Extended Sugarscape model in XL

German: „Realisierung eines erweiterten Sugarscape-Modells in XL"

Author: **Daniel Klöck**

Supervisor: **Winfried Kurth**

This document has been written as "Studienarbeit" for:

Brandenburg University of Technology at Cottbus
Department of Computer Science
Chair for Practical Computer Science / Graphics Systems

German:

Brandenburgische Technische Universität Cottbus
Lehrstuhl Graphische Systeme

Index

Summary

1. **Introduction:** The introduction will give a brief overview of what the sugarscape simulation system is, what it has been developed for, which are his fundamental elements, how it has been extended and which technology has been used to implement the project.

2. **Rule system documentation:** This chapter defines all rules implemented in this particular sugarscape system.

3. **Class documentation:** The class documentation describes all classes implemented for the sugarscape system.

4. **Simulating social behaviour:** In this section example simulations with following rules have been run and interpreted:
    a. Trade and pollution.
    b. Trade and combat.
    c. Territorial behaviour and cultural transmission.
    d. Territorial behaviour and combat.
    e. Territorial behaviour, trade and combat.

5. **Review:** A description of which problems have been encountered during the project and how they were solved, as well as an evaluation of the project is given.

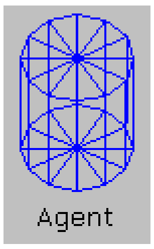
Introduction

Usually, when social processes are studied, they are isolated into separate subprocesses (economic, demographic, cultural…) because of the complexity of the social system [1].

The sugarscape model has been developed as a tool able to analyse social processes without isolating them. It applies agent-based computer modelling techniques to the study of human social phenomena, including interaction with an environment, combat, trade, migration, group formation, transmission of culture, propagation of disease, and population dynamics. It is an attempt to simulate fundamental social structures and group behaviour from the interaction of various agents operating in artificial environments under very simple rules [1].

In general, the sugarscape involves three basic elements: agents, an environment or space, and rules.

Agents

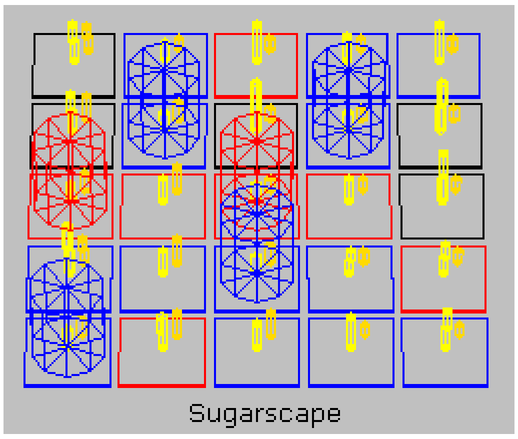


Agents are the "people" of the sugarscape. Each agent has internal states and behaviour rules. Some states are fixed for life, while others may change through the interaction with other agents or the environment. For example, an agent's vision and sex is fixed for life while its cultural identity and wealth may change as an agent moves around and interacts.
At the picture on the right an agent is represented. The colour of an agent shows the tribe to which it belongs.

Environment

Agents live in a spatial environment. They can move in two dimensions within a number of cells. Each of those cells contains certain amounts of sugar and spice which are collected by the agents and used as food, currency and for trade.

In the picture below, you can see Agents (big red and blue cylinders depending on the tribe they belong to), sugar (small yellow cylinders), spice (small orange cylinders) and regions (shown as rectangles; black if it does not belong to any tribe, otherwise the tribe's colour).



Rules

Finally, there are rules of behaviour for the agents and sites of the environment. For example, a rule for disease transmission may be:

- Select a neighbouring agent
- If the agent is not immune to the disease, transmit the disease to that agent.
- Repeat for all neighbouring agents.

Extension to the original sugarscape

This model has been implemented as an extended version of the model proposed by [1]. As extension of their proposed sugarscape, territorial behaviour has been implemented.

The territorial behaviour extension allows tribes to own cells (the number of cells a tribe can own depends on its wealth). If a cell is owned by a tribe only agents from the same tribe can collect from those cells, but if for some reason their wealth is not enough to maintain all the cells they got, another tribe may take a part of their territory.

You could think of this part of the simulation as a tribe of humans that defend their territory to gather its resources, the tribe may only be able to defend a territory the size of which is not bigger as the tribe has power to defend (in the sugarscape wealth is seen as power). If for example a tribe loses one of its members and its power shrinks, another tribe may be able to take part of its territory, but as long as it has enough power, the other tribes will respect its borders.

The XL language

First of all, the sugarscape implementation should be written in an object oriented language, because in such a language, agents and environmental sites are naturally implemented as objects, and as those, agents are easily simulated as individuals that just interact which their environment [1]. XL is an extension to the programming language Java [3], and therefore object oriented.

XL has also a rule-oriented part: Configurations of objects can be transformed according to graph grammar rules.

In addition XL gives the ability to link an object to a visualization form, and shows it on screen without the need of extra code, which raises the simplicity and clarity of the implementation [2].

Rule system documentation

The behaviour of the members of the sugarscape is based on a rule system. Following rules are supported by the simulation system.

Sugar and spice cell rules

Sugarscape growback (G_α): At each sugar and spice cell, sugar and spice grow back at a rate of α units per turn up to the capacity at each position (by default $\alpha = 1$).

Pollution diffusion (D_α): Each α turns and at each site an average pollution level over all von Neumann neighbouring sites is calculated[1] and becomes their new pollution (by default $\alpha = 5$).

Agent rules

Agent movement rule (M), modified for pollution, combat and territorial behaviour:

1. Look as far as vision permits in each of the four lattice directions: north, south, east and west.
2. Throw out all sites occupied by members of the same tribe.
3. Throw out all sites occupied by members of other tribes who are wealthier.
4. Throw out all sites which belong to other tribes, in case that the tribe does not have more regions than it can defend[2].
5. The reward of each remaining site is given by the sugar to pollution ratio of each site plus the occupant's wealth, if it is occupied.
6. If the best reward appears at multiple sites, select the nearest one (the lattice directions looking order is randomized).
7. Move to this site.
8. Collect reward and gain this cell for own tribe if the tribe did not already surpass its limit (maximal number of cells a clan can own = wealth of the tribe * sugarcells / total wealth).

[1] An average pollution for an agent is calculated as the sum of all pollutions of its Von Neumann neighbours and the agent's own pollution divided through the number of Von Neumann neighbours +1.
[2] The regions a tribe is able to defend is calculated as the total tribe's wealth, multiplied with the total number of regions, and divided through the sum of all tribes' wealth.

Pollution formation ($P_{\alpha\beta}$): When sugar or spice quantity s is gathered from the sugarscape, an amount of production pollution is generated in quantity α*s. When sugar or spice amount m is consumed (metabolized), consumption pollution is generated according to β*m. The total pollution on a site at a time t, p^t, is the sum of the pollution present at the previous time, plus the pollution resulting from production and consumption activities, that is, $p^t = p^{t-1} + \alpha s + \beta m$ (by default α=0.1 and β=0.2).

Agent Mating (S):

1. Select a neighbouring agent at random.
2. If the neighbouring agent is of the opposite sex and if both agents are fertile[3] and at least one of the agents has an empty neighbouring site then a newborn is produced by crossing over the parents' genetic and cultural characteristics (children have genetics of mendelian rules).
3. Repeat for all neighbours.

Agent Culture (K): Each agent is born with a tag string which stands for its culture. Agents are assigned to tribes depending on the number of 0s and 1s on that tag string.
Agent cultural transmission:

1. Select a neighbouring agent at random.
2. Select a tag randomly.
3. If the neighbour agrees on that tag position, no change is made, otherwise, the neighbour's tag is flipped to agree with the agent's tag.
4. Repeat for all neighbouring agents.

Agent Trade (T):

1. Agent and neighbour compute their MRSs[4], if these are equal then end, else continue.
2. The direction of exchange is as follows: spice flows from the agent with the higher MRS to the agent with the lower MRS while sugar goes in the opposite direction.
3. The geometric mean of the two MRSs is calculated (this will serve as the bargaining price, p).
4. The quantities to be exchanged are as follows: if p > 1 then p units of spice for 1 unit of sugar; if p < 1 then 1/p units of sugar for 1 unit of spice.
5. if the trade will (a) make both agents better off (increases the welfare of both agents), and (b) not cause the agents' MRSs to

[3] An agent is called fertile if it is in childbearing age and has at least the amount of sugar and spice it had at birth.
[4] MRS: Marginal Rate of Substitution. MRS is calculated with following formula:

$$\frac{spiceLevel}{spiceMetabolism} \Big/ \frac{sugarLevel}{sugarMetabolism}$$

cross over one another, then the trade is made and return to start, else end.

Agent Inheritance (I): If an agent dies its wealth is equally divided among all its living children.

Agent Credit (L_{dr}):

- An agent is a potential lender if it is too old to have children, in which case the maximum amount it may lend is one-half of its current wealth.
- An agent is a potential lender if it is of childbearing age and has wealth in excess of the amount necessary to have children, in which case the maximum amount it may lend is the excess of wealth.
- An agent is a potential borrower if it is of childbearing age and has insufficient wealth to have a child and has income[5] in the present period making it credit-worthy for a loan written at terms specified by the lender.
- If a potential borrower and a potential lender are neighbours then a loan is originated with duration of d turns at the rate of r percent, and the face value of the loan is transferred from the lender to the borrower.
- At the time of the loan due date, if the borrower has sufficient wealth to repay the loan then a transfer from the borrower to the lender is made, else the borrower is required to pay back half of its wealth and a new loan is originated for the remaining sum.
- If the borrower on an active loan dies before the due date then the lender simply takes the loss.

(By default d=10 and r=10)

Agent disease processes (E): Each agent is born with a string of 0s and 1s which is its immune system.
Agent immune response:

- If a disease is a substring of the immune system then end (the agent is immune), else (the agent is infected) go to the next step.
- The substring in the agent's immune system having the smallest Hamming distance from the disease is selected and the first bit at which it is different from the disease string is changed to match the disease.

Disease transmission:

- For each neighbour, a disease that currently afflicts the agent is selected at random and given to the neighbour.

[5] Income = resources gathered, minus metabolism, minus other loan obligations.

Class documentation

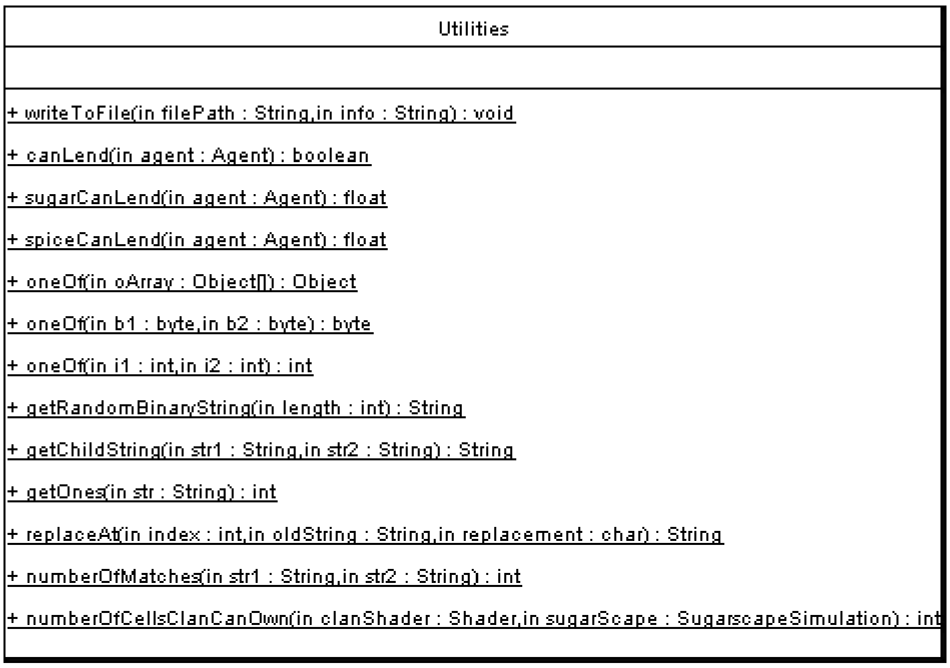
Following classes have been implemented for the sugarscape:

- Utilities (located at “Utilities.xl”)
- Region (located at “Cells.xl”)
- SugarscapeSimulation (located at XL “Sugarscape.xl”)
- Agent (located at “Agents.xl”)
- SugarCell (located at “Cells.xl”)
- SpiceCell (located at “Cells.xl”)
- Credit (located at “Agents.xl”)
- Disease (located at “Agents.xl”)

A brief word on these will be taken.

Utilities

This class only has static methods, and is used as a helping tool to all the other classes.

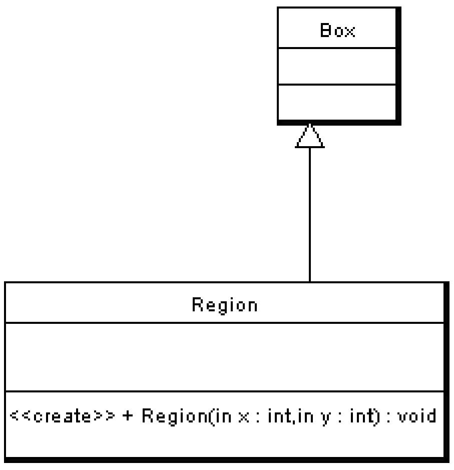


```
Utilities

+ writeToFile(in filePath : String,in info : String) : void
+ canLend(in agent : Agent) : boolean
+ sugarCanLend(in agent : Agent) : float
+ spiceCanLend(in agent : Agent) : float
+ oneOf(in oArray : Object[]) : Object
+ oneOf(in b1 : byte,in b2 : byte) : byte
+ oneOf(in i1 : int,in i2 : int) : int
+ getRandomBinaryString(in length : int) : String
+ getChildString(in str1 : String,in str2 : String) : String
+ getOnes(in str : String) : int
+ replaceAt(in index : int,in oldString : String,in replacement : char) : String
+ numberOfMatches(in str1 : String,in str2 : String) : int
+ numberOfCellsClanCanOwn(in clanShader : Shader,in sugarScape : SugarscapeSimulation) : int
```

Region

The region class extends Box to be visualized as one.
This class is used to show and store information about the tribal membership of a cell[6].

[6] A cell is a position in the sugarscape which contains spice and sugar resources.

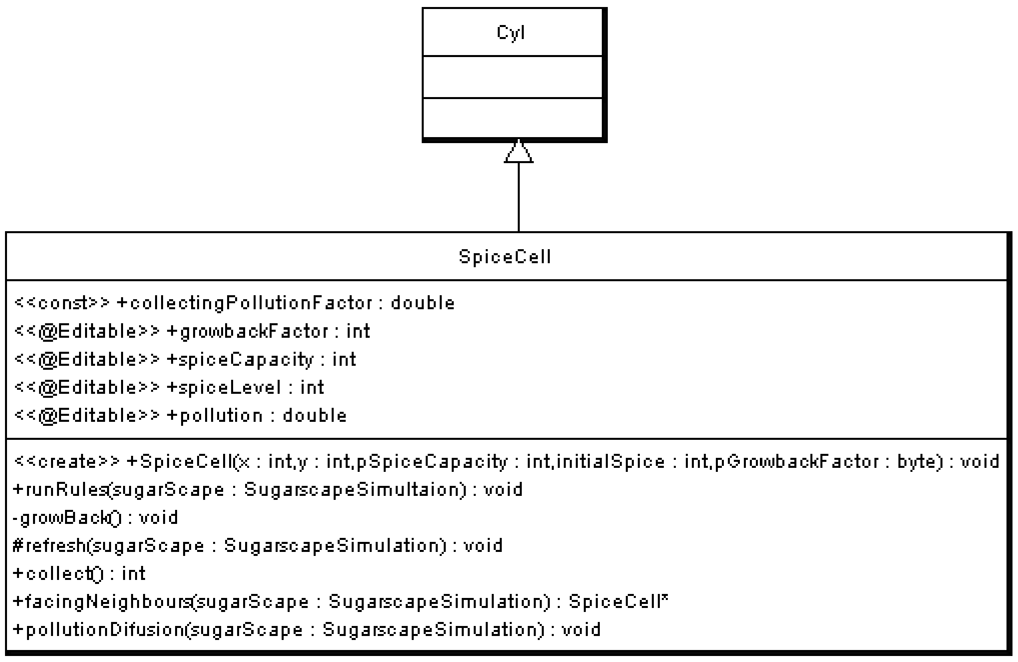


SpiceCell

The SpiceCell class extends Cyl to be shown as a cylinder.
This class is used to show spice on the sugarscape, as well as storing some information about it, such as:

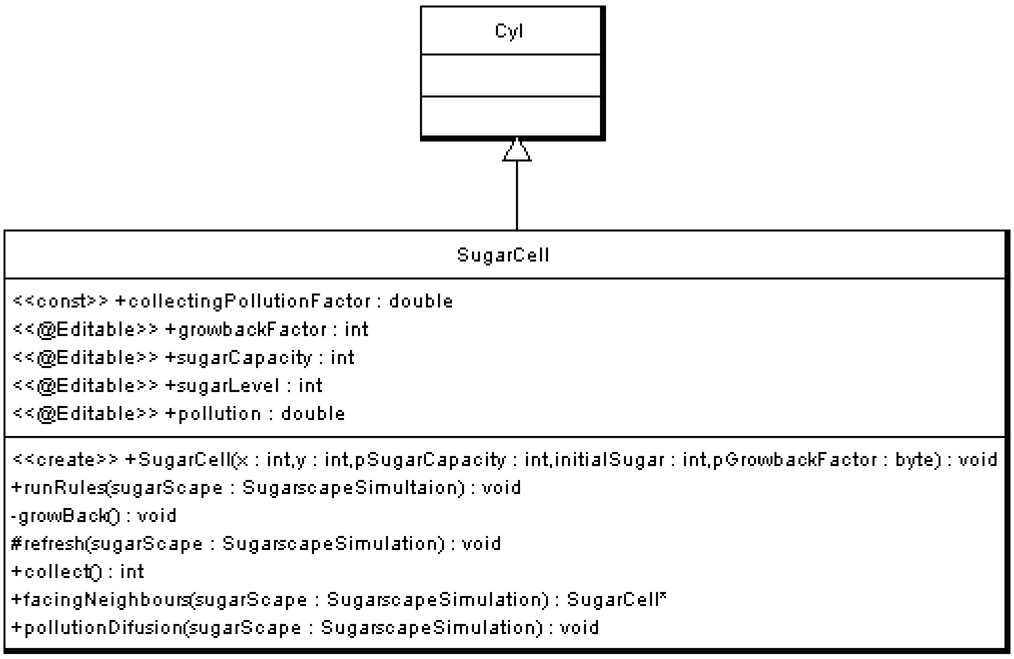
- Quantity of spice available.
- Capacity of spice for this cell.
- Pollution on this cell.
- Growback factor.
- The collecting pollution factor.

It also is used to run the spice's rules, as well as for taking actions like collecting.


Cyl
SpiceCell
<<const>> +collectingPollutionFactor : double
<<@Editable>> +growbackFactor : int
<<@Editable>> +spiceCapacity : int
<<@Editable>> +spiceLevel : int
<<@Editable>> +pollution : double
<<create>> +SpiceCell(x : int,y : int,pSpiceCapacity : int,initialSpice : int,pGrowbackFactor : byte) : void
+runRules(sugarScape : SugarscapeSimultaion) : void
-growBack() : void
#refresh(sugarScape : SugarscapeSimulation) : void
+collect() : int
+facingNeighbours(sugarScape : SugarscapeSimulation) : SpiceCell*
+pollutionDifusion(sugarScape : SugarscapeSimulation) : void

SugarCell

This class has the same structure and behaviour as the SpiceCell class. The only reason to separate this class from SpiceCell, and not creating a Cell class which contains spice and sugar instead, is that with this structure it is easier to separate the sugar's from the spice's behaviour.

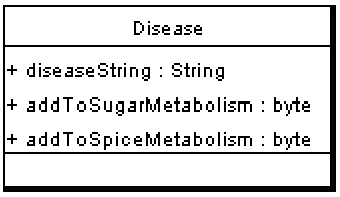

Cyl
SugarCell
<<const>> +collectingPollutionFactor : double
<<@Editable>> +growbackFactor : int
<<@Editable>> +sugarCapacity : int
<<@Editable>> +sugarLevel : int
<<@Editable>> +pollution : double
<<create>> +SugarCell(x : int,y : int,pSugarCapacity : int,initialSugar : int,pGrowbackFactor : byte) : void
+runRules(sugarScape : SugarscapeSimultaion) : void
-growBack() : void
#refresh(sugarScape : SugarscapeSimulation) : void
+collect() : int
+facingNeighbours(sugarScape : SugarscapeSimulation) : SugarCell*
+pollutionDifusion(sugarScape : SugarscapeSimulation) : void

Disease

This class is used to save needed disease information, which is:

- The disease string, which must be found in the immune system of an agent to be immune to it.
- The quantity which the disease will add to the agent's sugar and spice metabolism

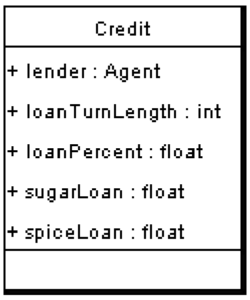
Extending this class would make it easy to crate new diseases which, for example, lower the agent's vision.



Credit

This class is used to store information about credits which agents take from each other. The information stored is:

- The lender of the credit.
- When the loan is supposed to be returned.
- By which percentage the loan will increase before it is returned.
- Amount of sugar loaned.
- Amount of spice loaned.



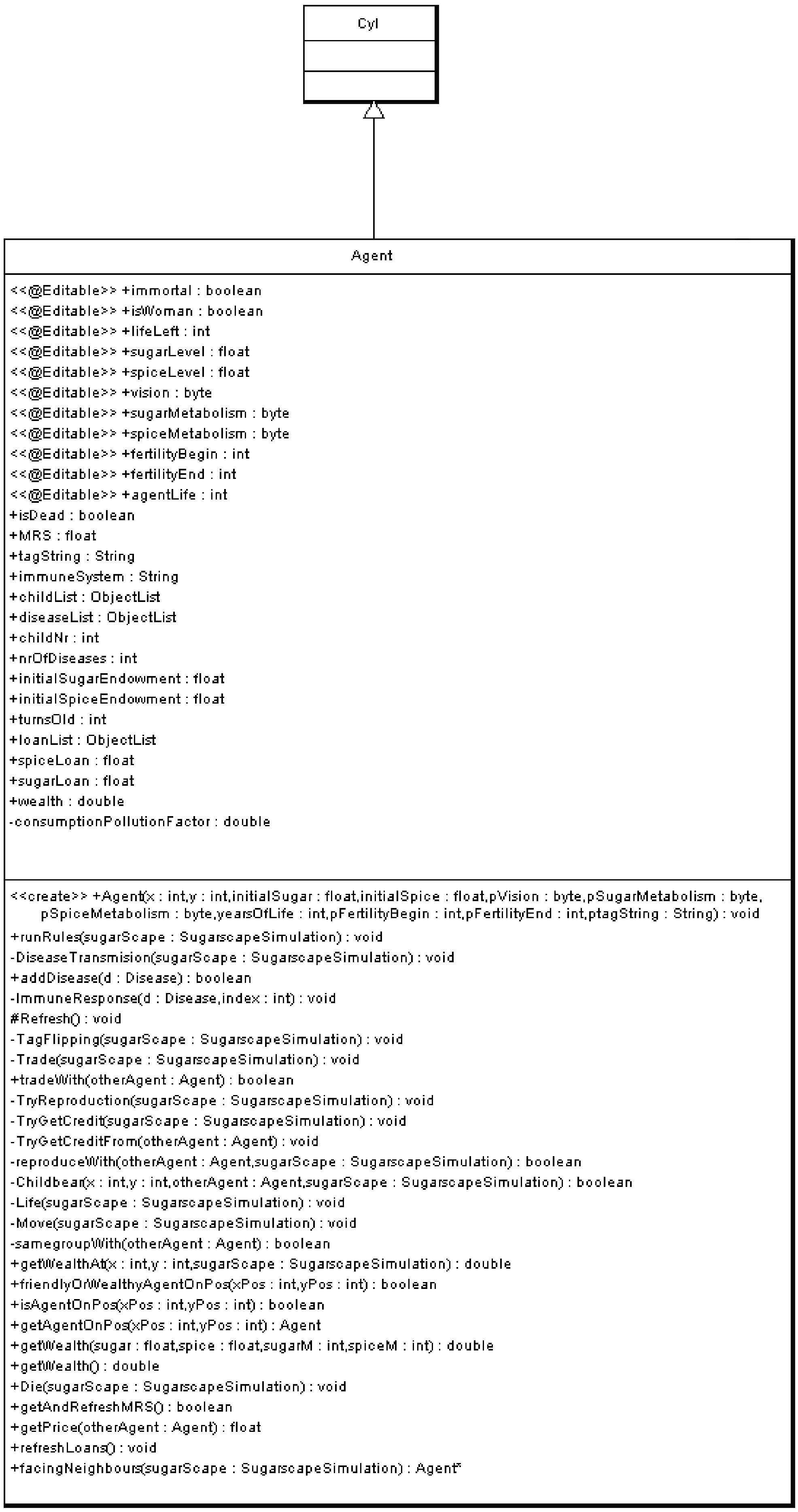
Agent

The Agent class is used to store data about the agents, like:

- Age.
- Sugar and spice metabolism.
- Wealth
- Childs
- Credits
- Diseases
- Internal value of sugar and spice for trade

This class also is used to run the agent’s rules, like for example:

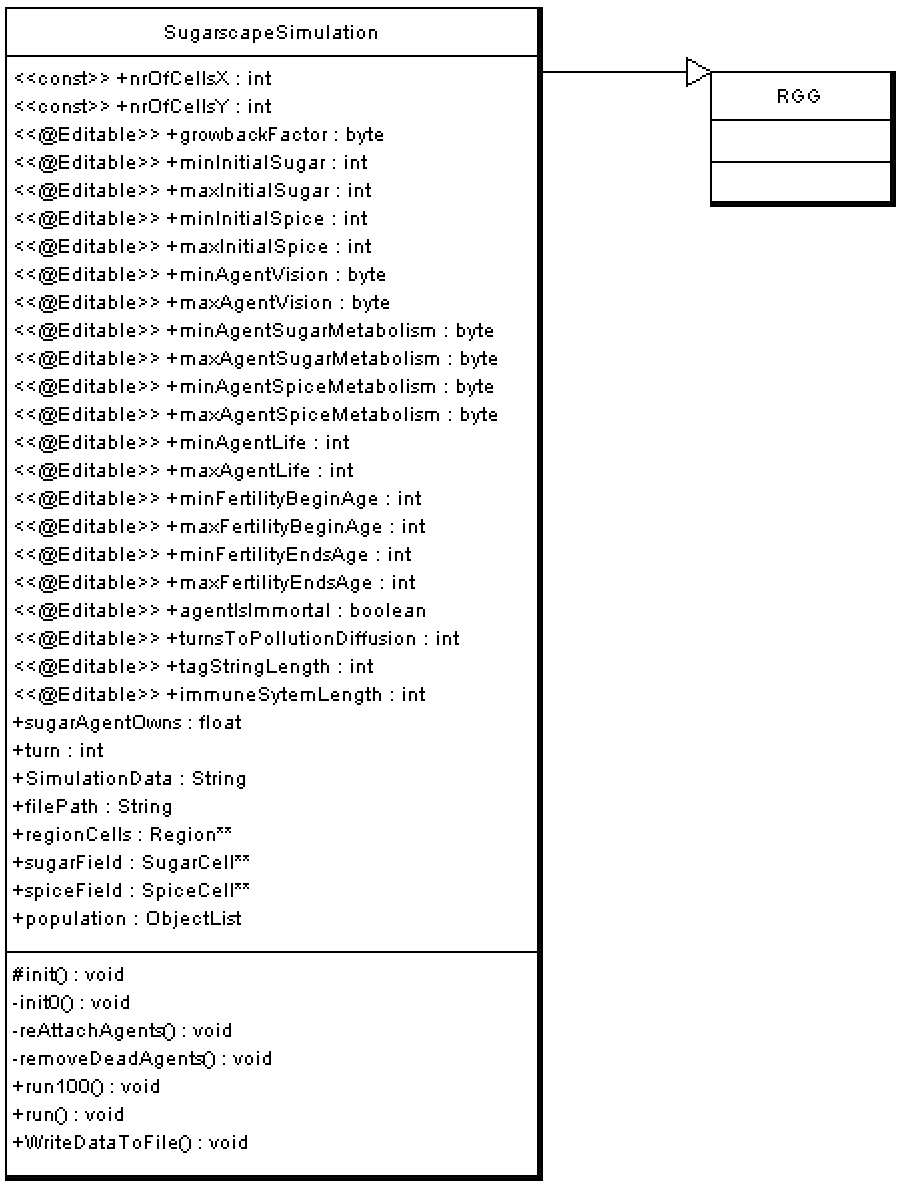
- Trade
- Disease transmission
- Movement
- Reproduction
- Combat



```
Cyl

Agent

<<@Editable>> +immortal : boolean
<<@Editable>> +isWoman : boolean
<<@Editable>> +lifeLeft : int
<<@Editable>> +sugarLevel : float
<<@Editable>> +spiceLevel : float
<<@Editable>> +vision : byte
<<@Editable>> +sugarMetabolism : byte
<<@Editable>> +spiceMetabolism : byte
<<@Editable>> +fertilityBegin : int
<<@Editable>> +fertilityEnd : int
<<@Editable>> +agentLife : int
+isDead : boolean
+MRS : float
+tagString : String
+immuneSystem : String
+childList : ObjectList
+diseaseList : ObjectList
+childNr : int
+nrOfDiseases : int
+initialSugarEndowment : float
+initialSpiceEndowment : float
+turnsOld : int
+loanList : ObjectList
+spiceLoan : float
+sugarLoan : float
+wealth : double
-consumptionPollutionFactor : double

<<create>> +Agent(x : int,y : int,initialSugar : float,initialSpice : float,pVision : byte,pSugarMetabolism : byte,
      pSpiceMetabolism : byte,yearsOfLife : int,pFertilityBegin : int,pFertilityEnd : int,ptagString : String) : void
+runRules(sugarScape : SugarscapeSimulation) : void
-DiseaseTransmision(sugarScape : SugarscapeSimulation) : void
+addDisease(d : Disease) : boolean
-ImmuneResponse(d : Disease,index : int) : void
#Refresh() : void
-TagFlipping(sugarScape : SugarscapeSimulation) : void
-Trade(sugarScape : SugarscapeSimulation) : void
+tradeWith(otherAgent : Agent) : boolean
-TryReproduction(sugarScape : SugarscapeSimulation) : void
-TryGetCredit(sugarScape : SugarscapeSimulation) : void
-TryGetCreditFrom(otherAgent : Agent) : void
-reproduceWith(otherAgent : Agent,sugarScape : SugarscapeSimulation) : boolean
-Childbear(x : int,y : int,otherAgent : Agent,sugarScape : SugarscapeSimulation) : boolean
-Life(sugarScape : SugarscapeSimulation) : void
-Move(sugarScape : SugarscapeSimulation) : void
-samegroupWith(otherAgent : Agent) : boolean
+getWealthAt(x : int,y : int,sugarScape : SugarscapeSimulation) : double
+friendlyOrWealthyAgentOnPos(xPos : int,yPos : int) : boolean
+isAgentOnPos(xPos : int,yPos : int) : boolean
+getAgentOnPos(xPos : int,yPos : int) : Agent
+getWealth(sugar : float,spice : float,sugarM : int,spiceM : int) : double
+getWealth() : double
+Die(sugarScape : SugarscapeSimulation) : void
+getAndRefreshMRS() : boolean
+getPrice(otherAgent : Agent) : float
+refreshLoans() : void
+facingNeighbours(sugarScape : SugarscapeSimulation) : Agent*
```

SugarscapeSimulation

The SugarscapeSimulation class is the main class of the simulation and is the place where instances of all the classes are created, to interact with each other.
The <<const>> and <<@Editable>> variables are the main configuration of the sugarscape and must be changed before initializing the sugarscape simulation in order to take effect.


SugarscapeSimulation
<<const>> +nrOfCellsX : int
<<const>> +nrOfCellsY : int
<<@Editable>> +growbackFactor : byte
<<@Editable>> +minInitialSugar : int
<<@Editable>> +maxInitialSugar : int
<<@Editable>> +minInitialSpice : int
<<@Editable>> +maxInitialSpice : int
<<@Editable>> +minAgentVision : byte
<<@Editable>> +maxAgentVision : byte
<<@Editable>> +minAgentSugarMetabolism : byte
<<@Editable>> +maxAgentSugarMetabolism : byte
<<@Editable>> +minAgentSpiceMetabolism : byte
<<@Editable>> +maxAgentSpiceMetabolism : byte
<<@Editable>> +minAgentLife : int
<<@Editable>> +maxAgentLife : int
<<@Editable>> +minFertilityBeginAge : int
<<@Editable>> +maxFertilityBeginAge : int
<<@Editable>> +minFertilityEndsAge : int
<<@Editable>> +maxFertilityEndsAge : int
<<@Editable>> +agentIsImmortal : boolean
<<@Editable>> +turnsToPollutionDiffusion : int
<<@Editable>> +tagStringLength : int
<<@Editable>> +immuneSytemLength : int
+sugarAgentOwns : float
+turn : int
+SimulationData : String
+filePath : String
+regionCells : Region**
+sugarField : SugarCell**
+spiceField : SpiceCell**
+population : ObjectList
#init() : void
-init0() : void
-reAttachAgents() : void
-removeDeadAgents() : void
+run100() : void
+run() : void
+WriteDataToFile() : void
RGG

This class also holds important variables for data retrieving and output which are:

- SimulationData, which should be used to store data retrieved in the simulation.

- filePath, which is a String containing the path of the file where SimulationData will be stored when using the writeDataToFile method.

The simulation will be initialized after saving the file, which is when the init() method will be called. It will then run a turn using the run() method (use run100() to run 100 turns).

Simulating social behaviour

Introduction

The sugarscape has been developed to simulate social behaviour.
At this time, following simulations have been run and interpreted:

- Trade and pollution.
- Trade and combat.
- Territorial behaviour and cultural transmission.
- Territorial behaviour and combat.
- Territorial behaviour, trade and combat.

Since there are too many different values in a single simulation to examine it entirely, we will always run simulations with a purpose (like prove a theory or answer a question) to reduce the amount of data which have to be analysed and interpreted.

Simulation 1: Trade and pollution

With these series of simulations we will try to solve following questions:

1. Does trade influence the quantity of pollution which is generated?
2. Does pollution influence the amount of sugar and spice which is traded?

Simulations 1a: Trade's influence on pollution.

To answer the first question five simulations have been run.
The number of trades that two agents may realize in one turn has been limited in order to control the number of trades which will be made in each simulation.

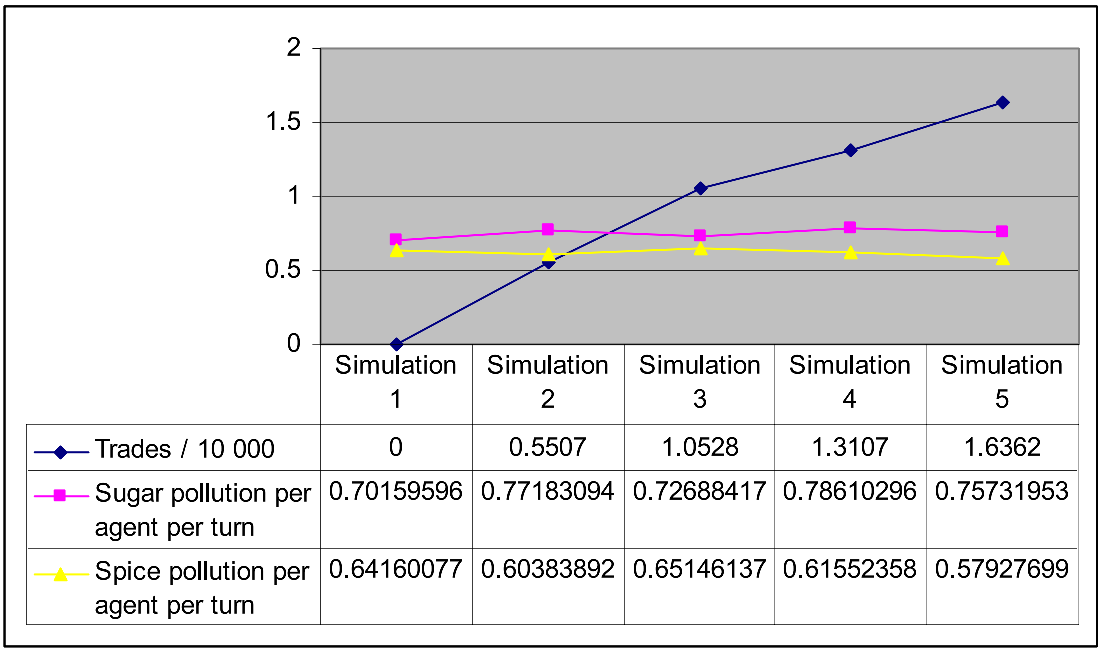
The simulations have been run under following rules:

- For the spice cells: growback and pollution diffusion ($\{G_1, D_5\}$).
- For the sugar cells: growback and pollution diffusion ($\{G_1, D_5\}$).
- For the agents: movement, pollution formation, trade, mating and inheritance ($\{M, P_{0.1, 0.2}, S, I, T\}$).

To gather data about the sugar and spice pollution which an agent creates at average, the sugarscape has been run under Rule System ($\{G_1, D_5\}$, $\{M, P_{0.1, 0.2}, S, I, T\}$).

Figure 1 shows the quantity of spice and sugar pollution an agent produces per turn on average with following trade limitations:

- Simulation 1: No trade is allowed
- Simulation 2: An agent may trade with each neighbour 2 times per turn.
- Simulation 3: An agent may trade with each neighbour 5 times per turn.
- Simulation 4: An agent may trade with each neighbour 10 times per turn.
- Simulation 5: An agent may trade with each neighbour 20 times per turn.



	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Simulation 5
◆ Trades / 10 000	0	0.5507	1.0528	1.3107	1.6362
■ Sugar pollution per agent per turn	0.70159596	0.77183094	0.72688417	0.78610296	0.75731953
▲ Spice pollution per agent per turn	0.64160077	0.60383892	0.65146137	0.61552358	0.57927699

Figure 1: Sugar and spice pollution per agent per turn weigh against the number of trades

On figure 1 we can observe the influence of the trade in the pollution. While the number of trades increases from 0 to 16362, the pollution an agent produces per turn remains between 0.5 and 0.8, which means that trade has no influence in the quantity of pollution generated by agents.

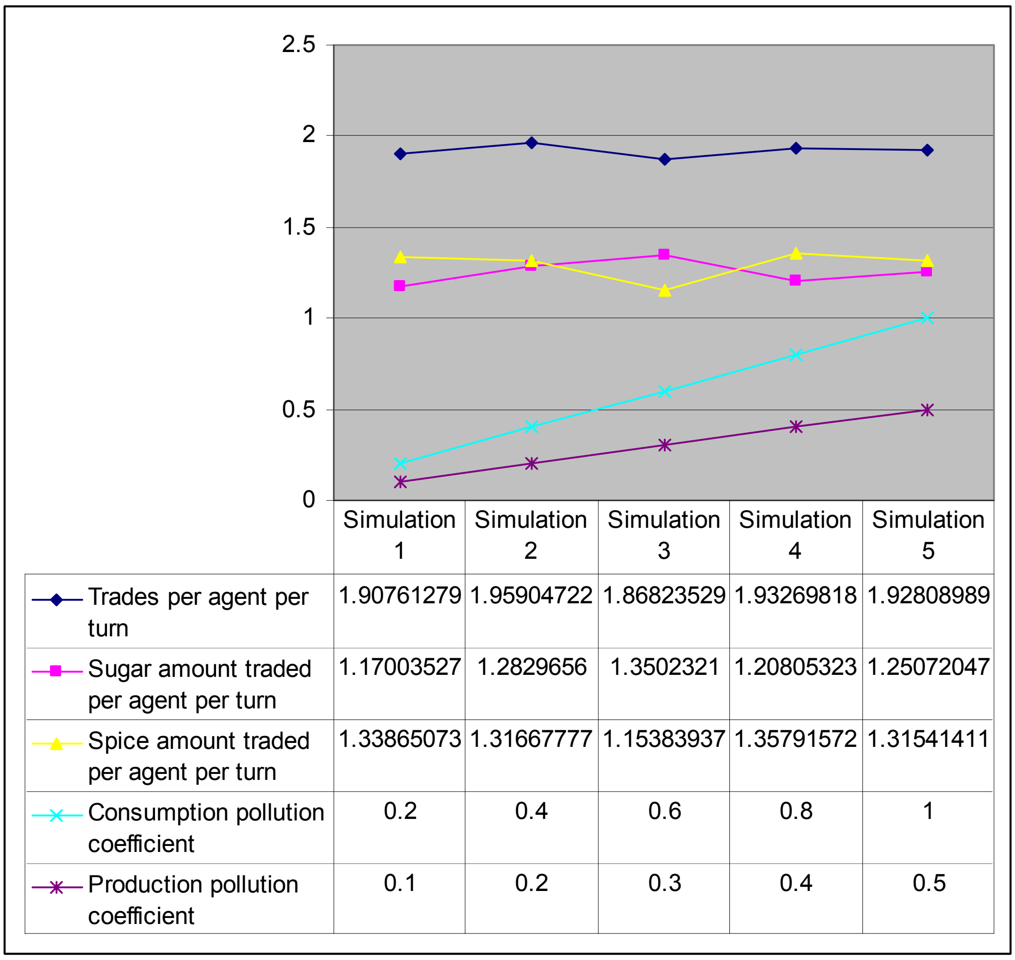
Why did we obtain that result? The answer is very simple: pollution is generated every time spice or sugar grows, and every time an agent consumes sugar or spice. Since none of these actions is influenced by trade, trade can not influence pollution.

Simulations 1b: Pollution's influence on trade.

To answer the question of how much pollution influences trade, five simulations have been run with different consumption and production pollution coefficients.

The simulations have been run under following rules:

- For the spice cells: growback and pollution diffusion ({G_1, D_5}).
- For the sugar cells: growback and pollution diffusion ({G_1, D_5}).
- For the agents: movement, pollution formation, trade, mating and inheritance ({M, $P_{0.1, 0.2}$, S, I, T}).



	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Simulation 5
◆ Trades per agent per turn	1.90761279	1.95904722	1.86823529	1.93269818	1.92808989
■ Sugar amount traded per agent per turn	1.17003527	1.2829656	1.3502321	1.20805323	1.25072047
▲ Spice amount traded per agent per turn	1.33865073	1.31667777	1.15383937	1.35791572	1.31541411
× Consumption pollution coefficient	0.2	0.4	0.6	0.8	1
✳ Production pollution coefficient	0.1	0.2	0.3	0.4	0.5

Figure 2: Trades and trade amount per agent per turn against pollution coefficients

The differences in the configuration of the simulations are the consumption and production coefficients, they have been configured as shown in figure 2. Production pollution goes from 0.1 to 0.5 and consumption pollution from 0.2 to 1, which makes a huge difference. For example, the amount of sugar pollution at the end of the simulation passed from 6953.699806 in simulation 1 to 32357.6734 in simulation 5 (which is about 6 times more). However, the number of trades and the amount of sugar and spice traded per agent per turn kept almost the same. Evidently, this result can only be interpreted as that pollution does not have any influence on trade.

Simulation 2: Trade and combat

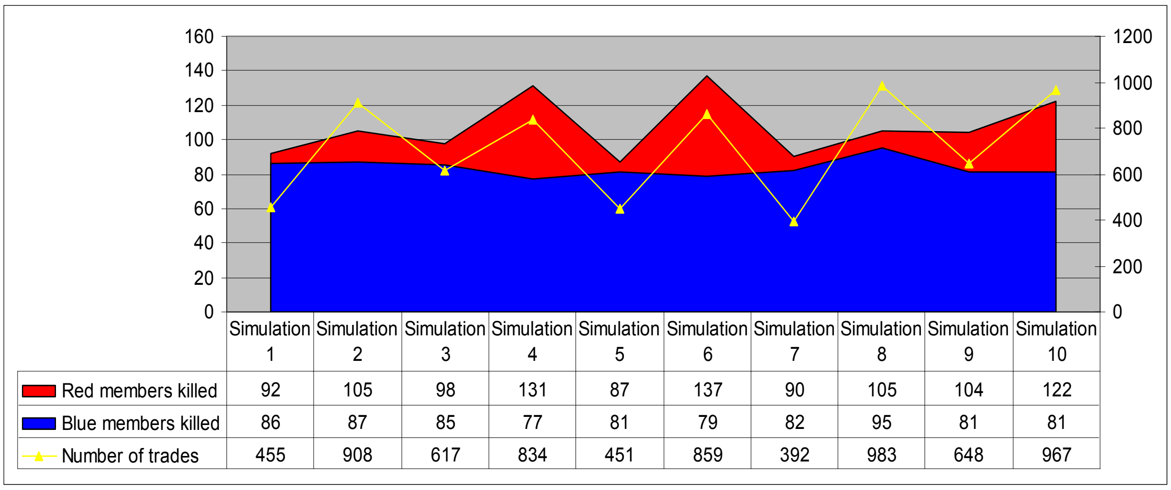
With these series of simulations we will try to answer if trade makes a tribe more powerful in combat.

Simulations 2a: The power of trade

We want to prove that if a tribe trades, their members become wealthier, and therefore more powerful, which makes the whole tribe more powerful in combat.
The simulations have been run under following rules:

- For the spice cells: growback and pollution diffusion ($\{G_1, D_5\}$).
- For the sugar cells: growback and pollution diffusion ($\{G_1, D_5\}$).
- For agents of the red tribe: movement, pollution formation, mating, inheritance and combat ($\{M, P_{0.1, 0.2}, S, I\}$).
- For agents of the blue tribe: movement, pollution formation, mating, inheritance, combat and trade ($\{M, P_{0.1, 0.2}, S, I, T\}$).

As we can see in figure 3, the blue tribe seems to be more powerful in view of the fact that they kill more members of the enemy tribe than they got killed. This means that trade is an advantage in sugarscape's war.



	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Simulation 5	Simulation 6	Simulation 7	Simulation 8	Simulation 9	Simulation 10
🟥 Red members killed	92	105	98	131	87	137	90	105	104	122
🟦 Blue members killed	86	87	85	77	81	79	82	95	81	81
▲ Number of trades	455	908	617	834	451	859	392	983	648	967

Figure 3: Killings (blue tribe trades, red tribe does not trade)

Simulation 3: Territorial behaviour and cultural transmission

A simulation has been run to find out whether territorial behaviour influences the quantity of cultural transmission within two different tribes.

Simulations 3a: Cultural transmission with regions (random)

First, a simulation with territorial behaviour and one without has been run with following rule systems:

- With territorial behaviour: ({G_5, D_5}, {M, $P_{0.1,\ 0.2}$, I, T}), where M is movement without the extension for territorial behaviour and without combat.

- Without territorial behaviour: ({G_5, D_5}, {M, $P_{0.1,\ 0.2}$, I, T}), where M is movement with the extension for territorial behaviour, but without combat.

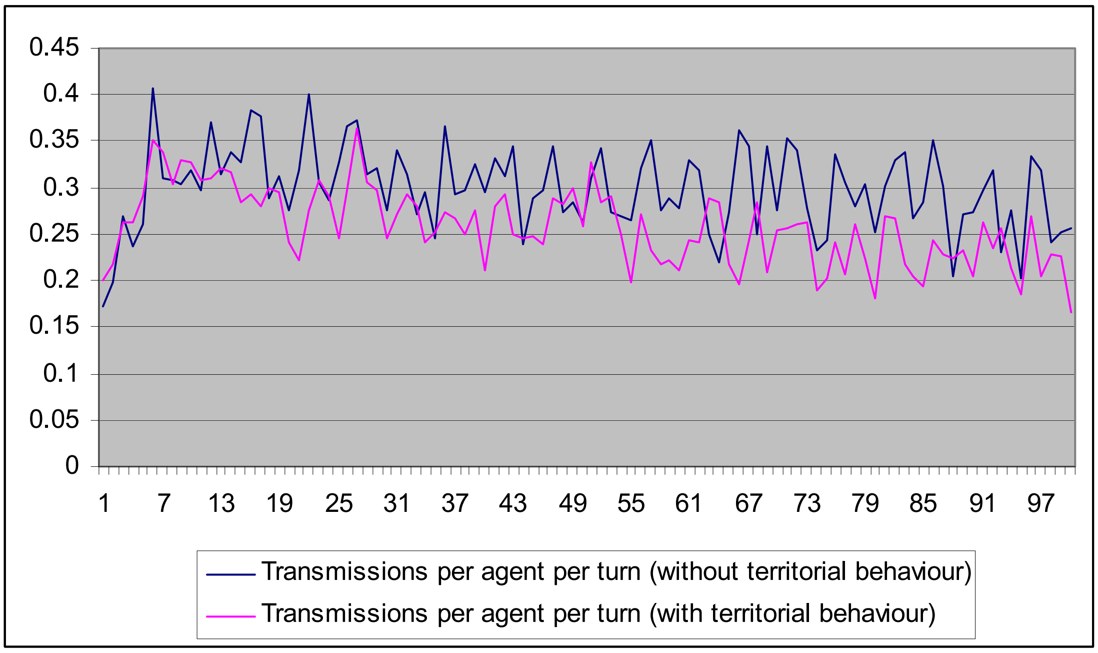


Figure 4: Number of transmissions per agent per turn

From figure 4 we can conclude that the transmissions made with territorial behaviour are slightly lower than without territorial behaviour.

The territories which were formed are shown in figure 5.

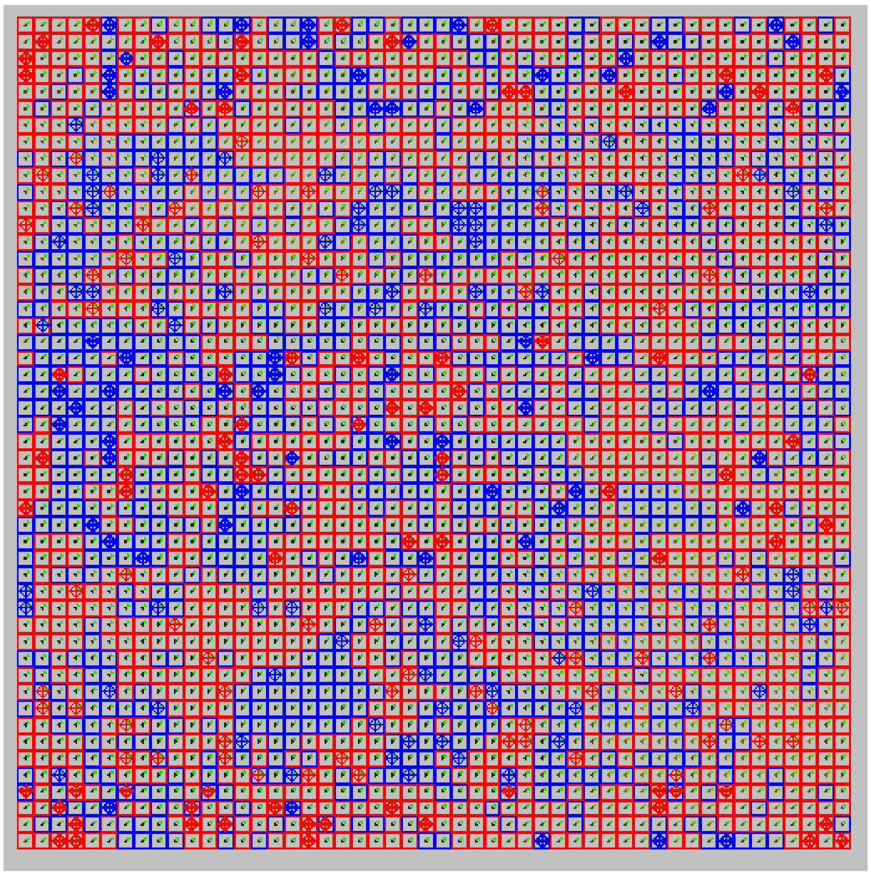


Figure 5: Territories at the end of Simulation 3a with territorial behaviour

In figure 5 we can see sugar and spice (which are drawn as brown and green cylinders, but because of the resolution seen almost as dots in figure 5), agents (drawn as red and blue cylinders[7]), regions (drawn as red and blue squares)[8].

The regions at the end of the simulation 3a with territorial behaviour are very irregular and do not seem to follow any kind of tribal unity. This may have brought the number of transmissions of the simulation with territorial behaviour to be very close to the number of transmissions without it.
The reason for this is that it makes the territories have almost only bordering fields, which enables agents to transmit culture more easily.

That is why another simulation was run to prove that only a slight difference in the number of transmissions will appear because of the territorial behaviour.

[7] Note that the view is set to "from top" and therefore cylinders may appear as circles. The colour shows to which tribe the agents belong.
[8] Regions without owner will be drawn black.

Simulations 3b: Cultural transmission with regions (fixed)

In these simulations, agents' starting positions have been fixed and agents of the same tribe start with the same culture.
The initial blue tribe's cultural string has been set to "0000000000" and the red one's to "1111111111", which makes their culture opposite. This means that a member of a tribe can only get new culture from another tribe's members or cultured members of its own clan.

The purpose of this new arrangement of the sugarscape is to simulate the behaviour of the agents in a more solid tribe.

The rule system has been left the same as in 3a, which is:

- With territorial behaviour: ({G_5, D_5}, {M, $P_{0.1,\ 0.2}$, I, T}), where M is movement without the extension for territorial behaviour and without combat.

- Without territorial behaviour: ({G_5, D_5}, {M, $P_{0.1,\ 0.2}$, I, T}), where M is movement with the extension for territorial behaviour, but without combat.

Figure 6 and figure 7 show the evolution of the simulations.

First we notice that figure 6 now got more uniform regions, and there seems to be only one border line, which goes from the upper left corner to the lower right one.
We also observe that while in figure 6 almost no agent surpasses the border line, in figure 7, agents move without restraint through the entire map.

This means that in figure 6 agents may only transmit culture at the border line, and after being cultured they may transmit their new culture to the members of their own clan.

At figure 8 the statistics for these two simulations are presented as a graph. On this graph we can see that the difference of the number of transmissions still is just slightly higher at the simulation without territory.

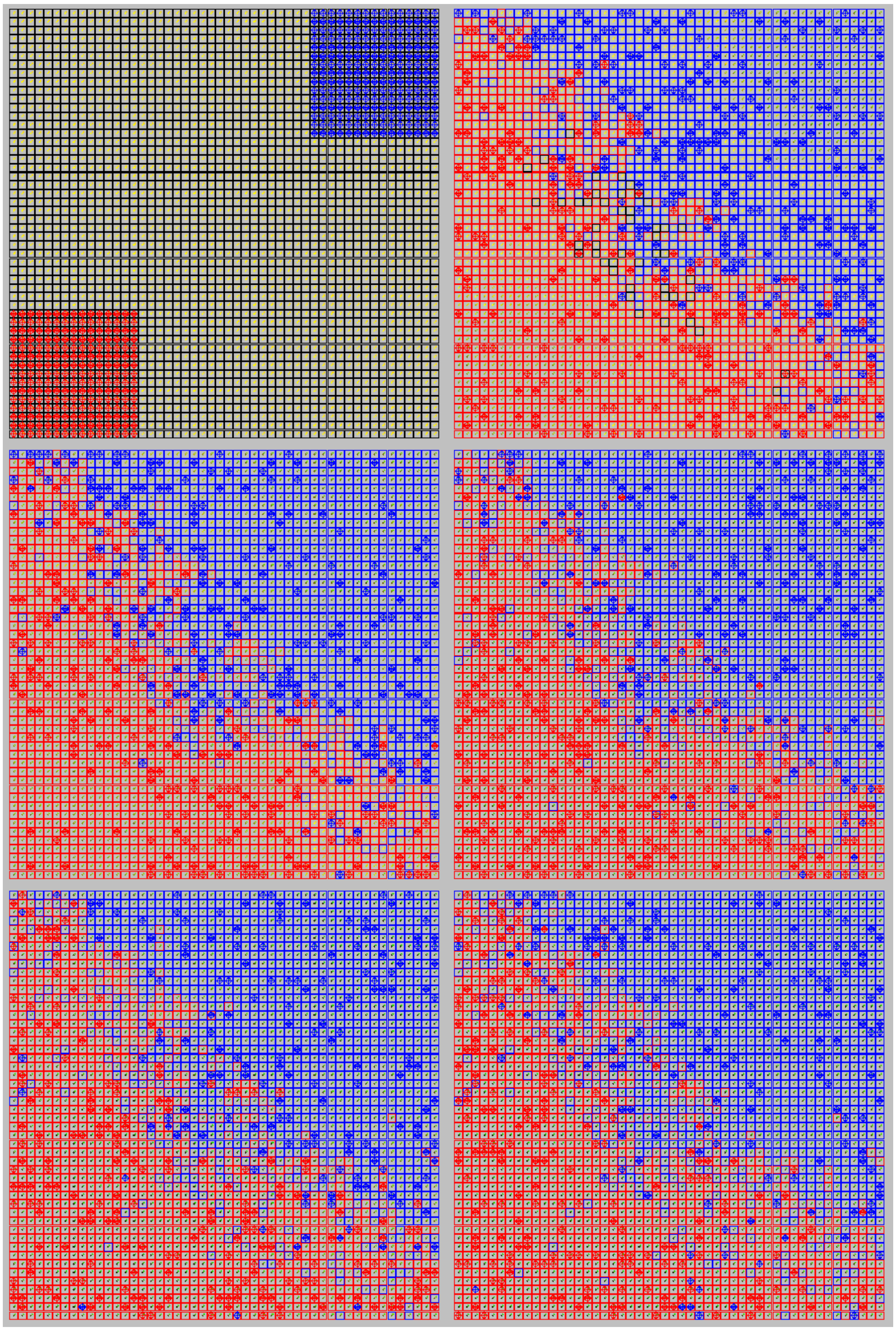


Figure 6: Cultural transmission within tribes with territories (snapshots every 20 turns)

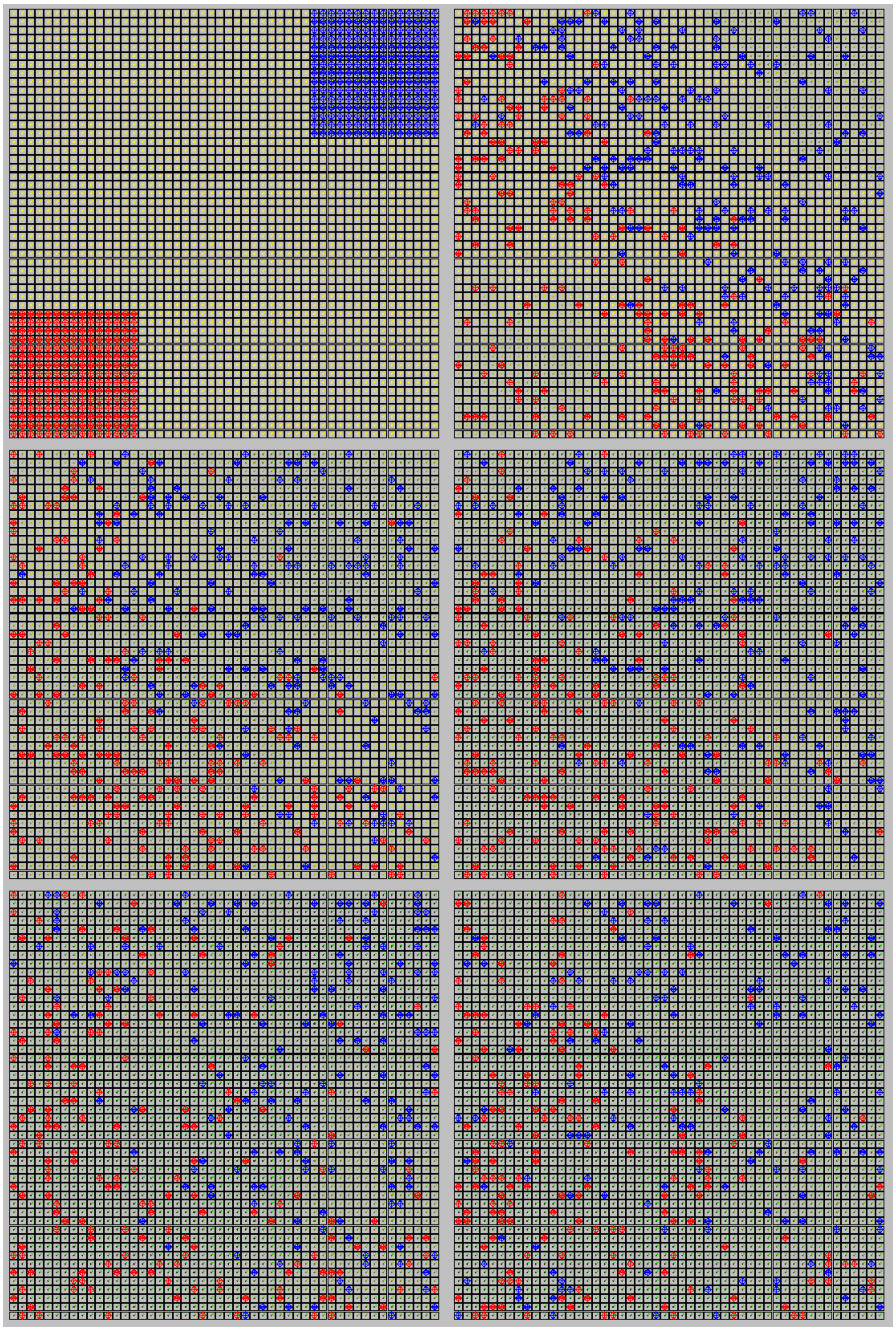


Figure 7: Cultural transmission within tribes without territories (snapshots every 20 turns)

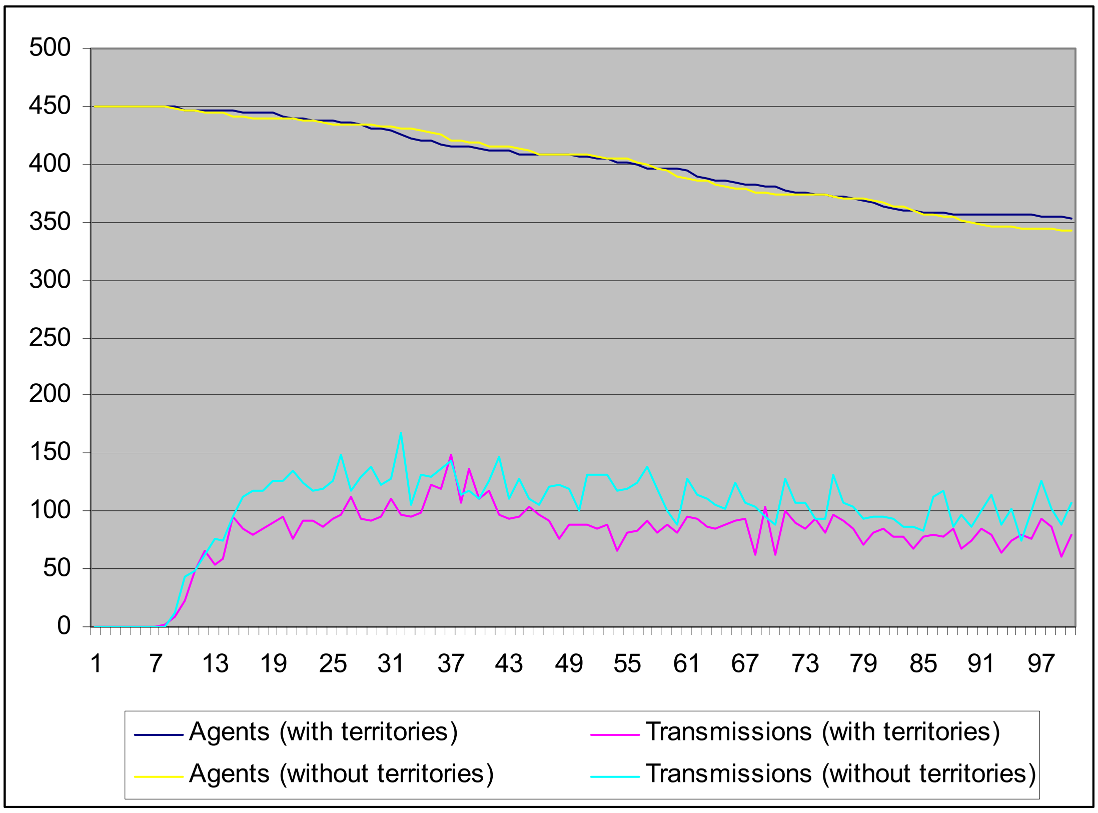


Figure 8: Agents and transmissions for simulations 3b

Simulation 4: Territorial behaviour and combat

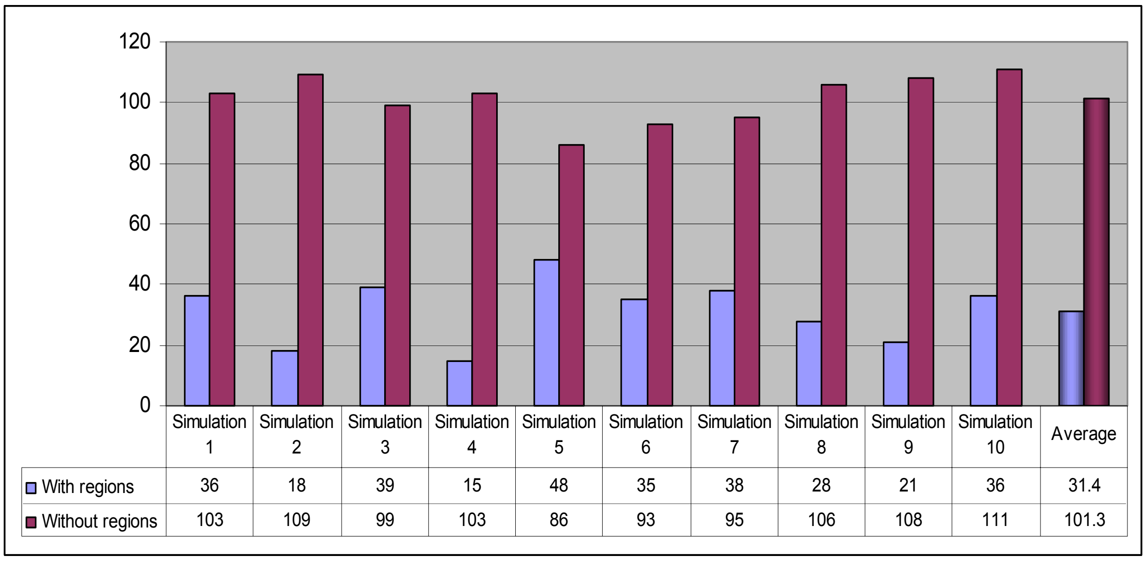
This time we will run simulations to study whether the sugarscape becomes a more peaceful place if territorial behaviour is enabled or not.

We will use fixed agent starting to get more uniform regions.

The simulations have been run under following rules:

- For the spice cells: growback and pollution diffusion ({G_5, D_5}).
- For the sugar cells: growback and pollution diffusion ({G_5, D_5}).
- For the agents: movement, pollution formation and inheritance ({M, $P_{0.1,\ 0.2}$, I}), where M is movement extended for combat and territorial behaviour.

Consequently, the sugarscape has been run under Rule System ({G_5, D_5}, {M, $P_{0.1,\ 0.2}$, I}).



	Simulation 1	Simulation 2	Simulation 3	Simulation 4	Simulation 5	Simulation 6	Simulation 7	Simulation 8	Simulation 9	Simulation 10	Average
■ With regions	36	18	39	15	48	35	38	28	21	36	31.4
■ Without regions	103	109	99	103	86	93	95	106	108	111	101.3

Figure 9: Number of kills with and without regions

As we see in figure 9, there obviously are more kills because of combat without regions than with regions (31.4 with and 101.3 without regions at average).

That is so because with regions tribes only fight if a tribe has a bigger territory than it could possibly defend, which happens only if the relative power of the tribes changes, and even then only a few regions may be invaded.

Simulation 5: Territorial behaviour, trade and combat

In the simulation series 2a we proved that a tribe is more powerful if it trades. Now, we will prove that territorial behaviour makes a tribe more powerful in trade, and therefore more powerful in combat.

In this simulation every tribe will have its own resource, which means that one tribe starts with the sugar field, and the other with the spice field, so that the agents must trade or fight with the other tribe to survive. One of the tribes also will be able to own regions.

Figure 10 shows the outcome of the simulation.

We will analyse the behaviour regarding to the amount of kills, trades, and regions owned.

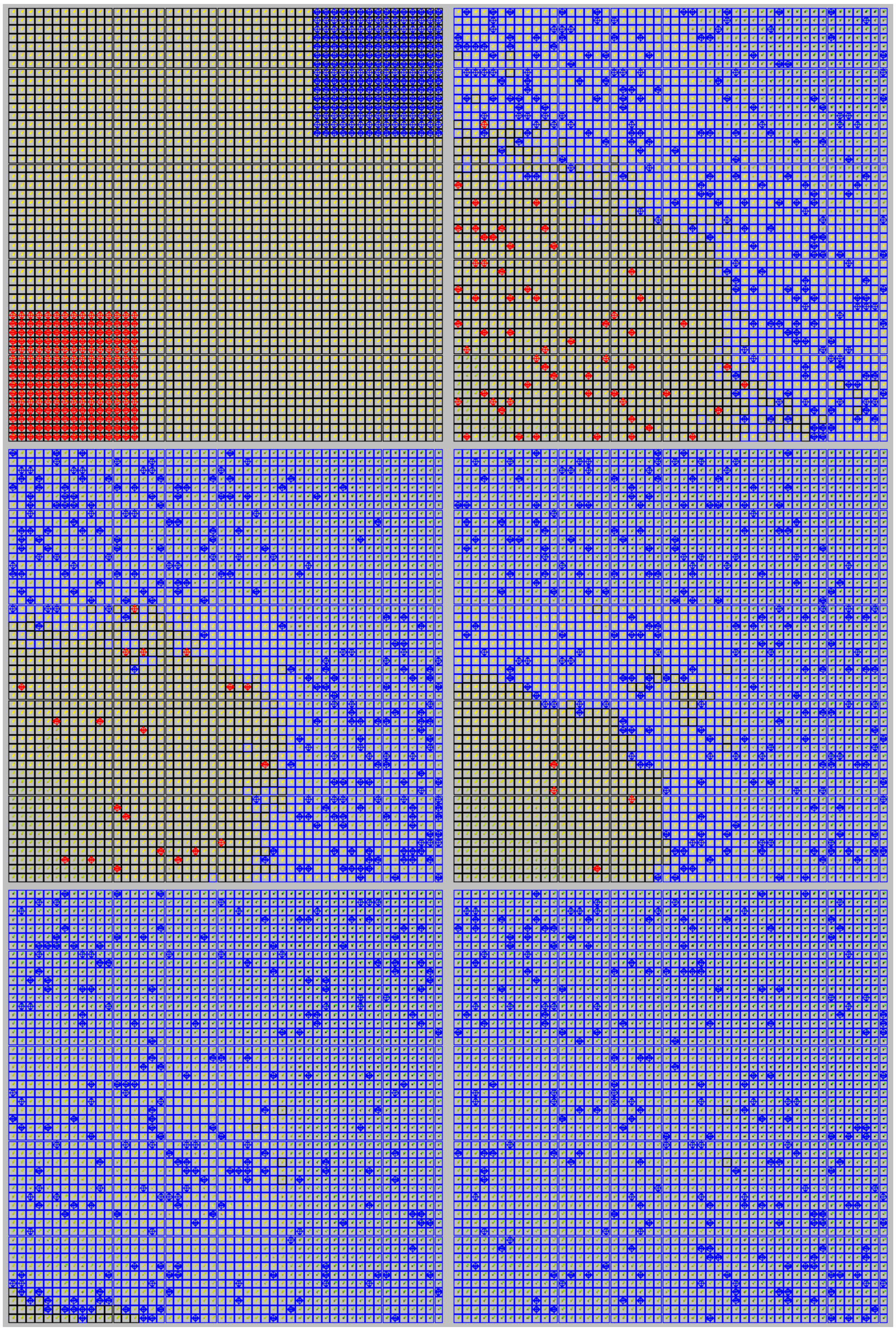


Figure 10: Tribe with regions versus tribe without regions (snapshots every 20 turns)

First of all we will analyse which tribe had more trades.

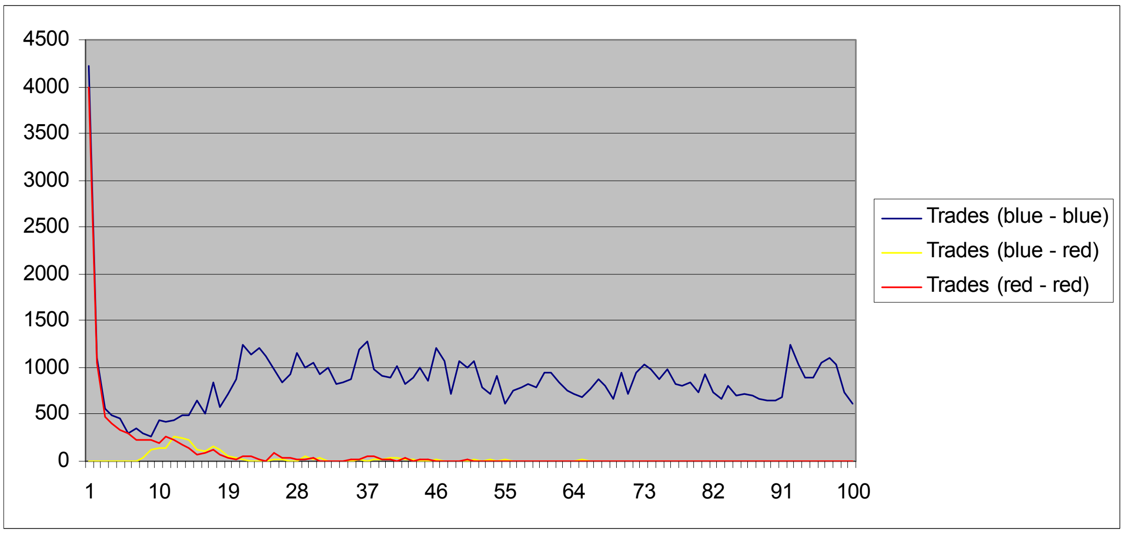


Figure 11: Trades

In figure 11 we can see that at the beginning blue traded only with blue and red only with red. At about turn 10, the tribes began to trade with each other, but then only blue traded with blue.

The explanation of this behaviour is as follows: first every tribe was in an opposite corner and as a result had no members of another tribe as neighbours, after 10 turns they came close enough to trade and at last one of the tribes died.

This brings us to the combat:

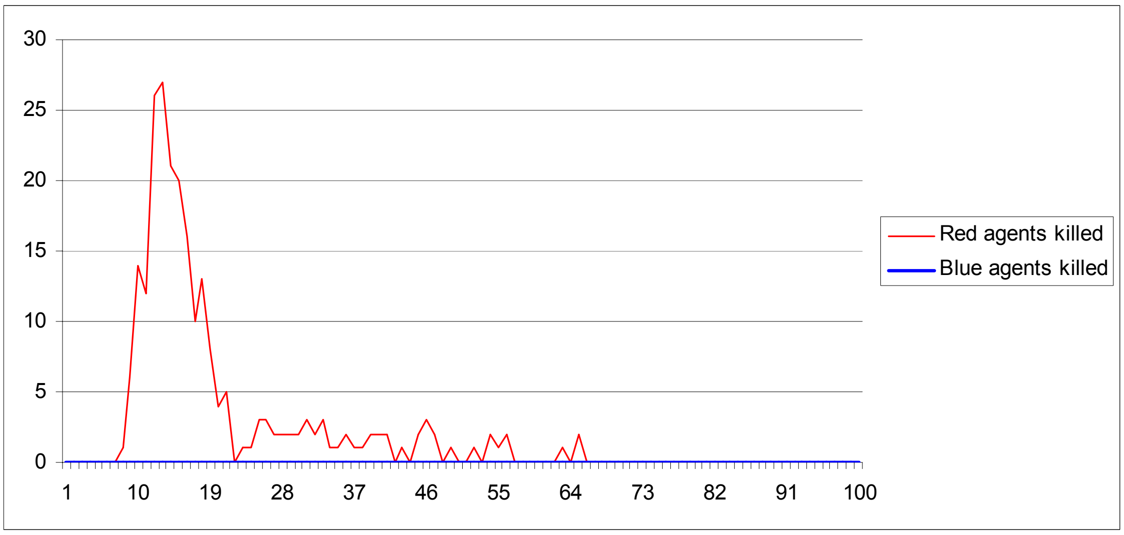


Figure 12: Kills

Figure 12 shows us how many agents died on hands of the other tribe.

It seems that only the red agents have been killed by the blue tribe, but not even one blue agent has been killed by the red tribe. The reason for that is that the red tribe would only have been able to enter the blue region if the region would have been too big for the blue agents to defend. This behaviour would be similar for two tribes, one of which builds walls and another which doesn't.

At the same time the red agents are not only weaker because they are not able to attack the blue ones, but also because they can not reach the sugar resources, which are the blue tribe's custody. And as we can see in figure 13, their space is getting smaller and smaller until they don't have enough space to live.

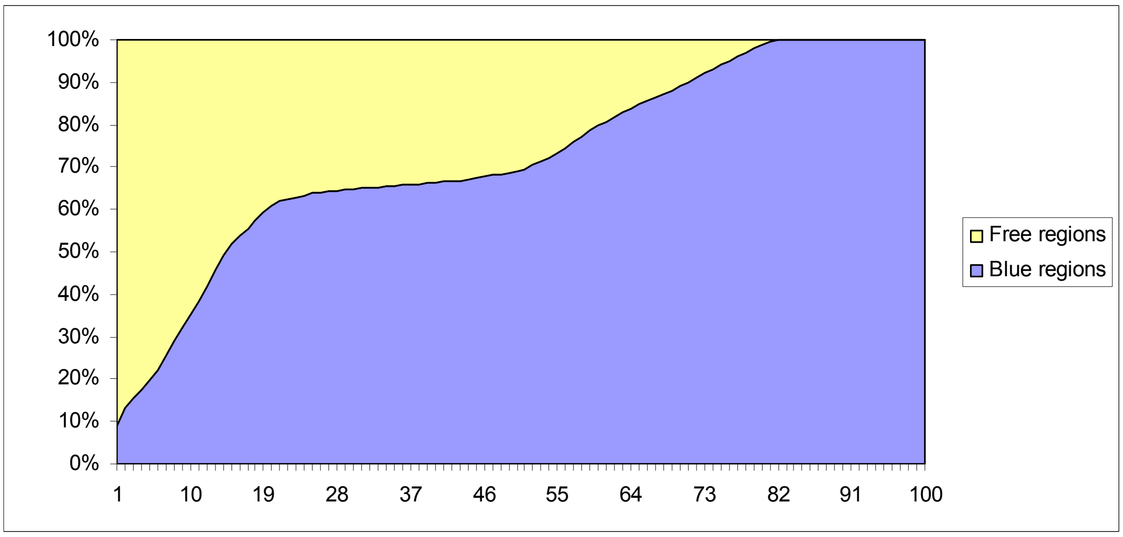


Figure 13: Amount of free and blue regions

As conclusion we can say that the trade was not that important to the blue tribe, the most important part was that regions banned the red tribe from their territory. To prove it, we tried again without trade and as shown in figure 14, the members of the red tribe died as fast as in the simulation with trade.

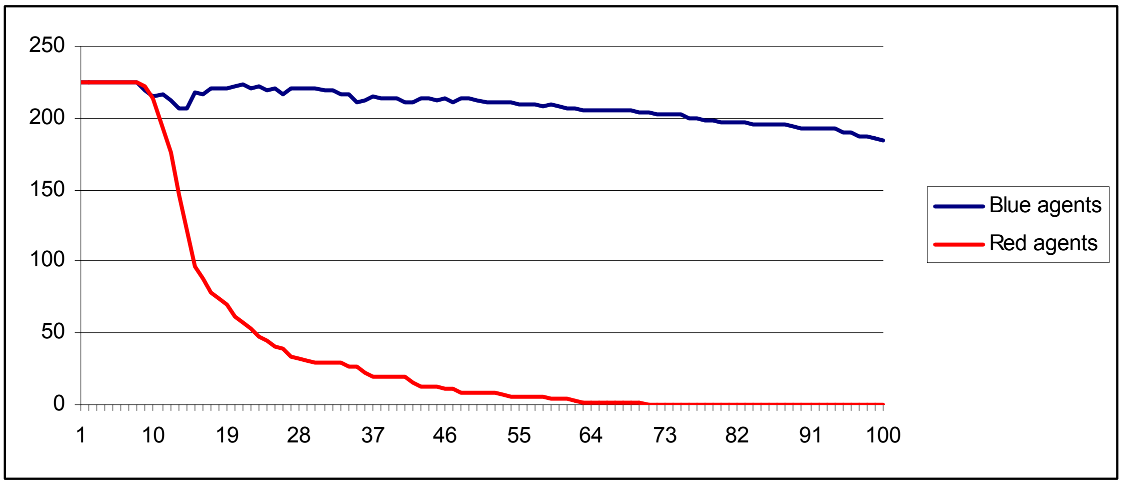


Figure 14: Living blue and red agents (simulation without trade)

Review

The biggest difficulty of this project has been the usage of the programming language XL in the GroIMP environment.
These were the reason for it to become a difficulty:

- At the beginning of the project the language still had no language specification.
- The language environment (GroIMP) is very basic, and therefore it is missing usual features nowadays seen as essential, like for example debugging tools.
- Compiling of the same code did not always have the same outcome. Sometimes, while compiling code with errors, the compiling process was marked as successful, while at the next time it was marked as with errors. These changes where mostly found after correcting another error, or after reopening a project.
- Sometimes, when compiling with errors, GroIMP would just shut down and the same would happen every time the project is attempted to open until the error is corrected with an external tool (such as notepad).
- Folding did not work properly in the jEdit window.

The difficulty of learning the language without a language specification could be handled with the help of Ole Kniemeyer (one of the current developers of the XL language), which helped me to understand the language as well as to provide code for features I could not manage to write with my knowledge of XL.

Some features have been added to the XL language in order to help to accomplish this project:

- Implementation of the setShader(Shader) method for primitive geometric objects. This feature has been used to colour agents, sugar, spice and regions.
- The ability to append bi-dimensional arrays to the graph has been added. Used to append agents, sugar cells, spice cells and regions at the initialization of the project.

The performance of the simulation system is quite good, since it only takes a few seconds to calculate 100 turns. Anyway, when simulating with territorial behaviour, the speed of simulation can slow down since it must calculate if a clan has more regions than it should very often. This problem can be handled if it is supposed that only maximal two clans will coexist on the sugarscape. However, this supposition has not been taken, and consequently simulations with territorial behaviour will take a little longer but will be able to work with more than two clans.

The transparency of the code is not as good as expected. Due to XL almost no code had to be written for the visualization, however in my opinion a much better result could have been achieved using custom folding (like for example the one supported by c#.net with the keywords #region and #endregion).

This project has been completed in three steps:
1. Reading of the book "Growing Artificial Societies" and at the same time writing the classes as pseudo code.
2. Gaining knowledge about the language XL and implementing the classes written in step 1.
3. Running simulations and writing the documentation.

This simulation system has been written based on [1]. However, some important differences may be appreciated:
- [1] has different simulation systems for different rule systems while our simulation system is complete (contains all supported rules). Different rule systems are simulated by turning rules on and off.
- Territorial behaviour is not supported by [1].
- Some rules may have been modified in our simulation system in order to make them compatible to all other rules since in [1] not all rules are compatible (for example combat and trade).

In the simulations, we could realize that territories, if owned, have the same behaviour as if a fort would have been built around it. Therefore if a fort is built around, you can still transmit information to the outside, and you can attack someone outside of it, but it will be difficult for someone to attack you when you are inside.

The great importance of owning territories in battle has been analysed in simulations 4 and 5. The importance of forts in real battle has been analysed in [4] and it is said to be decisive if used against traditional methods of battery and assault, and the authors have shown its superiority in battles like for example sieges checking Hapsburg heir Charles V's drive for hegemony in Italy in 1525.

References

[1] Joshua M. Epstein and Robert Axtell. 1996. “Growing Artificial Societies”. MIT Press, Cambridge/Mass.

[2] Ole Kniemeyer. 2004. “Rule-based modelling with XL/GroIMP software” (http://www-gs.informatik.tu-cottbus.de/grogra.de/publications/gwal6.pdf). Last used on 04.12.2006.

[3] Lehrstuhl Grafische Systeme, Brandenburgische Technische Universität Cottbus. 2006. “The XL Language Specification” (http://www-gs.informatik.tu-cottbus.de/grogra.de/xlspec/index.html). Last used on 04.12.2006.

[4] Emily O. Goldman and Richard B. Andres. University of California, Davis. “Systemic effects of military innovation and diffusion” (http://jciss.llnl.gov/syst.html). Last used on 04.12.2006.