e c t e d )

n m b T o t a l R e l o c a t i o n I n f e c t e d ++;

else

n m b T o t a l R e l o c a t i o n N o n I n f e c t e d ++;

r e t u r n true ;

}

}

/\* \*

* + M o v i n g the agent to a r a nd o m p o s i t i o n i n s i d e the n e i g h b o r h o o d . If some c o n f l i c t s happen , a n o t h e r f u n c t i o n s for r e s o l v i n g of c o l l i s i o n s will be c a l le d
  + \ param [ in ] caFrom , caTo d e p a r t u r e and d e s t i n a t i o n aread of a c e l l u l a r a u t o m a t a
  + \ param [ out ] bool w h e t h e r the i n h a b i t a n t b e c a m e s u s c e p t i b l e

\* \*/

p u b l i c bool Move ( ref I n h a b i t a n t [ ,] caFrom , ref I n h a b i t a n t [ ,] caTo

)

{

bool r e c o v e r e d = false ; if ( I n f e c t e d )

{

d o u b l e r n d R e c o v e r y = e x p e r t . mt . g e n r a n d \_ R e a l I n I n t e r v a l (0 , 1) ;

if ( r n d R e c o v e r y < r e c o v e r y R a t e )

{

}

else

}

I n f e c t e d = false ; r e c o v e r e d = true ;

Q u a r a n t i n e ++;

int [] c o o r d s = e x p e r t . G e t N e w C o o r d i n a t e s ( oldX , oldY , r a d i us ) ; if ( caTo [ c o o r d s [0] , c o o r d s [1]] != null )

R e s o l v e C o n f l i c t ( c o o r d s [0] , c o o r d s [1] , ref caFrom , ref caTo ) ;

else if ( e x p e r t . O c c u p i e d B y V i r u s ( c o o r d s [0] , c o o r d s [1]) ) this . F i n d A n o t h e r P l a c e ( ref caFrom , ref caTo ) ;

else

{

}

caTo [ c o o r d s [0] , c o o r d s [1]] = this ; c a F r o m [ X , Y ] = null ; C h a n g e I n h a b i t a n t s E n v i r o m e n t ( -1) ; oldX = X = c o o r d s [0];

oldY = Y = c o o r d s [1]; C h a n g e I n h a b i t a n t s E n v i r o m e n t (1) ;

r e t u r n r e c o v e r e d ;

}

/\* \*

* + M o v i n g the agent to the best p o s i t i o n i n s i d e the n e i g h b o r h o o d . If some c o n f l i c t s happen , a n o t h e r f u n c t i o n s for r e s o l v i n g of c o l l i s i o n s will be c a l le d
  + \ param [ in ] caFrom , caTo d e p a r t u r e and d e s t i n a t i o n aread of a c e l l u l a r a u t o m a t a
  + \ param [ out ] bool w h e t h e r the i n h a b i t a n t b e c a m e s u s c e p t i b l e

\* \*/

p u b l i c bool C o g n i t i v e M o v e ( ref I n h a b i t a n t [ ,] caFrom , ref I n h a b i t a n t [ ,] caTo )

{

bool r e c o v e r e d = false ; if ( I n f e c t e d )

{

d o u b l e r n d R e c o v e r y = e x p e r t . mt . g e n r a n d \_ R e a l I n I n t e r v a l (0 , 1) ;

if ( r n d R e c o v e r y < r e c o v e r y R a t e )

{

}

else

}

I n f e c t e d = false ; r e c o v e r e d = true ;

Q u a r a n t i n e ++;

int [] c o o r d s = G e t B e s t C e l l I n N e i b o r h o o o d ( oldX , oldY , radius , caTo ) ;

if ( caTo [ c o o r d s [0] , c o o r d s [1]] != null )

R e s o l v e C o n f l i c t ( c o o r d s [0] , c o o r d s [1] , ref caFrom , ref caTo

) ;

else

{

}

caTo [ c o o r d s [0] , c o o r d s [1]] = this ; c a F r o m [ X , Y ] = null ; C h a n g e I n h a b i t a n t s E n v i r o m e n t ( -1) ; oldX = X = c o o r d s [0];

oldY = Y = c o o r d s [1]; C h a n g e I n h a b i t a n t s E n v i r o m e n t (1) ;

r e t u r n r e c o v e r e d ;

}

/\* \*

* + R e t u r n s c o o r d i n a t e s of the most f a v o u r a b l e for l i v i n g cell in a n e i g b o r h o o d
  + \ param [ in ] x , y c e n t e r of the n e i g b o r h o o d
  + \ param [ in ] r a d i us r a d i u s of the n e i g b o r h o o d
  + \ param [ in ] ca area where cells are d i s p o s e d
  + \ param [ out ] r e t C o o r d s c o o r d i n a t e s of a new cell

\*/

p u b l i c int [] G e t B e s t C e l l I n N e i b o r h o o o d ( int x , int y , int radius , I n h a b i t a n t [ ,] ca )

{

int [] c o o r d s = e x p e r t . G e t B o r d e r s ( X , Y , r a d i u s ) ; int [] r e t C o o r d s = new int [2] { 0 , 0 };

int s t r a t e g y = ( c o o r d s [2] > c o o r d s [3] ? 1 : 0) + ( c o o r d s [0] > c o o r d s [1] ? 2 : 0) ;

i n f e c t e d R a t e = 0;

float [] out1 = new float [3] { 0 , 0 , 0 };

float [] out2 = new float [3] { 0 , 0 , 0 };

s w i t c h ( s t r a t e g y )

{

case (0) :

out1 = G e t B e s t C e l l ( c o o r d s [0] , c o or d s [1] , c o o r d s [2] , c o o r d s [3] , wV , wI , ca ) ;

r e t C o o r d s [0] = ( int ) out1 [1]; r e t C o o r d s [1] = ( int ) out1 [2]; break ;

case (1) :

out1 = G e t B e s t C e l l ( c o o r d s [0] , c o or d s [1] , 0 , c o o r d s [3] , wV , wI , ca ) ;

out2 = G e t B e s t C e l l ( c o o r d s [0] , c o or d s [1] , c o o r d s [2] , e x p e r t . Width - 1 , wV , wI , ca ) ;

if ( out1 [0] > out2 [0])

{

}

else

{

}

r e t C o o r d s [0] = ( int ) out1 [1];

r e t C o o r d s [1] = ( int ) out1 [2];

r e t C o o r d s [0] = ( int ) out2 [1];

r e t C o o r d s [1] = ( int ) out2 [2];

break ; case (2) :

out1 = G e t B e s t C e l l (0 , c o o r d s [1] , c o o r d s [2] , c o o r d s [3] , wV , wI , ca ) ;

out2 = G e t B e s t C e l l ( c o o r d s [0] , e x pe r t . H e i g h t - 1 , c o o r d s [2] , c o o r d s [3] , wV , wI , ca ) ;

if ( out1 [0] > out2 [0])

{

}

else

{

}

r e t C o o r d s [0] = ( int ) out1 [1];

r e t C o o r d s [1] = ( int ) out1 [2];

r e t C o o r d s [0] = ( int ) out2 [1];

r e t C o o r d s [1] = ( int ) out2 [2];

break ; case (3) :

float [] out3 = new float [3] { 0 , 0 , 0 };

out1 = G e t B e s t C e l l (0 , c o o r d s [1] , 0 , c o o r d s [3] , wV , wI , ca ) ;

out2 = G e t B e s t C e l l ( c o o r d s [0] , e x pe r t . H e i g h t - 1 , 0 , c o o r d s [3] , wV , wI , ca ) ;

if ( out1 [0] > out2 [0])

{

}

else

{

}

out3 [0] = ( int ) out1 [0];

out3 [1] = ( int ) out1 [1];

out3 [2] = ( int ) out1 [2];

out3 [0] = ( int ) out2 [0];

out3 [1] = ( int ) out2 [1];

out3 [2] = ( int ) out2 [2];

out1 = G e t B e s t C e l l (0 , c o o r d s [1] , c o o r d s [2] , e x p e r t .

Width - 1 , wV , wI , ca ) ;

out2 = G e t B e s t C e l l ( c o o r d s [0] , e x pe r t . H e i g h t - 1 , c o o r d s [2] , e x p e r t . Width - 1 , wV , wI , ca ) ;

if ( out1 [0] > out2 [0])

{

if ( out1 [0] > out3 [0])

{

}

else

{

}

else

{

}

r e t C o o r d s [0] = ( int ) out1 [1]; r e t C o o r d s [1] = ( int ) out1 [2];

r e t C o o r d s [0] = ( int ) out3 [1]; r e t C o o r d s [1] = ( int ) out3 [2];

if ( out2 [0] > out3 [0])

{

}

else

{

}

}

r e t C o o r d s [0] = ( int ) out2 [1]; r e t C o o r d s [1] = ( int ) out2 [2];

r e t C o o r d s [0] = ( int ) out3 [1]; r e t C o o r d s [1] = ( int ) out3 [2];

break ;

}

r e t u r n r e t C o o r d s ;

}

/\* \*

* R e t u r n s c o o r d i n a t e s of the most f a v o u r a b l e for l i v i n g cell in a r e c t u n g l e area
* \ param [ in ] x0 , x1 , y0 , y1 c o o r d i n a t e s of the r e c t a n g u l a r area
* \ param [ in ] weightV , w e i g h t I - c o o f i c i e n t s of the o b j e c t i v e f u n c t i o n ( for e s t i m a t i n g cells )
* \ param [ in ] ca area of a c e l l u l a r a u t o m a t a
* \ param [ out ] array an 1 x3 - dim array that c o n t a i n s an a s s e s s m e n t of the best p o s i t i o n and its c o o r d i n a t e s

\* \*/

float [] G e t B e s t C e l l ( int x0 , int x1 , int y0 , int y1 , float weightV , float weightI , I n h a b i t a n t [ ,] ca )

{

float b e s t V a l = Int16 . MinValue , val = Int16 . M i n V a l u e ; int x Best = -1 , y Best = -1;

for ( int i = x0 ; i <= x1 ; i ++)

for ( int j = y0 ; j <= y1 ; j ++)

{

val = - w e i g h t I \* e x p e r t . G e t I n h a b i t a n t D e n s i t y ( i , j ) - w e i g h t V \* e x p e r t . G e t V i r u s e s D e n s i t y ( i , j ) ;

if (( val > b e s t V a l ) && (! ex p e r t . O c c u p i e d B y V i r u s ( i , j ) )

)

{

x Best = i ; y Best = j ; b e s t V a l = val ;

}

}

r e t u r n new float [3] { bestVal , xBest , y Best };

}

/\* \*

* C h a n g e the e n v i r o m e n t i n f o r m a t i o n about d e n s i t y in a r e c t a n g u l a r area
* The c o r r e s p o n d i n g d u r a t i o n of the d i s e a s e is a s s i g n e d .
* \ param [ in ] x0 , x1 , y0 , y 1 c o o r d i n a t e s of the r e c t a n g u l a r area
* \ param [ in ] sign 1/ -1 for d e p a r t u r e / d e s t i n a t i o n p o i n t s a c c o r d i n g l y

\*/

void I n h a b i t a n t s I n s i d e A r e a ( int x0 , int x1 , int y0 , int y1 , short sign )

{

for ( int i = x0 ; i <= x1 ; i ++) for ( int j = y0 ; j < y1 ; j ++)

e x p e r t . C h a n g e I n h a b i t a n t D e n s i t y ( i , j , ( float ) sign / s q u a r e ) ;

}

/\* \*

* + C h a n g e the e n v i r o m e n t i n f o r m a t i o n about d e n s i t y in the n e i g b o r h o o d of d e p a r t u r e and d e s t i n a t i o n poing
  + In the case of b o u n d a r y p o s i t i o n s the d i s e a s e is s p r e a d i n g on a space torus like f o l d i n g
  + \ param [ in ] sign 1/ -1 for d e p a r t u r e / d e s t i n a t i o n p o i n t s a c c o r d i n g l y

\* \*/

p u b l i c void C h a n g e I n h a b i t a n t s E n v i r o m e n t ( short sign )

{

int [] c o o r d s = e x p e r t . G e t B o r d e r s ( X , Y , r a d i u s ) ;

int s t r a t e g y = ( c o o r d s [2] > c o o r d s [3] ? 1 : 0) + ( c o o r d s [0] > c o o r d s [1] ? 2 : 0) ;

s w i t c h ( s t r a t e g y )

{

case (0) :

I n h a b i t a n t s I n s i d e A r e a ( c o o r d s [0] , c o o r d s [1] , c o o r d s [2] , c o o r d s [3] , sign ) ;

break ; case (1) :

I n h a b i t a n t s I n s i d e A r e a ( c o o r d s [0] , c o o r d s [1] , 0 , c o o r d s [3] , sign ) ;

I n h a b i t a n t s I n s i d e A r e a ( c o o r d s [0] , c o o r d s [1] , c o o r d s [2] , e x p e r t . Width - 1 , sign ) ;

break ; case (2) :

I n h a b i t a n t s I n s i d e A r e a (0 , c o o r d s [1] , c o o r d s [2] , c o o r d s [3] , sign ) ;

I n h a b i t a n t s I n s i d e A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , c o o r d s [2] , c o o r d s [3] , sign ) ;

break ; case (3) :

I n h a b i t a n t s I n s i d e A r e a (0 , c o o r d s [1] , 0 , c o o r d s [3] , sign

) ;

I n h a b i t a n t s I n s i d e A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , 0 , c o o r d s [3] , sign ) ;

I n h a b i t a n t s I n s i d e A r e a (0 , c o o r d s [1] , c o o r d s [2] , e x p e r t .

Width - 1 - 1 , sign ) ;

I n h a b i t a n t s I n s i d e A r e a ( c o o r d s [0] , e x p e r t . H e i g h t - 1 , c o o r d s [2] , e x p e r t . Width - 1 , sign ) ;

break ;

}

}

}

}