Computer-based Simulation of Vegetation
—
Development of a General Grassland Distribution Model for an Application in New Zealand

Master Thesis

Michael A. Zaggl

Supervisors:

Prof. Dr. Klaus G. Troitzsch
Chair of Modeling and Simulation
Institute for Information Systems Research
Department of Computer Science
University of Koblenz-Landau, Germany

Ass.-Prof. Dr. Peter A. Whigham
Faculty Coordinator for Research
Department of Information Science
Division of Commerce and School of Business
University of Otago, New Zealand

Koblenz; September 30, 2008

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The natural world is a result of a complex interaction between biotic and abiotic factors. These interactions occur at a range of spatial and temporal scales and can be considered as a dynamic and evolving system. Vegetation structure is one important aspect of this system, and has been widely studied due to its role as both a biotic system and as a modifier and constructor of habitat. Observed vegetation patterns are a reflection of their evolution, adaptation, climate variability, ecosystem structure, and response to anthropogenic factors. They are therefore a fundamental system to understand in terms of environmental impacts, biodiversity and the maintenance and construction of suitable habitats. In addition, the succession of vegetation structure over a range of spatial and temporal scales is often required to understand the historical significance of biotic communities and the likely trajectory of future environmental systems. Hence, the ability to model aspects of vegetation distribution over a range of spatial and temporal scales is of fundamental importance to a wide range of scientific disciplines.

Vegetation modeling naturally involves the construction of a spatial and temporal framework to limit the types of interactions that will be considered. These frameworks differ in style depending on the type of model behavior that is desired. For example, a spatial framework may use a discrete model of space such as a lattice, where individual cells define a homogeneous region, or may define space as a continuous surface without an explicit scale associated with the representation. In a similar manner the temporal framework defines the rate and quality of change that can be considered, and is often associated with the life-cycle characteristics of the vegetation types under consideration. The models may also describe vegetation distribution as a simple density, a two-dimension position of individual plants, or may model both position and structure, allowing a consideration of the role of tiered vegetation as a modifier of habitat for other biotic species.

The work presented in this thesis has been motivated by the problem of being able to produce a simulation of tussock grassland distribution from the South Island of New Zealand. In particular, the model requirements were to be able to consider possible historical patterns and their relationship to currently observed patterns. This problem is important since there are conflicting views regarding the historical succession of these grasslands, and therefore one approach to addressing possible historical patterns is via simulation. Furthermore, two practical reasons for research regarding grasslands can be adduced. Firstly, grassland is an important part of agriculture and therefore for economy, particularly in New Zealand (Lemaire, Wilkins, and J. 2005). Secondly, invading plant species lead to fast changes in ecology of New Zealand (Fenner and Lee 2001) and a deep knowledge about grasslands is necessary to react effectively and efficiently to this problem.

This thesis presents a vegetation distribution model based on a cellular automata that can be generally applied. Flexibility is enhanced for the reason to apply it to different areas on the South Island of New Zealand, and grassland species, in particular different tussock species. Individuality of plants and spatially explicit representation are fundamental properties of the intended model. It should provide a framework for the research of botanists and ecologists and maybe it can help a tiny bit to protect this beautiful island in the Southern Pacific.

This thesis is structured as follows: Chapter 2 provides basic information about modeling. Therefore mathematical and computer-based simulation approaches for modeling are generally discussed. Thereafter (in Chapter 3) the field of vegetation is applied to modeling. Entities of existing vegetation models provide a basis for the development of a new simulation model and therefore the most important models will be introduced and discussed. In chapter 4 the requirements for the intended model are formulated and argued. The structure of the model is explained in the chapter *model description* (chapter 5). Different view points – regarding, for example, to the process or to the individuals – enhance an understanding of the whole model. The verification of the model in chapter 6 consists of documented tests and the interpretations of dependencies between parameter settings and results. The conclusion (chapter 7) shows the most important gains from this thesis and further implications.

Chapter 2

Modeling frameworks

2.1 Introduction to this chapter

Many different approaches and frameworks are possible considering the simulation or modeling of vegetation structures. This chapter gives an overview of general modeling frameworks and provides a basic vocabulary for the subsequent work. The first part describes the approach of mathematical models, followed by simulation model frameworks.
However, firstly two questions should be answered. *What is modeling and why is modeling useful?*

Modeling is the attempt to construct a replication of parts of reality (Troitzsch 1990). Therefore models are a focused view on certain aspects of a real existing circumstance. With this definition a broad range of concepts may be considered as models. Pictures, photographs or even synonyms can be rated as models. In the context of this paper we will only examine models for scientific use, and in particular for the modeling of vegetation patterns over space and time.

To answer the second question – what the use of modeling is – there are three major reasons. One reason to model is to make predictions. Predictions are necessary for planning and decision making. Another reason for modeling is understanding. Understanding of relationships between different aspects of a system is important for scientific work and essential for improvement of existing models. The third reason is to facilitate the management of complexity (Thornley 1998). Systems that exist in the real world are often very complex and difficult to handle, either because their interactions are too numerous to consider, or because information is not available for all levels of detail.
Modeling provides a powerful approach to reduce complexity. By reducing the complexity of a circumstance to a few important aspects, a valid scientific application can be enabled, even though the system as a macrocosm is too complex to be represented.

2.2 Mathematical models

Mathematical models are probably the most popular models and have a long history prior to the advent of computer-based simulations. They are also often the key elements involved in simulation models. Because of this strong connection between computer-based simulations and mathematics it can be interesting to mark out differences, but also similarities.

Mathematicians and computer scientists typically view computer simulation models as representing the mathematical structure of the objects they simulate. The flip side is that every simulation model itself is a mathematical object. As Matiyasevič proved (in his negative solution of Hilbert's tenth problem), any set which a computer program in general – and a simulation model in particular – can output, is *Diophantine*. A Diophantine set is any set whose members are those nonnegative whole numbers that give an equation that has a solution, when they are substituted into the variable x in some *Diophantine equation*[1] (Davis 1982). For example, the state of a system, which is being simulated, can be encoded as a number. The deterministic action of the computer-based simulation model on this number is matched perfectly by some Diophantine function. In this case both fulfill the same role. Therefore Diophantine functions are as powerful as computer simulation models.

A question therefore arises regarding the use of simulations: *Does simulation with a computer-based program make sense when – on the other hand – it is also possible to explain all circumstances with a (simple) equation?* The answer is given by the lack of possibilities to handle that equation.
Computer-based simulation modeling, however, has the advantage that it is much easier to manage than the mathematical approach. Simulation also gives a powerful alternative to utilize empirical research methods. This provides the opportunity to get a deep understanding of parameters' (or variables') impacts and their relationships. To get the capability to do empirical research with statistical measurements the simulation models framework has to provide detailed output information.

Mathematical modeling can be utilitarian for interpretation of outputs from computer-based simulation models, as well as for the development. Mathematical models are precise descriptions of relationships between variables. They may be, for example, differential equations or processes like Markov-Processes. Markov-Processes are often applied in modeling and therefore should be described. They are used for modeling as well as for analysis.
Another mathematical approach – particular to analyse and assess computer-based simulation – is the regression. Other mathematical models will be described later if they are needed.

2.2.1 Markov-Processes

The current state of a Markov-Process is independent of past incidents, and therefore the model is often represented as a fixed matrix of transition prob-

[1]A Diophantine equation is polynomial equation with integer coefficients where the variables also are allowed to be integers only (Ord and Kieu 2003), (Davis 1982).

abilities. If it is also possible to assume that time and states are discrete a Markov-Processes can be applied (Georgrii 2004).
Most simulation models are also independent from the past. Time is in almost every simulation model the determining dimension. Considering a certain part of a simulation run makes it obvious that the future is (partly) independent from the past. Certainly, the past has influenced the current state of the simulation run, but random values must be redetermined. Hence, the future behavior of the model is independent from the past and Markov-Processes can be applied as models.

2.2.2 Regression models

Regression models are useful to describe dependencies between different factors (see also (Berk 2003)). For instance, in vegetation the temperature is somehow related to the expansion of a certain plant species. The strength of this dependency can be measured by regressions.

2.3 Simulation models

In the field of vegetation dynamics analytical models are often used in the form of linear equations, which are typically overly simple for expressing the complexity of ecosystems (Shugart 1998). Computer-based simulation models, on the other hand, can provide a powerful approach to produce information about conditions in the future or by recreating and analysing past conditions. There are many general scientific fields in which simulation is used, especially in the natural sciences such as astronomy (e.g. (Pavlidou, Kuijpers, Vlahos, and Isliker 2001), (Winglee, Dulk, Bornmann, and Brown 1991)), biology (e.g. (Focks, Daniels, Haile, and Keesling 1995), (Ermentrout and Edelstein-Keshet 1993)), chemistry (e.g. (Lieberman, Chellamma, Varughese, Wang, Lent, Bernstein, Snider, and Peiris 2002), (Hassink and Whitmore 1997)), geoscience (e.g. (Raines, Zientek, Causey, and Boleneus 2002), (Lantuèjoul 2001)), medicine (e.g. (Eddy 2007), (Patel, Gawlinski, Lemieux, and Gatenby 2001)), and physics (e.g. (Birdsall and Langdon 2004), (Feynman 1982)).
Simulation models have also been used in social sciences (Gilbert and Troitzsch 1999), especially with the recent interest in modeling over networks. In the social sciences simulation modeling is used, for example, to produce information about economic theories, the behavior of groups or individual. Simulation models for political science can be seen as another example from the field of social science, as well as traffic simulations. Traffic simulation helps to plan and realize more efficient road systems and other urban infrastructure, through the modeling of individual behaviors and considering the resulting global patterns exhibited by the system. These types of models are also difficult to state mathematically and therefore simulation often allows complex systems to be modeled without an explicit mathematical basis. Here we can find another example for the interdependency of computer-based simulation and mathematics. However that may be, in each field, where simulation is used, different aspects must be considered.

The aim of this thesis is to develop a computer-based simulation model for a vegetation distribution in the Southern Island of New Zealand and the model's aim is to provide realistic behavior of the propagation and distribution of vegetation species over space and time. New Zealand is the most isolated continental island group (Halloy, Mark, and Dickinson 2001). Because of this isolation, the external influences are minimized and it is possible to perceive vegetation within this environment as a closed system. Hence, these islands provide ideal conditions for an application of a (simulation) model.

In summary, it is possible to argue that simulation models can be adapted to the field in which they are used. Possibly the main advantage of their application is to handle non-linear properties. The disadvantage is the lack of tractability and generality. Therefore the results of a simulation model are always uncertain and they have to be seen critical. To compensate for this disadvantage it is possible to compare different results from identical simulation models out of different runs with each other (also referred to as *Monte-Carlo-Simulation*). Thereafter statistical analysis can facilitate the measurement of reliability and other indicators of the models goodness out of varying results.

2.3.1 Cellular automata

Cellular automata consist of topologically or metrically structured identical cells typically arranged in a grid. Each cell can be in one of a number of predefined states (Gilbert and Troitzsch 1999). The grid is therefore a representation of discrete space.
Time proceeds also in discrete steps (Gilbert and Troitzsch 1999), although the model may *run*[2] in either a synchronous or asynchronous form of updating.
The cells states and the adoption of other states are controlled by rules. Often these rules consider the states of the neighbor cells. To clarify which cells are a neighbor basically two simple definitions for a two-dimensional grid exist[3].
Firstly the *von Neumann neighborhood* uses the north, south, east, west cells and the current cell, thus giving five cells (see figure 2.1a). Secondly the Moore neighborhood represents the neighborhood as nine cells, the current cell itself and the surrounding eight cells (north, north-east, east, south-east, south, south-west, west and north-west) (see figure 2.1b). It is possible to assign the von Neumann and the Moore neighborhood into a three- or more dimensional context. In a one-dimensional context both definitions of neighborhood are identical.
The shape of each cell in a cellular automata's grid is not necessarily a square. As long as the cells' number of dimensions is equal to the number of dimensions of the grid, every conceivable shape is possible. (For example, a two-dimensional cell cannot be assigned to a three-dimensional grid.) Therefore a regular tessellation is not compulsory and irregular tessellations are also possible.
Figure 2.2 shows a grid of regular tessellation (a) and an example of irregular tessellation (b). For a regular tessellation the form of shape can also be a triangle or a hexagon (Kier and Witten 2005) and even more complicated shapes

[2]A simulation run is from now onwards defined as "a single time path with fixed values for all its inputs and parameter" (Kleijnen 1998).

[3]In fact they are also applicable for one- and three-dimensional grids, but in principle the definitions are the same, independent of the amount of dimensions.

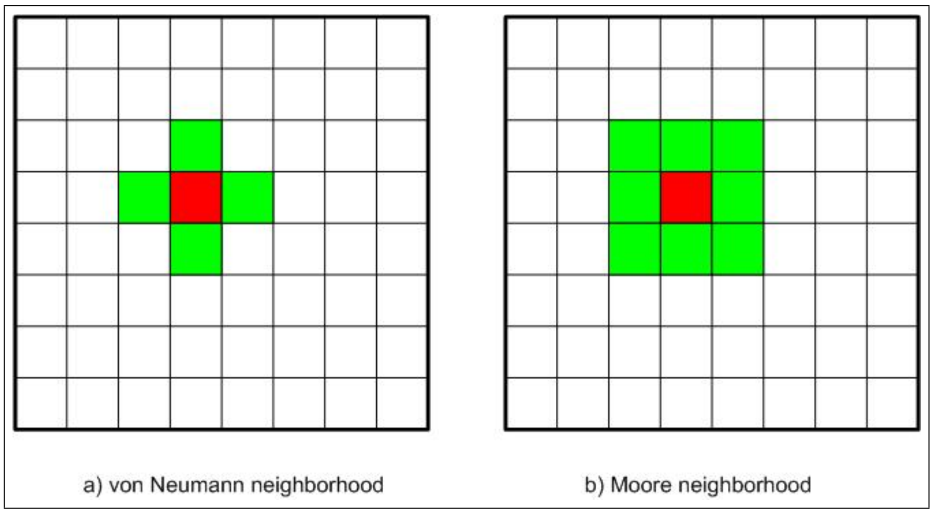


Figure 2.1: von Neumann and Moore neighborhood

are feasible for regular tessellations[4].

Cellular automata defining are commonly defined as a grid of regular, or identical cells (e.g (Wolfram 1998) , (Gutowitz 1991)). However, some models require the representation of space as irregular cells, and that is possible (Flache and Hegselmann 2001).
This (little) disagreement derives possibly from the different fields where the cellular automata have been applied. For instance in ecology it is often useful to have identical cells and thereby a regular tessellation, if the objects of simulation are homogeneous enough to assume an identical structure. In contrast to that stands the field of social science, as an example. Different objects can be represented by different sized cells. On the other hand, is it possible in ecology to aggregate homogeneous areas to cells, and these cells are often geometrically different, and if additionally the grid should keep its spatially explicit characteristic, it has to be irregular. That is often the approach in ecology (Polack and Stepney 2005).
However, making a general statement that links a science field to either regular or irregular tessellation is not possible. The important aspects is that both – regular and irregular tessellations are possible.

It is feasible to portray a cellular automata's topology not as grid, but as a network. The frequent usage of the term "grid" in literature (e.g. (Dijkstra, Timmermans, and Jessurun 2000), (Zhang, Sato, Takahashi, Muraoka, and Chiba 1999), (Toffoli and Margolus 1987)) suggests that the structure must be homogeneous, but in fact every topology for a cellular automata is possible. Certainly, the most common topology is the grid, but the term *lattice* is (slightly) more formal and allows a more abstract view on topology, because it includes the network topology.
In a network topology the cells' positions are not explicitly required to be spa-

[4]Some paintings of the artist M.C. Escher are examples for such shapes.

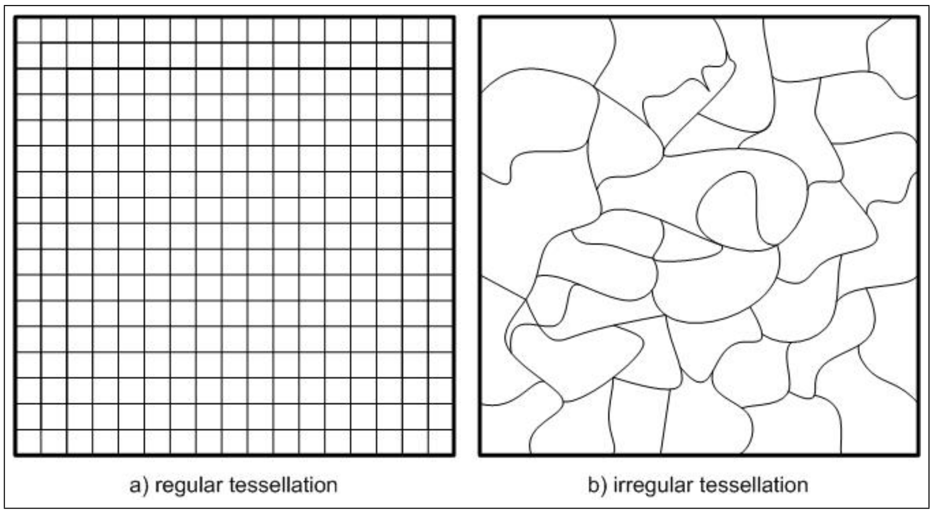


Figure 2.2: Regular and irregular tessolations

tially located, however the topological relationship between cells (i.e. the neighbors) are defined by the connectivity of the network. This abstract way to design neighbors allows spatial relationships that are not necessarily related to any form of multi-dimensional space, although the edges of the network may in fact be associated with a spatial dimension.
The network topology can embody every possibly neighborhood. That means that the grid topology is just a particular kind of a network topology. Therefore the network topology is a powerful approach, but its modeling is more difficult than a regular topology. Concerning topology a statement from Toffoli and Margolus is quite appropriate. They argue that basically every cellular automata is a network of interacting cells (Toffoli and Margolus 1994). Hence, there is a need to distinguish between a regular (a) and an irregular network topology (b) as shown in figure 2.3.

In summary, there are two main criteria for structuring a cellular automata. One is the tessellation and the other is the topology. The regular tessellation, combined with a regular topology seems to be the most common form of a cellular automata[5].

2.3.2 Agent-based models

The central concept of agent-based or multi-agent-based models are the self-contained programs or agents. Based on their perceptions of the environment the agents make decisions and are consequently controlling their own behavior (Huhns and Singh 1998). Four fundamental characteristics can be stated for agents. Firstly, autonomy, each agent has direct control over its behavior. Secondly, social ability, that makes an interaction between different agents possible. Thirdly, reactivity, an agent can perceive the environment and interact with it.

[5]By verifying the simulation model that will be developed, a function *wind* shows how powerful the network topology is (see section 6.2).

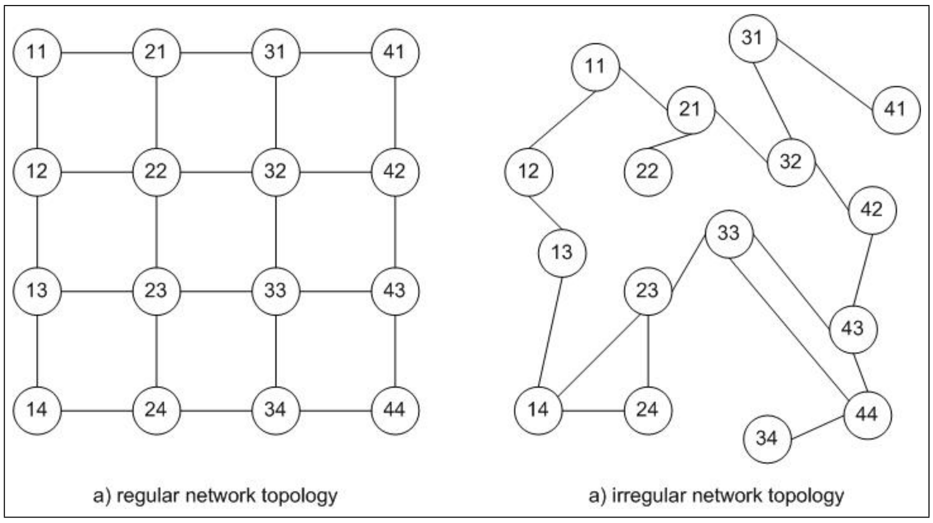


Figure 2.3: Classic cellular automata and network topology

Lastly, proactivity, that means agents are able to take initiative. By taking initiative their strategic behavior becomes notable (Wooldridge and Jennings 1995).
The difference between agent-based and multi-agent-based models is just the fact that agents in a multi-agent-based system work in teams, which allows the simulation of group behavior and cooperation.

2.3.3 Individual-based models

In agent-based models and cellular automata the aggregation of several individuals to one object is possible. However, modeling individuals means a strong disaggregation of these objects. Every single individual is modeled and simulated separately (Grimm, Berger, Bastiansen, Eliassen, Ginot, Giske, Guss-Custard, Grand, Heinz, Huse, Huth, Jepsen, Jørgenson, Mooij, Müller, Pe'er, Piou, Railsback, Robbins, Robbins, Rossmanith, Rger, Strand, Souissi, Stillman, Vabø, Visser, and DeAnglis 2006). This provides some advantages, because the modeling is very detailed. On the other hand, it limits the number of simulation objects.
Vegetation modeling often needs large numbers of simulation objects, but aggregation is difficult for a natural environment. Biodiversity leads to heterogeneous areas, even over very small areas. The decision as to whether individual-based modeling or aggregation should be applied cannot be made in general.

Chapter 3

Current approaches and methods in modeling vegetation

3.1 Introduction to this chapter

The current approaches to model vegetation distribution, particularly in New Zealand, are based mainly on Markov-Processes (Lough, Wilson, Mark, and Evans 1987) or finite state models[1] (Grove, Mark, and Dickinson 2002) (Westoby, Walker, and Noy-Meir 1989).
Finite state models are often used in ecology (see (McPherson and DeStefano 2003), (Stringham, Krüger, and Shaver 2003)), especially in modeling of grassland systems (see (Humphreys 2007), (Cowling, Richardson, and Pierce 1997), (Rodriguez Iglesias and Kothmann 1997)). A finite state model contains a catalog of alternative states and transitions from one state to another. The transitions are triggered by natural or artificial events and they provide the dynamics of the system (Perrings 1995).
Because of the need to define all possible states before application, a finite state model cannot produce general novel results. Interaction between individuals can certainly produce results that are not expected. In terms of basic novelty, however, the modeling method is limited, because states which are not considered are not possible.
Computer-based simulation models, on the other hand, can provide richer dynamics compared with finite state models. The reason for this is that simulation models do not require the explicit definition of all possible states in advance.
The first step to consider, regarding the simulation model – which will be developed as a part of this study – will be a review of the current literature. Different types of models to simulate vegetation will be discussed. Important aspects for comparing and describing these models are given in table 3.1. If the literature provides enough information, all aspects will be discussed in the context of the existing models.
Later an evaluation of the most useful approaches will enhance the development

[1]Finite state models in ecology are most often called *state-and-transition models.*

Form of spatial representation
Individual-based model versus area or distribution models based approach
Spatial modeling
Spatial scales
Temporal modeling
Temporal scales
Paramterization
Limitations and assumptions
Possibility for integration of real data

Table 3.1: Aspects for the development of a vegetation distribution model

for a general computer-based simulation model for grasslands in New Zealand.

3.2 Structuring existing models

In the current literature a range of different vegetation models exists. Every type provides different advantages and disadvantages. Vegetation models are strongly connected to landscape models and often the term *landscape model* is used for both – landscape models and vegetation models (Perry and Enright 2006). However, they are basically not the same, since landscape models often consider additional aspects like erosion. However that may be, these terms should not lead to confusion even though they are sometimes used synonymously.

Initially it is useful to consider the structure of existing models. Different aspects will be used to group the models. Such aspects can be the scientific method or the object of the model. Doubtless there are many dependencies between these aspects, but structuring the models in this way provides a better understanding for the important aspects of this project's model. The review will provide a background for the subsequent approach of this work and will hopefully give new perceptions of the scope and limitations of the work.

3.2.1 Mathematical models in vegetation

In the field of vegetation dynamics mathematical models are often used in the form of linear equations (Horn, Shugart, and Urban 1989). However, they are typically overly simple for expressing the complexity of ecosystems (Shugart 1998). The typical mathematical model in ecology is the regression model, often expressed as a combination of linear terms.

Often it makes sense to use mathematical models to evaluate single aspects of a model. The influence of one parameter on the predictions of a model, generally described as sensitivity analysis, can often be used to determine critical aspects of the modeled system. This has already been discussed in general (see section 2.2).
The existing mathematical models in the field of ecology and vegetation are not useful for the purpose of this paper, because of their specialization on certain

aspects or empirical data. An exception could possibly be the application of these data at the stage of verifying outputs from the simulation.

3.2.2 Simulation models in vegetation

Simulation models are often used in the field of ecology, but these models are adjusted for special regions and species, as we see later. Transformations to other regions or characteristics seem to be difficult or even impossible. A complete new development might be easier. However, it makes sense to discuss these approaches to get knowledge and ideas for the development of a model.
To structure vegetation simulation models, two main approaches are of note. Firstly, the area of *spatially explicit landscape models* and secondly, the *gap replacement models* (Perry and Enright 2006).

Spatially explicit landscape Models

Spatially explicit landscape models are often used to simulate the spread of fire in vegetation areas (Sklar and Costanza 1991) and they have been extensively refined and extended since their initial development in the 1970s. Nowadays spatially explicit landscape models are often combined with geographical information systems (GIS) (Perry and Enright 2006). Miller et al. define vegetation modeling as "predicting the distribution of vegetation across a landscape based on the relationship between the spatial distribution of vegetation and environmental variables" (Miller, Franklin, and Aspinall 2007, p. 225). Hence, spatial explicitness is a required element for vegetation modeling.
In vegetation modeling the spatial environment and relationships are clearly a central criteria. To prevent potential confusion, it must be stated that the criteria of a spatially explicit representation depends on the model's purpose.
For example, economic models are often spatially explicit in their field only, but from an ecological point of view the same models are not spatially explicit (Anselin 1992).
An example – out of the field of economy – can be a model where agents communicate. All agents are scattered somewhere in a space. The ability to communicate is not influenced by the distance between the agents. Hence, the exact position does not matter and they are placed randomly. In ecologic terms the model is not spatially explicit, because of the random placement, but for the economists the localization is not important, they can assume that there are modern ways of communicating, like telephone.

Spatially explicit landscape models are often regarded to large areas (i.e. small scales) and for that reason most of them are highly aggregated (Pausas J. and Noble 1997). On the other hand, unaggregated – individual-based – landscape models are presentational, but they must be applied to small area (i.e. large scales), because of limited computing power.

Gap replacement models

All gap replacement models are based on the same kind of design. They are also known as *mechanistic approaches* (Martin 1992) and consist of discrete areas which are also called *patches*. The whole environment is set up like a mosaic

	JABOWA	FORET	FORICO	ZELIG	BRIND	Kiambram
Light	X	X	X	X		
Climate	X	X			X	
Moisture	X					
Leaf litter		X				
Mineral soil		X				X
Herbivore		X	X			
Seed pool				X	X	
Fire					X	
Vegetative Reproduction		X			X	
Phenology of seed						X
Dormancy			X			X
Site availability					X	X

Table 3.2: Different filters in entities of gap models (Leishman, Hughes, French, Armstrong, and Westoby 1992, p. 603)

of these patches (Leishman, Hughes, French, Armstrong, and Westoby 1992). Perceiving the mosaic as a lattice and the patches as cells leads to the idea that this definition is quite similar to the description of a cellular automata (see section 2.3.1). In fact gap replacement models are individual-based simulation models in a cellular automata (Perry and Enright 2006).

Often the gap replacement models are labeled as *gap models* only. They all simulate the dynamics of tree populations at annual time steps[2]. Growing is based on the current size and it is negatively influenced by crowding or shading. In addition, mortality is often considered (Shugart 1987).

Gap replacement models have been used for different research objectives. To get an overview of the different entities of gap models table 3.2 shows different gap models and which filters[3] are used.
The most important existing gap replacement models will be described next:

JABOWA: JABOWA is an acronym for the developers names (Janak, Botkins and Wallis) and can be seen as the master pattern from which all other gap replacement models are derived. It was developed in 1972 and it is specialized for modeling forestry systems (Botkin, Janak, and Wallis 1972b). One of the major aspects that are modeled in JABOWA is the canopy size of a tree (Shugart 1984). This is important for light distribution. Smaller trees or plants may be disadvantaged, because vast and dense canopies of bigger plants prevent the availability of the resource light (see also (Aussenac 2000)). The modeled space is two-dimensional and the

[2]Later an exception will be introduced (see the ZELIG model on page 14).
[3]Filters are certain properties of the environment (Leishman, Hughes, French, Armstrong, and Westoby 1992). Therefore filters are parameters that represent the environment and aspects of species, too.

calculation of light availability occurs only by considering the canopies of the neighbored trees (Botkin, Janak, and Wallis 1972a).
A special aspect about JABOWA is that the cells cannot communicate at all. They only receive instructions. The cells are discrete and arranged in a lattice (Deutschman, Levin, Devine, and Buttel 1997). The cell's lack of the ability to communicate seems to be extraordinary in the context of a cellular automata.
The cells size is constructed for the assumed maximum size of a canopy (Deutschman, Levin, Devine, and Buttel 1997).

FORET: FORET stands for *Forests of Eastern Tennessee.* It was developed by Shugart and West in 1977, and it is a descendant of JABOWA. The most distinctive change is the ability of the cells to communicate. The cells of FORET are arranged discretely, as well as in JABOWA. A more detailed modeling framework has been implemented allowing trees to sprout out of the root or the stem. Also the resource light is a central aspect in FORET (Deutschman, Levin, Devine, and Buttel 1997) and reproduction by seedlings is new compared to JABOWA (Shugart and Prentice 1992).

FORICO: The FORICO model was developed by Doyle in 1982. Its purpose is to simulate the Puerto Rican rain forests under the effects of hurricanes (Shugart, Smith, and Post 1992). It can process 36 different tree species (Doyle 1982).
The model simulates a forested area of 1/30 ha and the time steps are – as usual for gap replacement models – yearly. Growth, death and establishment of new trees is the modeled sequence. The major result is that a diversity in the vegetation structure can help to prevent damages from hurricanes (Doyle, Shugart, and West 1982).

ZELIG: The ZELIG tree simulator model was designed to consider the growth and mortality of trees. Urban finished the model in 1980. The temporal interval which is used is one month (Miller and Urban 1999). ZELIG retains all important features of its parent model which is the FORET model. The scale of the model is an area of 10 x 10 meters for each cell. The number of cells is 900. Therefore an area of 90,000 square meters can be simulated by the model (Urban, Bonan, Smith, and Shugart 1991).

BRIND: BRIND is another model developed for a certain region. It simulates a 1/12 ha area of an Eucalyptus forests near Canberra in Australia. It was created by Noble, Shugart and Schauer in 1980 (Noble, Shugart, and Schauer 1980).
The model has many successor models – for instance the EDEN model (Pausas J. and Noble 1997) – but it cannot provide novel ideas of modeling vegetation. However, the model and its successors show that a specialization to other regions makes a complete remodeling necessary. This can be prevented by the development of a common model.

Kiambram: The Kiambram model is an approach by Shugart in 1980. The model was used to simulate the sub-tropical rain forest in Australia (Vanclay 1995). The name is derived from the aboriginal word for *thick forest.* Kiambram allows the modeling of 125 species and is a detailed gap replacement model developed by different repetitions of design, application and verification (Shugart, Mortlock, Hopkins, and Burgess 1980). Subsequently the OUTENIQUA system was developed based on concepts from this model.

OUTENIQUA: OUTENIQUA is named after a mountain range in South Africa and was developed by van Daalen and Shugart (van Daalen and Shugart 1989). The model seems to be very specialized, and therefore does not contribute to general modeling techniques.

FORENZ: The reason for mentioning FORENZ is that is has been developed for forests in New Zealand (Shugart, Smith, and Post 1992). It has been derived from JABOWA and FORET (DeVelice 1988).
The model uses an area of 1/10 ha and can represent five certain species. It is designed very simply and forgoes parameters like temperature and light (DeVelice 1988). Because of this simplicity it is easy to handle, but it is also surprisingly reliable. Comparing the results of field observations (see (Mark, Scott, Sanderson, and James 1964)) with this model was used to support the models coherence.

FORMIND: The FORMIND model is very well described in the literature compared to most other models. Developed by Köhler and Huth in 1997 it uses an area of 20 x 20 meters. An interesting approach in FORMIND is that similar species are aggregated to functional groups (Köhler and Huth 1998). Hence it is basically not individual-based. This provides the advantage that many different species can be modeled in a simple way. FORMIND was successfully evaluated and further versions were developed (Köhler, Ditzer, Ong, and Huth 2001).

SORTIE: SORTIE is based on modeling and simulating the diameter of a tree stem. The resources which are contained in the model are water, light and nitrogen. Wind and fire can be simulated in SORTIE, but its principle task is to get information about the growth and mortality of tree communities or single trees and the amount of wood they can produce (Deutschman, Devine, and Buttel 2000).
Because of the purpose of this model it seems to be developed for economic forecasting only. Forest management is probably the only field that is interested in the amount of wood.
Competition in SORTIE happens only with regards to light. The developers of the model argue that the simulation addresses a real existing part of a forest[4], but the positions of the individuals are determined randomly

[4]More precisely it is regarded to an oak forest (Pacala and Deutschman 1995) in the northeast of Connecticut in the USA (Deutschman, Devine, and Buttel 2000).

(Pacala, Canham, Saponara, Silander, Kobe, and Ribbens 1996). This may seem contradictory, but in a homogeneous area, this approach could be useful.
The gap replacement model can process empirical data and it works as an individual-based simulation (Deutschman, Levin, Devine, and Buttel 1997). The ability to handle empirical data seems to be a useful concept, for obvious reasons. Otherwise, the model would be too much based on assumptions that cannot enhance the quality of results or be justified.
The functionality of light should be described more detailed, because it seems to be the key function in SORTIE. The competition between trees in SORTIE is only about light. This seems to be typical of gap replacement models. The data about light is an aggregated value for a whole season. The distribution of light is calculated for each individual tree. The simulation program regards the size, density, position and species of the neighborhood. After the amount of light for each tree is calculated, the resulting value is used as the basis to determine the probabilities for mortality for each tree and the growth rate (Deutschman, Devine, and Buttel 2000).

Common to all of the gap replacement models is that they are developed to model forests or trees. Often vegetation simulation is put on the same level as forest simulation. However, since forest models are principally concerned with the growth and management of a controlled environment there are some doubts as to how applicable their methods are when general vegetation patterns are considered.
Also the competition about light – which is essential in all gap replacement models – seems to be less important for small sized vegetation. Smaller plants like tussock grass and and other grassland species, like herbages, are not influenced by their neighbors in terms of light.
These plants are not so high that they can trigger shade, neither do they have canopies. Experimental research examining shade for grassland species has shown that a strong influence between mortality – particular for seedlings – and shade exists (Silvertown and Tremlett 1989), (Goldberg and Werner 1983). However, the trigger for shade in those experiments has not been natural. They had to place some artificial barriers to prevent light. Hence, light as resource can be ignored by examining grasslands and therefore the gap replacement models are inappropriate. However, even if the models of the gap replacement are not suitable to the purpose of grassland simulation, many ideas are derived from these methods of modeling. So the individual-based approach seems to be successful tested by those models. Also the gap replacement models show that small areas – like 1/30 ha for FORICO or 1/10 ha for FORENZ – are useful to model, and not too small for vegetation models.

3.2.3 Forest, non-forest and vegetation models

It can be useful to distinguish models, which are solely concerned with trees and groups of trees, the *forest models*, from models which consider only plants (without trees) and from models which consider any form of plants, the *vegetation models.*

Forest models

Many forest models are developed to predict the growths of trees. Mortality and reproduction is not included. These models are used in forestry (Botkin, Janak, and Wallis 1972b). Their aim is to provide projections of future income potentials. The stem diameter and the growth of a tree is more important than for example competition between seedlings, because human influence can control seedlings.

The important forest models are already discussed (see section 3.2.2). They can provide new ideas for the development of a vegetation distribution model, however both approaches – forest models and vegetation distributions models – are very different. Therefore a more detailed discussion makes no sense in terms of the goal for this research. Instead non-forest models should be examined.

Non-forest models

Non-forest models are models which consider smaller plants only (Shugart, Smith, and Post 1992). Hence models that examine grassland are also non-forest models. Two models that are described in detail could be found in literature.

AcevedoRaventòs: AcevedoRaventòs[5] is a model which simulates three species of grass in the Venezuelan savanna during a period of one year. Fire is an important aspect, because usually once a year widespread fire destroys the plants, except the roots. The model is based on very detailed data collection and shows an effective way to implement reproduction. However, it should be mentioned that the three vegetation species considered by this model exclusively use a reproduction strategy by placing shoots (Acevedo and Raventòs 2002).
The model provides an advantage by being very simple. On the other hand, the period of only one year is quite short and allows no long-term measurements. Fire is the reason, but it occurs regularly only in certain regions.

WinklerKlotz: The model from Winkler and Klotz[6] is applied to a certain grass area in Germany. The plants in the model use two strategies to reproduce: Cloning and sexual reproduction. The simulated area is divided into cells of size 1 x 1 cm arranged in a lattice of 100 x 100 cells, which means large-scale (in the sense of small area) behaviors are considered. Different weather influences are considered (Winkler and Klotz 1997).

Vegetation models

Specialized models for universal vegetation – trees and for instance grassland – are rare. Modeling vegetation in general is much more complex than the particular development of a forest or grassland environment.
The first justification for this statement is the fact that mainly data is available

[5]This model has no special name so the developers names are used.
[6]The authors did not designate the model with a name as well as Acevedo and Raventòs.

about pure forests. The governmental and economic concern regarding forests seems to be more significant than the scientific interest of common vegetation or plants (Shugart, Smith, and Post 1992). Another reason is the already mentioned complexity of vegetation systems. Vegetation includes trees and smaller plants, at the same time, while the forest models consider trees only. The interaction between trees are not so complex especially when the quantity of different tree species is low. Also the size of entities is quite similar. If plants and trees interact in one model, aspects like the light distribution must be considered more carefully than in models without smaller plants, as well as in a model of smaller plants without trees.
Furthermore the space which a plant takes is usually smaller. Hence, there are more plants in a given area than trees, and the properties of the system may require relationships to be modeled at multiple scales. The number of vegetation models is noticeably less than the number of forest models (Shugart, Smith, and Post 1992) or the non-forest models.

3.2.4 Different approaches to vegetation-climate models

Vegetation is influenced by the climate but in addition the climate is also influenced – vice-versa – by vegetation (Martin 1992). On the basis of this fact the existing models in vegetation distribution can be categorized in a different manner. Possibly some aspects of the vegetation-climate models are assignable for the development of a grassland distribution model, although a feedback from grassland to climate is unlikely.

Probabilistic approaches

The probabilistic approaches are based on empirical data out of surveys (Martin 1992). This seems to be useful particularly in areas which are not influenced by human activities like forestry or other anthropogenic impacts. Probabilities can be generated out of the survey data. Probabilistic approaches assume that single plants can be replaced with plants of other species or also with the same species (Martin 1992).
An example for a probabilistic approach is the model *FATE*[7]. FATE uses a discrete pattern to simulate a vegetation area. Reducing complexity of different species is possible by grouping different species in cohorts (Moore and Noble 1990) which is a similar approach to considering functional groups. Criteria for this grouping is, for example, the age of plants. Hence the type of species can be ignored if the behavior is similar given a similar age. The fact that the parameters of the FATE model are static over the simulated timespan should be noted as a limitation. On the other hand, it can be useful for certain research foci. The coefficients of the model, which determine the probabilities of assumed aspects for each species, are constant during the whole simulation process (Martin 1992).

Inferential approaches

The inferential approaches are linked to the fact that plants should establish and grow to a point of maturity. For that purpose data out of biology and botany

[7]Fate is an acronym for *Functional Attributes in Terrestrial Ecosystems* (Terradas 2005).

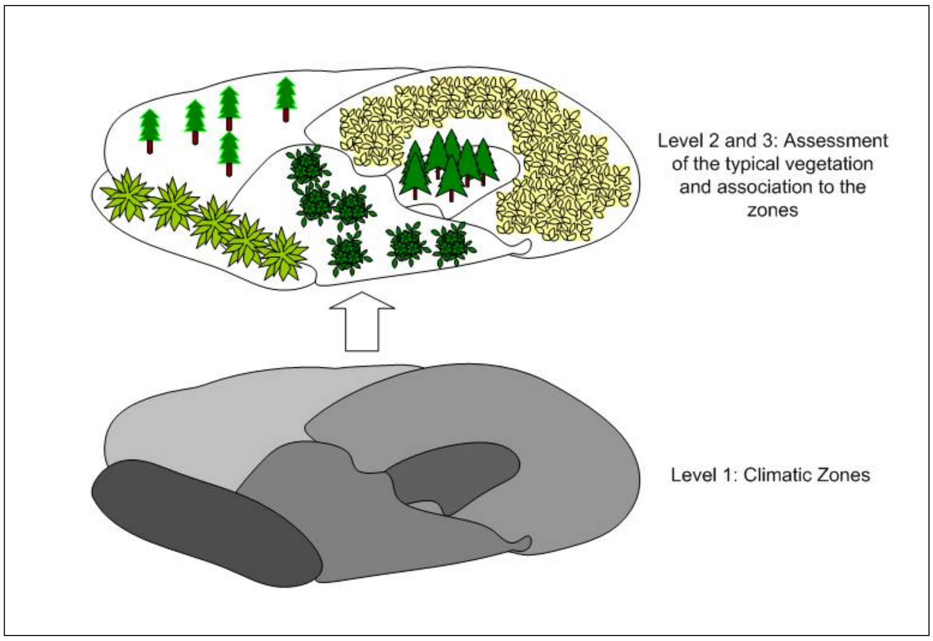

Level 2 and 3: Assessment of the typical vegetation and association to the zones
Level 1: Climatic Zones

Figure 3.1: Empirical approaches

is needed. Measurements out of laboratories provide the data about the plants physiology, for instance the demand of water or the responses of vegetation to a change in temperature or other environmental conditions (such as soil nutrients) (Martin 1992). The main object of inferential approaches is the vegetation and not the climate, even though the climate is also part of these models.

Correlative approaches

The correlative approaches are divided into empirical, statistical and threshold approaches (Martin 1992).

- Empirical approaches

The empirical approaches are very similar to the inferential approaches. The difference is their focus. While the inferential approaches are mainly linked to the individuals (or groups of individuals), the empirical approaches are respectively linked to environment and climate.
The empirical approaches are based on empirical data of the environmental conditions. The environment itself is modeled in four levels on the basis of climatic zones. They are distinguished by climatic criteria, like the influence of light, warmth, etc.. Given a particular climatic zone the current vegetation is modeled. Typical areas of vegetation are summarized and linked to the environment. The result is a typical vegetation associated with the climatic zone (Emanuel, Shugart, and Stewenson 1985) (see figure 3.1).
The simulation of these models allows the prediction of changes in the vegetation caused by a change in climatic conditions. For example, if the climatic conditions in one area are changing, another vegetation (which is typical for the

new climatic conditions) will replace the current vegetation.

- Statistical approaches

The statistical approaches work with the so-called *climate space* (Gates 1980). The climate space is a set of environmental conditions in which an organism can exist (Martin 1992). Therefore it is necessary for the survival of the organism that the conditions of the environment are a subset of the climate space.
The reason for the name *statistical approaches* seems to be that statistical measurements are used to determine and characterize the set of climate spaces.
One example from the statistical approach is the BIOCLIM model. In BIOCLIM the sets of environmental conditions for the survival of species are declared as *envelopes* (Walker and Cocks 1991). The model is very popular and also in use for modeling possible distributions for animal species (Lindenmayer 1996).

- Threshold approaches

Threshold approaches add to inferential, empirical and statistical approaches some dynamic aspects, or constraints. For example, the propagation of species into new areas is limited to one kilometer per year (Martin 1992). This assumption would probably be adjusted, depending on the species and environmental conditions. For instance, because of the prevailing wind direction.

Ecophysiological approaches

These kind of approaches use experimental data about an organisms minimum requirements. These data comprise the amount of sun light, the minimum temperature and the water balance (which is the ratio of incoming and outgoing water in the vegetation) (Woodward and Rochefort 1991).

3.2.5 Other models

There are other models in the field of vegetation modeling, which do not easily fit into the current classification. Models which possibly contribute new aspects or ideas to the aim of this research will now be discussed.

Invasion models

Invasion models in the field of vegetation focus on the competition between plants. Tolerances of environmental conditions are important. Because of the focus on competition, individual-based models are the most common approach in this field (Bolker and Pacala 1999).
In the field of invasion models regarding vegetation distribution additional influences are provided by modeling wind (Hastings, Cuddington, Davies, Dugaw, Elmendorf, Freestone, Harrison, Holland, Lambrinos, Malvadkar, Melbourne, Moore, Taylor, and Thomson 2005). The influence of wind could also be a crucial influencing factor for non-invasion models.
Invasion models are basically very similar in their structure to non-invasion models. The difference to non-invasive models is primarily the focus and the distinction of invading and native species.

Genetic focused models

Genetic based models are applied to animal and plant species and basically concentrated on propagation and success of certain genes in a population. Environmental aspects are considered. The best described simulation model in the literature is QU-GENE (QUantitative-GENEtics) (Wang, van Ginkel, Podlich, Ye, Trethowan, Pfeiffer, DeLacy, Cooper, and Rajaram 2003).

QU-GENE is a highly flexible software with graphical interfaces and detailed opportunities to generate outputs (Podlich and Cooper 1998).
The genetic focused models seem to be all considering humanly genetic manipulated species. These are the best explored organisms. For a natural vegetation model that works with genetic aspects the required data is probably not (yet) available.

3.3 Deeper approach to vegetation models

In terms of designing a general simulation model of the vegetation distribution in the South Island of New Zealand the WinklerKlotz model seems to be the best described simulation model in the literature. However, for using the model's structure as a kind of guide line, the large scale that the model uses could be a problem. As mentioned before the model uses only a patch-size of one square meter (see section 3.2.3). This has the disadvantage that only an extraction of a vegetation area can be modeled, but it provides a certain accuracy.
Also it must be considered that the WinklerKlotz model is developed for a certain region and certain species. The aim of this research study is to develop a general model, that can be applied at different places in New Zealand and manage different species. Even the amount of species should be variable.
Because of the detailed description of the WinklerKlotz model in literature, the next approach will be a closer discussion about the WinklerKlotz model, after that a small-scaled vegetation distribution model will be discussed. It might be possible to combine the accuracy of the large-scaled WinklerKlotz model with aspects of a small-scaled model. Therefore the WinklerKlotz model should be discussed in more detailed and furthermore at least one small-scaled model should be introduced.
To get an overview of the spatial scales that are in use for modeling ecological properties Kirkby (1999) provides a framework. In figure 3.2 the typical scales of time and space for research in ecology and geoscience are shown. The range of possible scales is satisfactory, in particular for the spatial dimension.

3.3.1 A large-scale model: WinklerKlotz

The WinklerKlotz model is spatially explicit, it uses real distance scales and not simplified distances (Winkler and Klotz 1997). This is important for a vegetation model, because spatial distances are crucial for distribution in general, and in particular for representing competition between single individuals.
The reason for choosing such a small patch size (one square meter) was partly due to the limited computer performance available in 1997 (Winkler and Klotz 1997). However, it is also useful to discuss a large-scale model as argued earlier. In the study area, the researchers found five different species. However, only

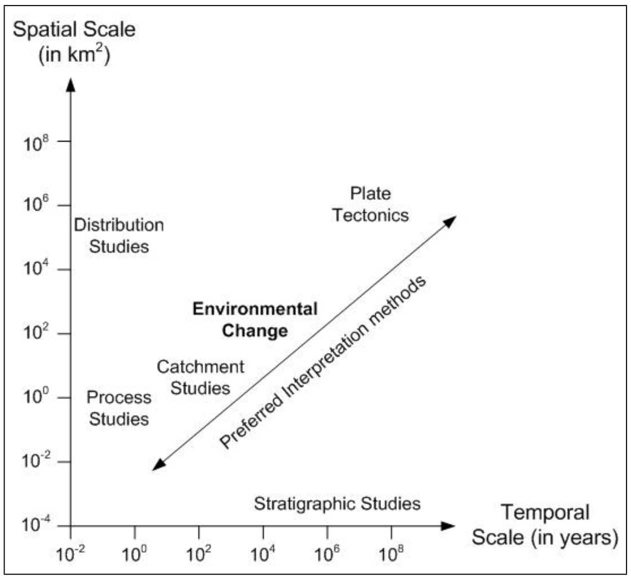


Figure 3.2: Temporal and spatial scaling in ecology and geoscience modeling (modified from (Kirkby 1999, p. 191))

three of them were modeled. The model is cell-based which means one cell contains one individual (or a part of an individual) or it is empty. The possibility for bigger plants to occupy more than one cell also exists. These tufts are modeled with a central or original cell surrounded by peripheral cells (Winkler and Klotz 1997).
The individuals are characterized by age, by the position of its (central) cell and the radius measured by the amount of cells if it is a tuft. By reaching maturity the individual is capable of producing seeds. New seeds can only germinate on empty cells which implicitly introduces resource competition to the model. The lengths of an individuals life is determined by probability distribution depending on their current age (Winkler and Klotz 1997). The different life expectancies of the species are considered as part of the model.
The probability distribution predicates an individuals minimum life span with a probability of 100%. That means that plants cannot die before the break in the distribution graph is reached. This possibly seems to be oversimple for obvious reasons (see figure 3.3).
Seed dispersal is not modeled explicitly in WinklerKlotz. Every empty cell can be populated by a species. When more than one germ arrives on an empty cell, a competition happens. Each species has a certain probability to prevail (Winkler and Klotz 1997).
It is essential to model spatial interactions explicitly in an ecological model (Miller, Franklin, and Aspinall 2007), this is especially true for small-scaled models. Because of the large modeled scaling, used by Winkler and Klotz, spatial structure is not so important for seed dispersal. However, because of the spatially explicit modeling of the individuals, it should easily be possible to extend the model by considering these aspects at a smaller scale.
The growths of individuals is simplified, since only the growth for tufts is con-

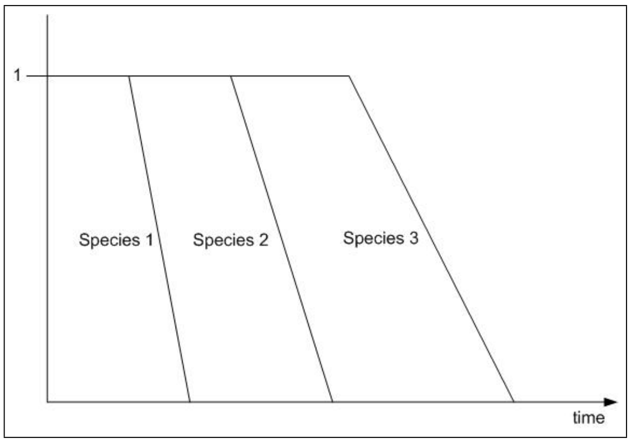


Figure 3.3: Conceptual probability distribution of life span depending on age (Winkler and Klotz 1997, p. 191)

sidered. If more than one individual wants to enter a cell by growing the same competition rules, as in case of seed dispersal, are applied (Winkler and Klotz 1997).
Light availability seems to be important in most of the vegetation distributions models. This availability strongly depends on the height and also on the species in the surroundings of the considered individual. Height is a result of growth and this should be modeled in any vegetation model where light competition is relevant. In WinklerKlotz it is not important to consider the height of the individual, since they seem to be at a similar level regardless of their age or species.
Parameterization in WinklerKlotz is provided for fecundity delay, diaspore productivity per cell, mortality, mean number of fragments per tuft, mean diaspore import[8], probability of diaspore export[9], mean dispersal distance, seedling establishment probability, seedling competition strength and (relative) competition strength for the growing of tufts. All parameters exist for each species (Winkler and Klotz 1997).

The ten parameters used in WinklerKlotz are manageable. In the vegetation distribution model for the South Island of New Zealand some additional parameters would be useful, although, of course, additional parameters require values to be given, and these may not be known. For example, precipitation and temperature are examples of more complex environmental conditions. However, the work of WinklerKlotz shows that it is possible to use a short list of parameters for model complex vegetation patterns successfully.
Additionally, the WinklerKlotz model provides another feature. It is possible to enlarge the size of the simulated area by designing more than one patch and creating connections between all of them (Winkler and Klotz 1997).
In summary, the WinklerKlotz approach seems to provide a sound basis for the

[8]The WinklerKlotz model does not assume that the simulated area is completely isolated. In fact there are *virtual* individuals outside the area that spread diaspores. The parameter *diaspore import* steers the number of diaspores coming out of the not-simulated area.
[9]This is the opposite of diaspore import. Some of the diaspores leaving the simulated area.

model development of the general vegetation distribution model for the Southern Island of New Zealand. The small area which is considered by Winkler and Klotz can be the most significant difference between WinklerKlotz and the intended model. However, the advantages for a simulation model of a small area cannot be ignored. For example, a verification by empirical data from surveys is much easier, furthermore the survey itself can be arranged without a large effort. Nevertheless, a consideration of smaller scaled models should be analyzed prior to a decision being made for the design of intended model.

3.3.2 Two small-scale models: MEDRUSH and MAPSS

Small-scale vegetation models are very rare. In fact only two serious types exist. MEDRUSH is one of them and it is developed as a part of a project of the European Union. It is applicable in several regions, but mostly applied to areas close to the Mediterranean Sea. The maximum size of area in the simulation model is 5,000 square kilometers and time spans are up to 100 years. The time step used in the model is one hour (Kirkby 1999). The model is not just focused on vegetation distribution, it also contains erosion and hydrology (Kirkby 2001), and therefore is more complex than the intended modeling problem of this project.
Parameterization is for example given by assumptions regarding rainfall, solar-energy and temperature (Kirkby and McMahon 1999).
In terms of the complexity of a large-scale model, aspects like hydrology and even erosion seem to be inseparable from the environment of the modeled vegetation. In a model that is just focused on vegetation the causes and effects of erosion can perhaps be omitted, although this would largely depend on the environment and whether the geomorphology and changing environmental structure is crucial to the growth of vegetation or not. For MEDRUSH the time step of just one hour seems to be required for the detailed modeling of the erosion, whereas this time scale is not required when we consider vegetation patterns that have slow growth and maturation rates. Hence, for the modeling and simulation of vegetation distribution there are little reason for requiring such a detailed time step.

As mentioned before MEDRUSH is not focused only on vegetation. It shows that small-scales are possible, but it does not provide a good basis for the intended vegetation distribution model in New Zealand. The vegetation is modeled, but the focus is too different for the purpose of our model.

Concerning the general vegetation distribution model in New Zealand the model MAPSS[10] seems to be a more useful approach than MEDRUSH.
MAPPS is focused on vegetation distribution and it also considers water balance (Neilson 1995). The dependency between these two aspects is obvious (Stephenson 1990). The parameterization describes physical limits of the environment for species like temperature (Woodward 1987), and therefore similarities to the correlative approaches that consider climatic envelopes (see section 3.2.4) are likely.
Attributes concerning the vegetation are *biomes*[11] and their distribution. The

[10]The *Mapped Atmosphere-Plant-Soil System.*
[11]Biomes are a community of species which are living in a specific ecological region.

	large-scale model: WinklerKlotz	small-scale model: MAPSS
Form of spatial representation	spatially explicit	spatially explicit
kind of model	individual-based	aggregated model
Spatial modeling	1 square meter	ca. 50,000 square kilometers
Spatial scales	unknown	1 : 10,000 meters
Temporal modeling	discrete	discrete
Temporal scales	1 year	1 month

Table 3.3: Large-, and small-, scale models in respect of the aspects for development of a vegetation distribution model

leaf area index[12] and its pattern is the other vegetation attribute. For the calculation of the water balance it is also important to consider the stomatal conductance of the leafs, depending on the season (Neilson 1995).
Many parameters are utilities in the model. Environmental conditions (for example, rain, snow or a mixture of rain and snow) are considered in MAPSS. In addition, the model distinguishes between precipitation, which stays in the canopy and that, which falls through (Rutter, Morton, and Robins 1975). Furthermore wind spread is implemented in the model (Neilson and Marks 1994). Figure 3.4 gives schematically an overview of environmental influences covered by MAPSS and its complexity, only for the water balance. By integrating light and other parameter it becomes, of course, even more extensive.
The time steps are monthly and besides water, light is an important resource. Individuals are highly aggregated by calculating, for example, the average of the leaf area index (Neilson 1995).
The only information available about the spatial aspects of the model is that MAPSS has a resolution of ten kilometers and it covers circa 50,000 square kilometers (Neilson 1995). Therefore it seems to be a continuous spatial model with a high aggregation.

3.4 Summary of existing vegetation models

Table 3.3 shows the important properties of the models that have been previously discussed in detail.
The different models are diverse and they show a range of possible environmental parameters like water, temperature, light and soil. In particular, the MAPSS model shows that the parameters are strongly connected with each other. This circumstance demands much complexity.
However, to develop a general model these aspects may be too difficult to consider. For verifying the model all these parameters must be assigned with values and a lot of these values must be estimated, if no information is available. And this possibly makes the resulting model uncertain. Furthermore a focus on vegetation distributions is needed. A typically grassland should be modeled and most of the aspects of the large-scale models seem to be unimportant.

[12]The leaf area index is percentage of the leaf area relative to the ground area (Neilson 1992). It is an important indicator for transpiration and transpiration is crucial for the water balance.

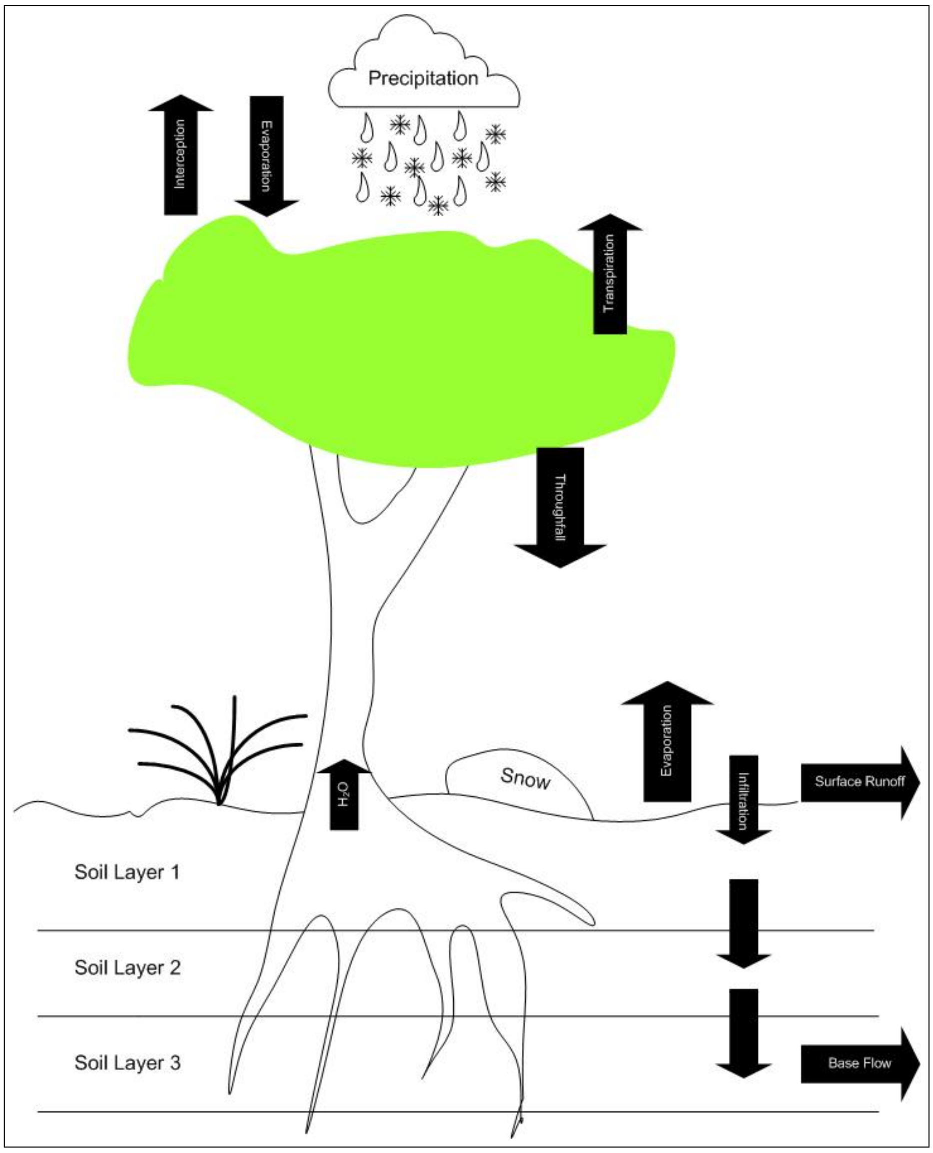


Figure 3.4: Modeling of the water balance in MAPSS (modified from (Neilson 1995, p. 363))

On the other hand, the WinklerKlotz model provides a solid base. Its only disadvantages are its large-scale and a conspicuousness regarding tufts.
Ignoring of plant height in the WinklerKlotz model seems to be useful. Grassland plants are not big enough to be an obstacle in the vertical dimension. The horizontally growth of tufts in WinklerKlotz, on the other hand, is critically implemented. In the WinklerKlotz model the tufts grow horizontally and this additional growth is not viewed as the origin of new individuals. If the oldest part of a tuft dies in WinklerKlotz the whole tuft dies as well (Winkler and Klotz 1997). This seems to be unrealistic against the background of botanical literature in grassland dynamics, because tufts consisting of more or less independent individuals and for that reason a tuft can survive even when some parts of it have died (Tomlinson, Dominy, Hearne, and OConnor 2007).
Hence, a plant's horizontally growth should be better replaced with reproduction. Additionally, the specialization of the WinklerKlotz on a certain area with certain species is somewhat limiting. The model for New Zealand needs more flexibility.

Chapter 4

Model requirements

Chapter 3 has described many aspects for the modeling of vegetation and therefore also for the development of such a model. The intended model should be spatially explicit. Thereby arises a problem, because the model should also be independent from certain species and areas. In addition, as many species and regions as possible should be applicable for the model. The only constraint is that the species should be domestic or invading grass-plants of southern New Zealand, and the environment is also located on this island.
Most of the models in chapter 3 are specially focused on certain areas and species. Hence, very precise modeling is feasible. Therefore detailed information about region and species are necessary. Such kind of information, however, is not available for this project. Moreover the model should not be very specialized. A general model is needed, which allows the configuration of the species and the environment. Also the number of species should be modifiable. Hence, only abstract species and environments are applied for the verification of the model (see chapter 6).

Because of the support for a spatially explicit representation a cellular automata with individual-based entities is implemented. The updating procedures work synchronously. The spatial design depends on the size of the species. In principal, a size of 20 x 20 centimeters per cell is assumed. This is the average diameter of a tussock[1] (Münzbergová, Křivaneká, Bucharová, Jukličková, and Herben 2005). The size of the lattice is changeable. The basic configuration is 100 x 100 cells. Therefore an area of 400 square meters is modeled. That seems to be a useful approach to simulate grassland vegetation at a large scale.
The dynamics of the system are implemented by interaction of the species with its environment. Therefore the Moore neighborhood is applied, because it seems to be the most realistic approach. Plants can breed inside an eight-cell environment (see also section 2.3.1, especially figure 2.1). Additionally, wind is modeled and enlarges the environment. It can be determined how many seedlings will be spread per period. The basic setting is one seedling per plant in each period[2]. Of course, the plant must be mature to get this ability. To occupy a cell the

[1]Tussock grass is widespread in New Zealand's grasslands and therefore it provides a solid orientation (Calder, Wilson, Mark, and Ward 1992).

[2]In the verification of the model (see chapter 6) wind is completely switched off, except for the experiments where it is mentioned.

seedling must enter an environment with conditions that fit the requirements of the species. Otherwise it dies. The change of the environmental conditions – temperature and humidity – while the simulation is running is also possible.
Because of the small size of the modeled area, all cells have the same likelihood that a descendant of a plant reaches it via wind, there are no disadvantages due to the distance. Furthermore all cells should be treated equally. Therefore cells on the border of the lattice should have as many neighbors as cells in the middle (the solution for this requirement is described in section 5.2.1).

The WinklerKlotz model (see section 3.3.1) will be a raw role model, not at least because of its detailed description. The aspects of reproduction by seedlings from the FORET model is also implemented. These seedlings must reach a cell and afterwards they must establish on it. Because of the lack of well described vegetation models in the literature, the model of this thesis will be described in detail. Further development and progress in science is otherwise very difficult or even impossible[3].

The implemented parameters can be divided into parameters of the environment and parameters of species. The environmental parameters will be temperature and humidity.
The reason for choosing humidity is that the water balance is an important aspect of most of the models. It is hard to get detailed information about water balance, but in grassland it seems to be a good opportunity to aggregate the complex system of water balance to humidity. Information about humidity is probably available in an easy way.
Temperature is the other essential environmental parameter. In times of global warming the temperature is crucial and should be considered. It has not been modeled in all the discussed models of chapter 3, but it seems to be essential for obvious reasons.
The third potential environmental parameter is soil. For that, information also is rarely – or not at all – available. Another reason for not modeling soil is that it is often seen as an indirect factor in botany (Burke, Lauenroth, Vinton, Hook, Kelly, Epstein, Aguiar, Robles, Aguilera, Murphy, and Gill 1998). Hence, the program should provide the ability to extend the model for soil parameters.
The reasons for renouncing light as parameter has already been explained (see 3.3.1). In a grassland the shade is not crucial.
Species parameters are age, maturity, and life expectancy. The life expectancy should be modeled more realistic than Winkler and Klotz (see figure 3.3). One possibility is to use a gaussian distribution. A preferred humidity is also important for each species, as well as preferred temperature.
Of special importance is the competition of different species. Availability of information about competition in flora is limited. However, particular data about the competition of seedlings from different species exists. Most botanical studies are of short duration, therefore research is concentrated on competition effects of seedlings and not of adult plants (Sackville Hamilton 2002). Hence, in the intended model the competition will be implemented for seedlings. It is assumed that one cell of their model can only be occupied by one plant. This is

[3]For this very reason the source code of the computer program is fully available (see appendix A).

a sound approach for a spatially explicit model and also used by Winkler and Klotz (see section 3.3.1). However, compared to the WinklerKlotz model, the competition rules are extended (see 6.4.12).
To enhance the dynamics of the model, the plants of a species should not be absolutely equal. Some opportunities for mutations should be enabled. The implementation of this requirement is explained in the next chapter (5).
For more details about the parameterization see table 5.1.
Handling of empirical data is a basic requirement. Hence, changing the settings for species and environment must be easy for the user.

The intended model transfers different characteristics from the models of chapter 3. Firstly, it is a non-forest model (see section 3.2.3) and it also fits into the field of probabilistic approaches (see section 3.2.4), because the assumptions about replacement are basically the same.

The model should also correspond to *Liebig's law* that is also known as the *Law of the minimum.* Liebig's law is a very basic assumption in botany. It claims a dependency of plant on the minimum quantity of a resource (Martin 1991). If one need of a plant is not satisfied the others are not important any more. This law seems to be very crucial for modeling vegetation.

Chapter 5

Model description

5.1 Introduction to this chapter

The model structure for the spatially explicit general vegetation distribution model will be described in this chapter.
The model is encoded in Java and therefore independent from any platform. Since there is little information regarding the form of species and environmental conditions during the design phase, the software will be implemented to be as flexible as possible. Another reason for this decision is to enhance further development. Changes and modifications for specific usage are a central consideration of the software design and should allow a range of vegetation and environmental properties to be used without recoding the model.
Time and space are implemented discretely as a cellular automata. The aggregation is on an individual level (see chapter 4).

The description of the developed model will be structured into a functional and procedural specification to enhance efficient usage. An object-orientated depiction assists further development. This flexibility is a tribute to the complexity and diversity of different possibilities for the application of the model in the field of vegetation simulation.

5.2 Functional and procedural description

The functional description can be divided into two major parts. Firstly, the *environmental view* allows an understanding of the model concerning the environment. The cellular automata consists of cells and therefore the environment is the complete set of cells, associated with their spatial arrangement. This arrangement is a lattice with a regular tessellation and a partly irregular topology[1].
Secondly, the *individual view* provides an understanding of the plants behavior. The assumptions for their behavior are argued here. For the usage of the simulation model it is necessary to comprehend details about the parameters and algorithms.

[1]The irregular element is given by the including of wind (see section 2.3.1).

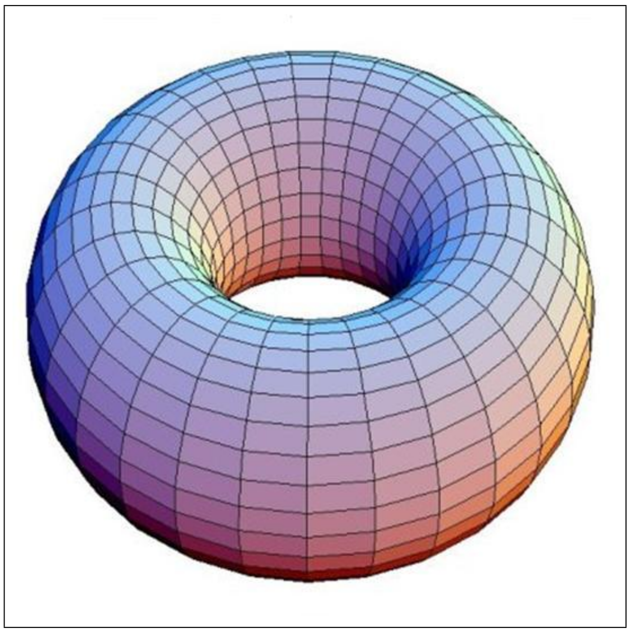


Figure 5.1: Torus as model shape

Finally, the process of the simulation program is described. This gives a detailed understanding of the model as a whole. The process, of course, depends on the functions and vice versa. However, the functions are more constitutional and for that reason they are specified firstly.

5.2.1 Environmental view

The environment is modeled as a discrete surface. Each cell, ordered in a lattice, represents an element of the surface. Concerning the vegetation of a surface, the cells can be basically in one of two states: Either they are occupied by a plant or not[2]. Therefore it is not possible for two plants – even if they are from the same kind of species – to occupy the same cell at a single time.

To avoid issues where plants are located at the border of the lattice, the surface is represented as a continuing surface without boundaries. Hence, the cells directly located at the border of the lattice are neighbors of the cells at the opposite border (see figure 5.1). This structure – also known as *torus* – is not natural, but the abstraction provides the advantage of equality of all cells.
The cells conditions constrain the plants behavior by using different parameters. These parameters are temperature and humidity. Temperature and humidity of each cell are determined by two digital maps, one for each parameter. These parameters are not necessarily static over time. Basically they can be influenced to all possible values of their spectrum at any time (see section 5.2.3). It is also possible to use the same map for both parameters. The maps will be scaled automatically by the software to the size the model needs. Alternatively

[2]By considering temperature, humidity, or the kind of plant that can occupy the cell, the amount of states is actually unlimited.

a determination of the parameters by random values at the setup of the lattice is enabled. This setup happens only once for each run. More precisely, at the beginning, during the initializing of the simulation program. To determine the random value two gaussian distributions – one for temperature and one for humidity – are applied. The reactions of the plants concerning these two parameters are described later in the section *Individual View* (5.2.2).
Both environmental parameters are related to each other in respect to Liebig's Law. That means that substitution between temperature and humidity is not possible. If one condition does not match a species' requirements, then the condition of the other parameter is not important, irrespective of how suitable it is.
Temperature is modeled as whole numbers on a scale of 0 to 255. This is reasoned by the fact that the usage of digital maps is possible. The color of each pixel will be identified and translated into RGB-Encoding. In the common 24-Bit encoding RGB provides 256 different possible values for each of the colors red, green and blue (Gonzalez and Woods 2007). In the simulation model the color red is used for temperature and blue for humidity. The need for translation of the interval [0,..,255] into a Celsius scale is essential to use the model in a more realistic way. The exact conversion depends on the required spectrum on the Celsius scale. Therefore it is necessary to find and use values that reflect real environmental conditions for the Southern Island of New Zealand.

During one year the average temperature differs a lot in some regions of New Zealand. For instance, the mountain structure on parts of the Southern Island leads to cold temperatures – values below zero on the Celsius scale are common here – even in summer time, but they can also be very high (Leisnham, Cameron, and Jamieson 2003). This means that the temperature is volatile in these altitudes and makes an aggregation of temperature to an average number for a whole year problematic.
For the use of the simulation model – that should be developed – the most important group of plants is possibly tussock grass (see chapter 4) and these species do usually not live in the mountains[3] (O'Connor 1982). Therefore the volatile temperature in these regions is not a problem for the simulation model.

It is useful to find a *typical* grassland area on the Southern Island where this species usually lives. The annual average temperature in that region can be applied for the average temperature for the random values in the model, and – even more important – it can also give an idea how to convert the RGB-Encoding to the Celsius scale.
A matching example for a typical tussock area is Wye Creek Conservation area. It is located beside Lake Wakatipu near Queenstown in Otago (45°09'S, 168°46'E). The average annual temperature is +10.4° Celsius[4] (Miller and Duncan 2004). The application of a gaussian distribution is useful for the same reason to use it for the species.
Wye Creek's altitude is in-between 515 and 575 meter above sea level (Miller and Duncan 2004). Therefore the volatile behavior of mountain temperature during a year cannot be assumed.

[3]Exceptions are species of the snow tussocks (Rose and Platt 1987), (Mark 1969).
[4]Recorded over 130 years (Miller and Duncan 2004).

Taking the opportunity to aggregate the available information about the monthly temperature in this region, to an annual average is supported by the fact that the standard deviation is only 3.8° Celsius (NIWA 2008).
This means that not only for the random values, to determine the temperature for the model' cells, an aggregation of temperature is possible, furthermore it is feasible to aggregate temperature for a whole year. Hence, annual time steps are applicable for the model.
It is useful to determine the middle of the RGB-Scale to the mean of the temperature. That means that a RGB value of 127 (for the color red) enunciates 10.4° Celsius. The next step is to find the most simple equation that covers an acceptable spectrum for the model on the Celsius scale.
The equations which probably meets this condition best is linear and converts one step on RGB scale with 0.1° Celsius (see equation 5.1).

$$Temperature(x)_{^\circ Celsius} = 0.1 \cdot x_{RGB} - 2.3 \tag{5.1}$$

With that equation an interval of −2.3° until +23.2° on the Celsius scale is supported.

5.2.2 Individual view

At the beginning of the simulation the plants are located randomly on the lattice. The amount of one species at the beginning is independent from the amount of other species[5] and it is represented by a probability, which is applied on each cell of the lattice.
It is also possible to control exactly the placement of the plants at the initialization, that means at the beginning of a run.

The plants are parameterized with an age. The age determines maturity and death of a plant. For both events a gaussian distribution provides a functional dependency from the plant's age (see equation 5.2).

$$f(x) = \frac{1}{\sqrt{2\pi}\sigma} \quad e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{5.2}$$

In nature gaussian distributions can often be observed (Stewart 1990). Therefore it is reasonable to use this function unless there is other evidence, such as empirical data, that is available.
At a first glance the gaussian distribution does not appear to be a natural condition for the death of organisms. Intuitively the deviation must be negatively skewed, because an early dying for a plant is more likely than an age that is almost double of the average. For instance, the likelihood that a plant dies in the first 2% of its expected life time is much higher than the likelihood for reaching an age of 198% of the average life time of its species. Therefore the mean is smaller than the mode. Hence, the deviation is not symmetric, rather it should be left-skewed (see (b) in figure 5.2).

[5]By verifying the model it becomes clear, that even so, there is an indirect dependency between the amount of different species for random initialization (see section 6.4.1, and especially equation 6.3).

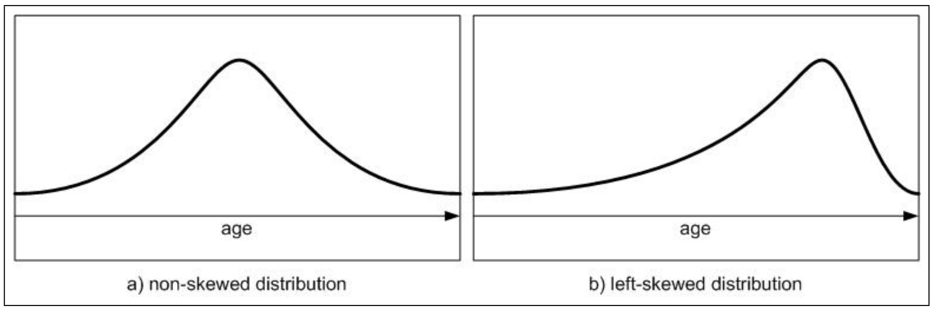


Figure 5.2: Symmetric and left-skewed distribution (modified from (Engel, Möhring, and Troitzsch 1994, p. 42))

However, plants seem to behave different. Observations of different plant species show that the mortality is the highest in early seedling stages. Ignoring the early dying of seedlings leads almost to a symmetric distribution (Hamann 2001). Hence, it is useful to use a gaussian distribution (see (a) in figure 5.2) and model the high seedling mortality by competition of seedlings. For the deviation regarding the maturity no study could be found. Therefore, the normal deviation will be assumed, too.

In the simulation model, the mean (μ) and the standard deviation (σ) are characteristics of the plant species. On the basis of these two values a gaussian function is calculated and the current age determines the possibility to breed offspring or the death of a certain plant.

Two different solutions are possible to implement this in the simulation model. One possibility is that the simulation program can calculate the values for the probability of maturity and death during the whole simulation run. Whenever the time to breed has been reached, the value will be calculated and determines if the plant can breed or not. The same applies for death. The other possibility is to calculate the values beforehand at the first possible point of time, which is the individual's birth. Both solutions provide different advantages and disadvantages.
The first possibility, which is the repetitive calculation of the value in every breeding (or dying) phase, needs obviously more calculations. On the other hand it seems more natural and intuitive at a first glance. The plant has to face the current circumstances and these conditions have an immediate influence. They may, in some circumstances, change over time, and therefore it would not be realistic to assume that the death of a plant could be determined at the time of its birth. Therefore the decision whether a plant can breed or must die is the question of the current situation and must be continually assessed. However, another, and more profound disadvantage is the fact that the results can be inconsistent. If, for instance, the possibility for a plant's maturity will be calculated and the species' average age for breeding is five periods and the standard deviation is two periods ($\mu = 5$ and $\sigma = 2$) then a result of the gaussian equation of four is likely. Furthermore it is assumed that the age of the plant is five periods. Hence, the algorithm will allow the plant to breed, because the current age of the plant is higher than the random value under consideration of

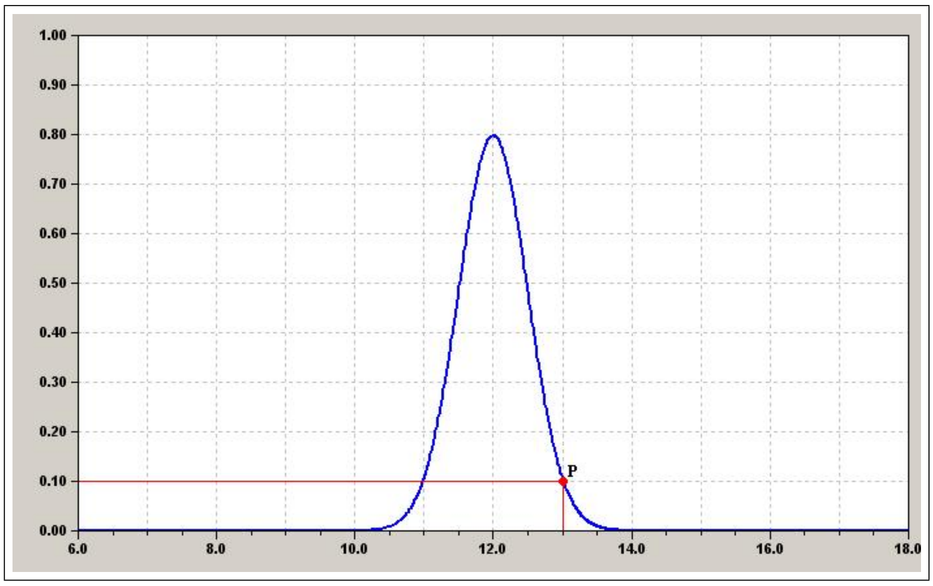


Figure 5.3: Gauss function with $\mu = 12$ and $\sigma = 0.5$

the gaussian distribution. However, the inconsistency can occur in the next cycle when the plant has the age of five and if the gaussian distribution calculated a result of six for the plant's maturity age. The consequence is that the plant cannot breed in this cycle. This seems not to be visceral, because in the period earlier it had the ability to breed. However, plants do not breed at every time period, and so there is a procedure for this behavior. An algorithm provides the possibility to constrict the probability for breeding. It is possible to constrain the amount of seedlings or the whole ability to breed.
The application of the gaussian distribution is more meaningful and realistic, if it is used independent from the cycles of the model and only calculated at the beginning of a plants existence – the birth. It works like destiny for the individual plants but it is probably more compatible with most of the empirical data.

In detail the program determines the values for maturity and death, separately, at the very first point of time when the plant is born. Thereby the gaussian function uses species' specific values for the mean and the standard derivation. For instance, the species is characterized by a mean of twelve periods and a standard derivation of a half period ($\mu = 12$ and $\sigma = 0.5$) for its death. With a probability of ~10% the plant gets a life of 13 periods length (see figure 5.3).
The calculation of the preferred humidity and temperature works by the same approach. With a gaussian function the two values are calculated, separately. The control of the exact value can be determined by the input of mean and standard deviation.

The competition strength works without a gaussian deviation. It is only a real number that defines the strength of a species-species competition.

Table 5.1 summarizes the parameters of species. The values are freely selectable by the user.

5.2.3 Process view

The simulation process works in periods (or time steps). Each periods consists of one cycle. These periods continuing until the program is stopped by user intervention. Before the periods can commence a setup is necessary. This setup creates the lattice which defines the spatial universe for the plants. With a certain probability on each cell of the lattice, a plant of a species will be placed. This is determined by the probability chosen by the user (the so called *setup ratio*).
The run of the simulation is controlled by the user. Run means the start and the continuation of the time steps of the model (see also section 2.3.1).
After the setup, the first step of a cycle is the subsequent aging of the plants. Every existing plant ages by one time step, that currently is assumed to represent a year. It is possible to note here again that a discrete time is implemented in the program, as well as the discrete space (see also section 5.1).
The second step in the cycle is defined as the spreading operation. Concerning this, every plant searches in their Moore neighborhood for unoccupied cells. Additionally the wind helps to exceed the direct neighborhood. If such cells exist, the conditions for humidity and temperature of these cells will be verified. If the conditions fit the species criteria the seedling establishes. It is possible to set an additional probability value to decrease the seedling likelihood even if the environmental conditions are appropriate for the plant. This procedure concerns every unoccupied cell in the Moore neighborhood and other unoccupied cells by wind spreading. However, the seedling procedure works not completely in this part of the process. Plants on cells which are located at the lattice in a position where the spreading process starts would have an advantage over plants that are located on a later position at the lattice. To synchronize the procedure a *competition list* has been implemented.
The competition list is a kind of stack. Every cell has such a stack and at each time a plant tries to seed on a cell the sprout will be placed in that stack. After the completion of all existing plant propagations the competition list is full. A contest between all members of the competition list determines the successful individual. This plant can occupy the cell and the others die. To find a basis for the contests every species has a specific *competition strength.* The value for *competition strength* is combined with a random value. Of course, if empirical data exists relating the competition between different species this can be represented as a n x n matrix of competition probabilities. Later (see section 6.4.12) it is argued that both possibilities – a matrix for competition between certain species and an individual competition strength – are implemented and it is possible to choose one method. The decision which method should be used depends on the available data about the species.
The chosen implementation of spreading shows that only one plant can occupy a cell. The idea comes directly from the WinklerKlotz Model (see section 3.3.1). This constrains the model to a behavior that is much easier to analyze and fits also to the conception of a discrete spatial model.
The last step of the cycle regarding the individuals is the dying of the plants. If a plant has reached the age which has been calculated for its life expectancy the plant dies. The plant will be erased from the simulation program by canceling

Parameter	Description
Species Number	Unique number which identifies the species.
Species Name	Name of the species, that makes it easier for the user to analyze the output data.
Setup Ratio	The probability for each cell that a species exists on it at the setup of the simulation.
Humidity Mean	The mean of optimal humidity for the species.
Humidity Standard Deviation	The standard deviation of optimal humidity for the species.
Temperature Mean	The mean of optimal temperature for the species.
Temperature Standard Deviation	The standard deviation of optimal temperature for the species.
Maturity Mean	The mean of the age when the species becomes able to breed.
Maturity Standard Deviation	The standard deviation of the age when the species becomes able to breed.
Life Expectancy Mean	The mean age of the species' life expectancy.
Life Expectancy Standard Deviation	The standard deviation of the species' life expectancy.
Competition Strength	The strength of a species when it has to compete with another plant for cell occupation.
Color (RGB)	The Color (divided in red, green blue), that makes it easier for the user to analyze the output data.

Table 5.1: Parameters of species

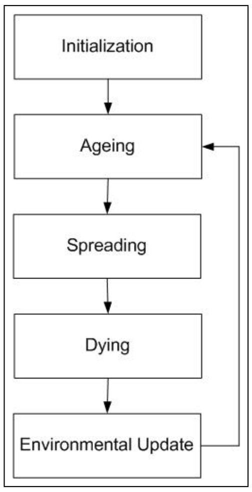


Figure 5.4: Process of the model

the reference from the lattice to the plant[6].
Eventually the update of the environment occurs. In this step of the cycle the temperature or the humidity of the cells can be changed. It is possible to implement all possible settings. Therefore the model provides richer dynamics.
After the update of the environmental conditions the cycle is finished and the simulation program starts a new cycle, starting with the aging of the plants. Figure 5.4 gives an overview of the whole process.

The initial setup is the only possibility for plants to occupy cells which do not fit their environmental needs. The reason is that plants can be seated there by humans or through other external influences. Of course, it is more or less unrealistic that these plants survive for their average life expectancy on these places, but to ignore this fact gives the model's application more dynamics. The user can set the requested *setup ratio* and hence the requested number of plants (see section about parameters (5.2.2)) is applied. Furthermore, we show that this assumption does not lead to differences in the long term distribution (see also section 6.5). Also it is often useful to analyse plants behavior under less than ideal conditions (see the test design in section 6.5).

5.3 User interfaces

To enable an efficient use of the simulation tool a short description of the user interfaces should be given. The interfaces are designed for an intuitive use. However that may be, the major aim of the development is not to create a user friendly program. The target group for the software consists of professional botanical and ecological scientists and therefore a basic understanding can be presumed.

[6]The garbage collector from Java assures that the plants will be destroyed completely, because every object without a reference will be deleted from memory (Samaschke 2004), (Flanagan 2005).

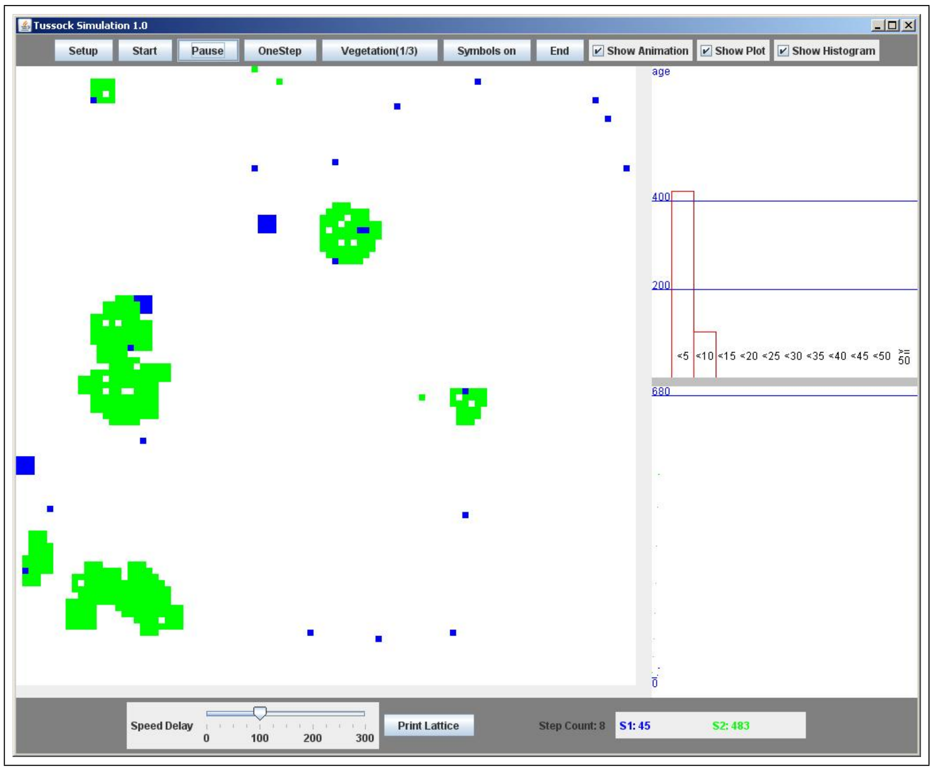


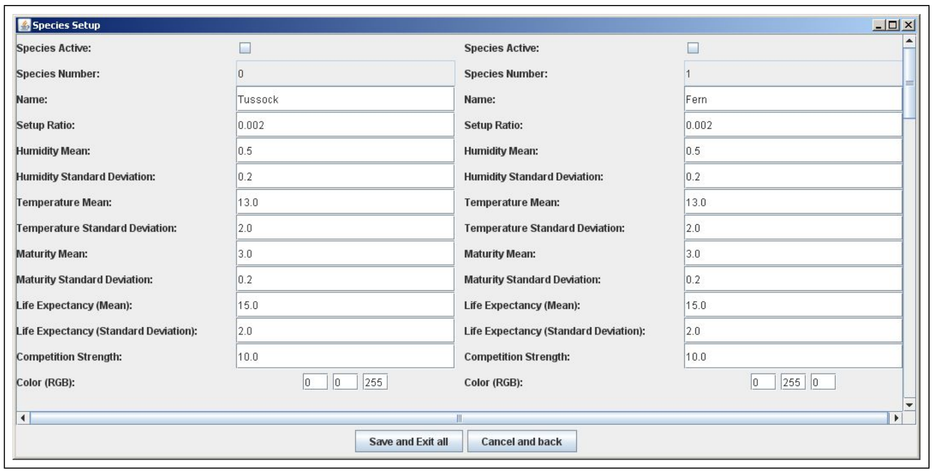
Figure 5.5: Main interface of the simulation software

Basically two graphical interfaces enable the handling without recoding or programming the software. The main interface provides different views and control elements (see figure 5.5).
The functions are ordered in the way the user typically works with the tool:

Setup: The button *Setup* opens a graphical interface. It allows the user to configure the parameters of environment and species (see figure 5.6). Because of the specification of the parameters (in table 5.1) a detailed description is not necessary.
The number of different species is determined by filling out the right quantity of forms. A form must be filled out completely otherwise the species will be discarded.
Scrolling down to the end of the setup view leads to the configuration of the environment. Here it can be decided whether maps for humidity and temperature will be used or random values. The decision for humidity and temperature are independent from each other.

Start: *Start* is the button to run the simulation.

Pause: *Pause* stops the simulation run, but it does not reset the setting, furthermore it freezes every view element and the user can analyze the current



Species Setup

Species Active:	☐	Species Active:	☐
Species Number:	0	Species Number:	1
Name:	Tussock	Name:	Fern
Setup Ratio:	0.002	Setup Ratio:	0.002
Humidity Mean:	0.5	Humidity Mean:	0.5
Humidity Standard Deviation:	0.2	Humidity Standard Deviation:	0.2
Temperature Mean:	13.0	Temperature Mean:	13.0
Temperature Standard Deviation:	2.0	Temperature Standard Deviation:	2.0
Maturity Mean:	3.0	Maturity Mean:	3.0
Maturity Standard Deviation:	0.2	Maturity Standard Deviation:	0.2
Life Expectancy (Mean):	15.0	Life Expectancy (Mean):	15.0
Life Expectancy (Standard Deviation):	2.0	Life Expectancy (Standard Deviation):	2.0
Competition Strength:	10.0	Competition Strength:	10.0
Color (RGB):	0 0 255	Color (RGB):	0 255 0

Figure 5.6: Setup interface of the simulation software

patterns. After that it is possible to continue the simulation with the *Start* button.

OneStep: This function is nothing else than a combination of *Start* and *Pause*. The simulation tool works in a discrete time and *OneStep* executes exact one time step. Of course, it works only when the simulation is not running.

Vegetation, Humidity and Temperature: These three functions are controlled by one button. It changes the content of the main view. Principally the main view shows the vegetation, – that means the plants in different colors placed on the cells – but information about the environmental conditions of the cells are often useful. Therefore it is possible to switch to humidity and to temperature view. The button changes the view from vegetation, to humidity, to temperature, and back to vegetation.

Symbols on, Symbols off: The problem with the humidity and temperature view is, that the user looses the overview whether a plant exists on a cell or not. However, that can be an important information. Furthermore it is useful to distinguish between the different species here. Therefore symbols for each species exist and these symbols can be added with the *Symbols on* function. With the symbols the user can easily recognize where the different plants are, while he still sees the conditions of the environment.

End: This function exits the software.

Show Animation, Show Plot, and Show Histogram: The check box *Show Animation* is switched on in default. Switching off disables the animation that shows the plants on the surface in the main view. The advantage is improved performance. The same function is supported for the plots, and the histogram with the check box *Show Plot*, or *Show Histogram*, respectively.

Speed Delay: The *Speed Delay* slows the running speed down to make the observations of changes easier.

Print Lattice: The *Print Lattice* button triggers a very detailed output. An Excel-file is created that contains information about the whole lattice and the plants. A more detailed description of the file's contents ensues later (see section 5.4).

The Counters: The first counter (*Step Count*) displays the elapsed time in steps. For each complete cycle of the program one step will be added. The other counters represent the amount of species. Firstly, the name of the species is displayed, followed by its amount. Then the second species follows, and so forth. Everything is represented in the color of the species. The species' name with the highest amount of all is displayed in capital letters.

Histogram: To represent the age pattern, a histogram is provided. The depiction is independent from the species and aggregates the plants in age-clusters.

Plots: The plots display the current amount of the plants grouped by species on a time scale and relative to one another.

5.4 Outputs

Apart from the information output inside the main interface – for instance counters and plots – two different outputs are claimable. The first output contains the amount of plants for each species dependent on each time step. This information will be stored automatically in a CSV-file and the syntax for the file's name is *"Output_"(current Year)"-"(current Month)"-"(Current Day)"_"(current Hour)"-"(current Minute)"-"(current Second)".csv"*. In this context *current* means the current time of the computer's clock. Hence, all files are stored on the hard disk in chronological order, if more than one run is executed. The files' content is shown in table 5.2 as a scheme.

"StepCounter:"	1st species Name	...	n-th species Name
"1"	amount of 1st species	...	amount of n-th species
"2"	amount of 1st species	...	amount of n-th species
...	...	...	...

Table 5.2: Scheme of output (CSV-file)

Furthermore a detailed output is possible. By creating an output a trade-off between the degree of information's particularization and the level of information's

aggregation always exists. Aggregation of information has the benefit of easy and fast analyses; but the aggregation's lack of powerfulness is the advantage of particularization. That is the reason for the simplicity of the CSV-output and the additional ability to create a very detailed output when necessary.
As mentioned before the *Print Lattice* function creates an Excel-file. It includes different worksheets and each contains the whole lattice from different view points. For instance the first worksheet contains the information of temperature (the second humidity, the third the species number etcetera). Eventually a sheet with prepared measurements is provided and some statistical measures. However, these statistics are only simple; the user is required to enhance these measurements.
An English installation of Microsoft Excel is recommended. Also the usage of StarOffice has been tested successfully, excepting the prepared measurements.
The syntax for the name of this Excel-file is *"Output_"(current Year)"-"(current Month)"-"(Current Day)"_"(current Hour)"-"(current Minute)"-"(current Second)"_"(current value of the step counter)"_detailed.xls"*. With that name and the name of the more simple CSV-file it is easy to order all files usefully and thereby the files who are belonging to one simulation run are placed together.

5.5 Object-orientated description

Basically there are two reasons for describing the simulation program from an object-oriented view. Firstly, it provides a better understanding of the model. And secondly, – the more important reason, is that – further development of the model is facilitated.
Fourteen classes define the core of the program. Additional packages – like the *JXL-Package* for working with Excel-files, and a package for the gaussian functions – are included.
The most important class is the *TussockSimulator*. It is the center of the program, contains the main view of the simulation program, and it controls a two-dimensional array, which represents the lattice of the cellular automata. The lattice consists of cell. Each cell is an own object and it is occupied by a plant or not. Furthermore the class *Cell* controls the competitions, therefore each cell is associated with one object of the class *CompetitionList*. The competition list only stores all seedlings that want to occupy the cell in the same period. Once the period over, the list will be exhausted, but the object will be reused for the next period.
The cell is also attributed with a temperature and a humidity. The values will be determined by a digital picture of a map or by random values (see also section 5.2.1).
Objects of the class *Species* are individual plants. They store information about the plants preferred humidity and temperature, as well as the life expectancy and the maturity. These values are also determined by a gaussian function (see also section 5.2.2). The current age, of course, is also an attribute of all species objects. Basically all parameter values of the species are part of the correspondent class. Hence the name, the number, the *setup ratio*, the color, and the *competition strength* are part of it.
Setup is the class for the management of the syntax – for the species and the environmental settings – and is sent to an object of *FileOperator*. *FileOperator*

stores and reads the data.
Furthermore the setup consists of n objects from the class *SetupSpeciesControl* and one object from the class *SetupEnvironmentControl*. Therefore n is the number of different species.
The class *PictureView* is used by *Cell* to read the RGB-Code and it gives *Setup* the possibility to represent the maps graphically in the setup view. Additionally *PictureView* is responsible for the scaling of the maps' size to the dimensions of the current lattice.
Each object of the class *SetupSpeciesControl* consists of the graphical representation of one species for the setup mask (see figure 5.6). It also provides the syntax for the data for one species and gives it back to *Setup*.
OutputCreator creates an output into CSV-file (see section 5.4) and stores it on the hard disk. *OutputCreatorDetailed* is responsible for the Excel-file that contains detailed output (see also section 5.4). The function for creating a detailed output can be executed as often as required.
Each of the objects from *Counter* is used to count the amount for one species. This amount is written and read from *TussockSimulator*. The counter for the time steps is directly implemented in the *TussockSimulator*.
The class *Plot* receives information about the amount of plants for each species and creates visual graphs. Only one object of *Plot* exists. The same way as *Plot* works, *Histogram* is implemented, but it processes other information like the age structure.
Figure 5.7 shows the most important parts of the model's object-oriented structure as an UML-diagram.

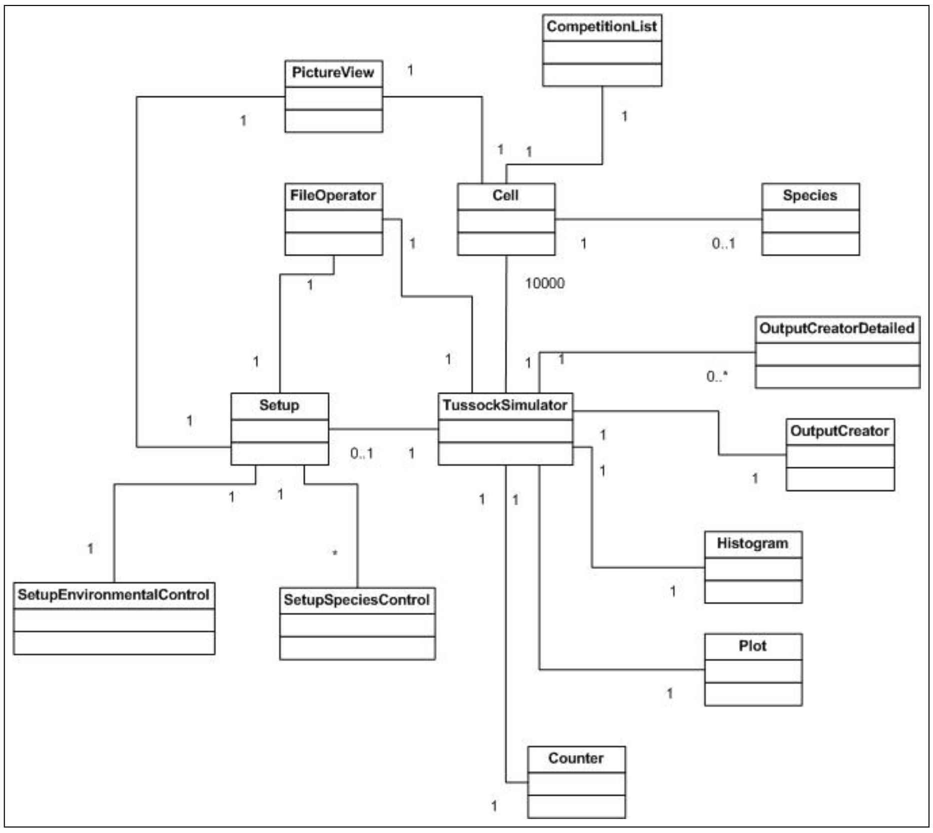


Figure 5.7: Simplified UML-diagram of the simulation program

Chapter 6

Verification of the model

6.1 Introduction to this chapter

One major part of the development process is the verification of the model. The understanding of the results from a simulation model can be grasped as " the real meat of a simulation project" (Kelton 2000, p. 32). Simulation modeling without testing it is a very dubious practice. Results from such a model can never be trusted. Basically two arguments lead to this opinion.
Firstly, an untested program with a certain complexity cannot be presumed as working properly in the way it should. Some behavior is unpredictable, even for the modeler of the program and this is not just a side effect, it is furthermore the reason for simulating. Secondly, the possible results of a simulation program can increase exponentially with its parameters, but often users want to concentrate on only a few combinations. Therefore a plain model without any verification is not very useful for the users. To interpret results they need much more. A well described verification can help. Otherwise it is not possible to distinguish between behavior that is part of model's assumptions or results caused by factors determined by the user.

Species' behavior to different environmental conditions is one major part of the verification. Another aim is to compare different species with each other. After some basic tests, verification of parameters – regarded the environment and species – will be implemented. Each test series will consider a certain parameter (see section 6.4). The first test series is about the *setup ratio* (see section 6.4.1) and it follows after the basic and environmental tests. The goal of the basic and environmental tests is to assure that the underlying properties of the basic setup are correct.
The term "experimental design" indicates that more than one parameter is changed in the relevant test series.

In summary, this chapter should provide two purposes. Firstly, a sensitivity analysis, which evaluates the dependencies inside the model. And secondly, the tests of this chapter should assure that the model works properly.

Parameter	Value
Species Name	TestSpecies1
Setup Ratio	0.002
Humidity Mean	50.0%
Humidity Standard Deviation	5.0%
Temperature Mean	10.4° Celsius
Temperature Standard Deviation	1.0° Celsius
Maturity Mean	4 periods
Maturity Standard Deviation	1 period
Life Expectancy Mean	400 periods
Life Expectancy Standard Deviation	2 periods
Competition Strength	10.0

Table 6.1: Species parameter settings

6.2 Basic tests

The basic tests are designed to show the most simple interactions of one species with environmental parameters. The species is not intended to be a real plant, but the model behavior will allow a verification of the system implementation. To achieve this, a map with a temperature of 10.4° Celsius, and another map with 50% humidity, for each cell of the lattice are created and applied. Table 6.1 shows the species paramaters for this and most of the following tests, but for this, and all other basic tests, the *life expectancy mean* is determined to 200 periods (and not to 400 periods as shown in the table). The *setup ratio* is not applied, but a single entitiy of the species will be placed. Therefore the opportunity to determine fixed amounts and localizations for plants are used.
The results of the first runs were as expected. The propagation of the descendants from the initially entity builds a roundly shaped population island, due to the Moore neighborhood. After 70 time steps a certain area is homogeneously covered with vegetation, that means no unoccupied gaps are inside this cluster. For a graphical impression of the clustered offspring from origin individual see figure 6.1. So ~14.5% of total space is covered with the species.
At almost 200 time steps the plants have taken 100% of the space provided by the model. About this time the first plants have reached their life expectancy and they die. These plants are the oldest and they are located close to the initial plant – or more precisely: These are the initial plants and their first descendants. For that reason some gaps – that means unoccupied cells – appear, but because of the perfect conditions for the species these gaps are refilled directly. By watching the model in elapsing time it seems that these gaps are moving from the *center* to the *border* (see also figure 6.2). This seems to be obvious, by considering the allocation of the plants and their age.

The result of a Monte-Carlo-Simulation with 20 runs is a fully occupied lattice after circa 211 time steps. A graph (figure 6.3) shows the growth with an exponential phase followed by a saturated stage. The saturation is caused by the decreasing amount of unoccupied cells.
By including wind in the model the population grows significantly faster. Also 20 runs are executed. On average, after 37 periods the lattice fully populated.

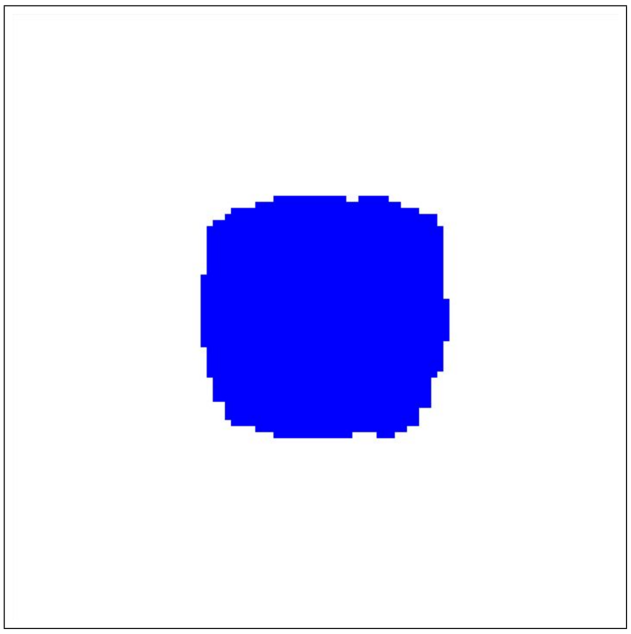


Figure 6.1: Offspring spreading of one plant at time step 70

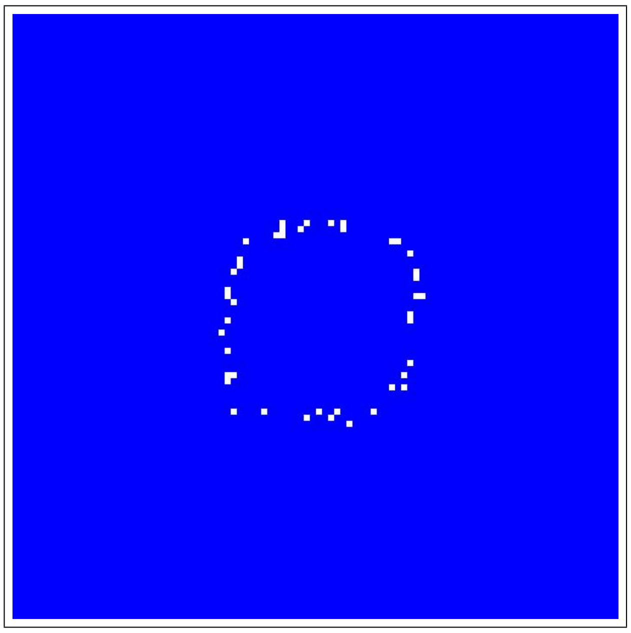


Figure 6.2: Vegetation gaps at time step 503 reasoned due dying of plants

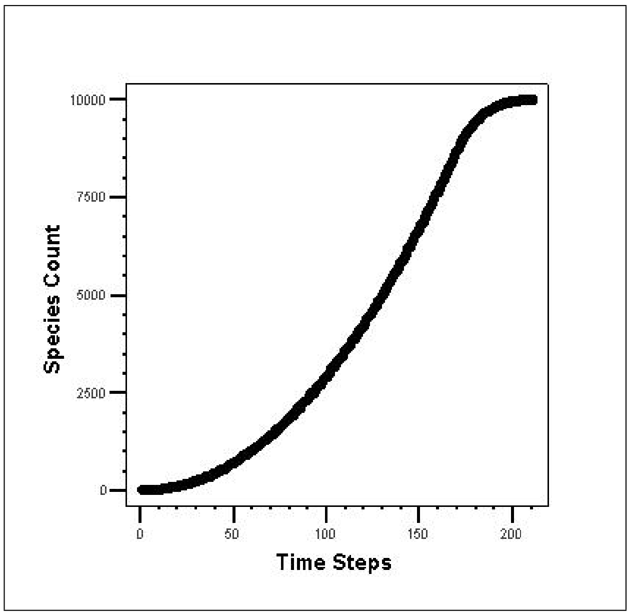


Figure 6.3: Growth of population dependent on time

6.3 Exploration of environmental conditions

First – before exploring the species parameters – it can be useful to change some environmental conditions and therefore increase the difficulty of survival for the current species. The first test includes the application of a temperature map which consists of a life friendly region and some areas with too hot or too cold temperatures for the species. To operationalize this, the most extreme temperatures are chosen ($-2.3°$ and $+23.2°$ Celsius) for the not life friendly areas of the map. The ratio for the amount of species at initialization – the *setup ratio* – is defined to be 10%. So approximately every tenth cell is occupied at time step zero.
The model behaves again without surprises. The plants spread on the cells of the region with the supportive temperature and in the cold and hot areas no establishment of seedlings or long-term survival happens.
Replacing the temperature map with the old one (with optimal conditions for the species) and changing the humidity map to an area with partly extreme conditions (0% and 100%) shows similar results.
Also the combination of maps for extreme temperature and extreme humidity shows that only the areas are populated where both conditions fitting with species needs. That shows that the model satisfies Liebig's Law (see also chapter 4).

A series with 20 repetition of the runs with changed temperature and humidity maps does not falsify that impression.

6.4 Exploration of species parameters

To focus the verification to the species itself it is useful to change every single parameter. The tests will be carried out with respect to the *ceteris paribus clauses* which means that the same conditions for each test will be applied. Therefore, only the value for the examined parameter will be changed. All other values will stay constant (Zahar 2007). With some exceptions the design of the tests will stick to this statement. At each time where a change of more than one parameter, or the species amount is implemented, a comment and a justification will be given.

The basic setting for the parameters of the plants is the – already known – data from table 6.1. In the first test, information of environmental conditions will be determined randomly, so without usage of any maps.

6.4.1 Exploration of the *setup ratio*

The lowest tested *setup ratio* is 0.01%. Lower values often lead to completely empty surfaces at initialization.
In the first run of this experiment random values for temperature and humidity are applied. The setup creates two plants and after a short while (the *maturity mean* is four periods) the plants start to breed. The reproduction leads to three individuals after five periods, four after six periods, and after ten periods to five plants. After 13 periods there are six plants on the surface, and the last breed for the moment happens after 18 periods to the total amount of seven plants. Until the life expectancy of the first plants has not yet lapsed (this happens after about 400 periods) nothing changes regarding the amount of individuals. An observation of this test, is that two *population-islands* arise on the surface. Each individual from the initialization creates one point from where its successors propagate. This concentration is reasoned by the fact that more distant cells are out of range for the successors of the plant[1].
Two things can be recognized as a result. Firstly, it seems that a use of the random values to determine the environmental conditions is improper. It does not support the propagation of the plants and therefore it is hard to estimate how the behavior under other conditions would be. The cells conditions around the islands of plants are not fitting to the plants needs, either temperature is too low or too high or the humidity does not suit. Aside from that it is to assume that these conditions are not very natural. Such heterogeneous conditions seem to be untypical for the small size of the modeled area. Also climate variables are generally smoothly varying. Moreover a regional clustering of environmental conditions is more realistic. Temperatures and humidities are influenced by the neighbor areas. Hence, environmental conditions are not independent from closely located cells.
The second observation is that the low *setup ratio* (accompanied by for the plants survival not supportive environmental conditions) leads to the establishment of islands. These islands are insulated from each other. Death of individuals result in empty cells. These cells will be reoccupied very soon. The reasons are obvious, because of the former occupation the cell must provide a

[1] *Wind* is not part of the test series.

supportive environment for the species needs. And because of the establishment of islands, the likelihood of an occupied neighbor cell is high, the individual of the neighbor cell can be the ancestor of the new plant.
Other tests under the same conditions, but with wind lead also to the establishment of islands, but their quantity is higher. Theoretically every cell can be occupied via wind, no matter how far the distance from the occupied cell is.

The first results lead to an additional test with maps for temperature and humidity. These maps will provide optimal conditions for the current species, homogeneously, that means for all cells. With that experiment it is possible to get results about the speed of propagation.
Again individuals are created for the initialization.
The result is now that two islands with a cyclic shape emerge (similar to figure 6.1).
Measuring the growth's speed shows that it is faster (see figure 6.4) compared to the basic test (see section 6.2, especially figure 6.3). To occupy all cells of the lattice only 175 time steps are necessary.

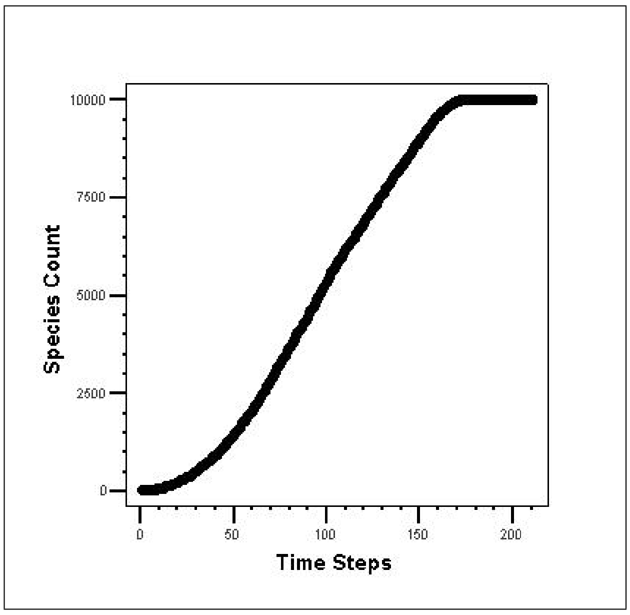


Figure 6.4: Growth of population depending on time

The reason for the increasing speed compared to the basic test (see section 6.2) is obvious. Starting with two plants means double amount of individuals from the basic test with only one plant.
However, the speed is not exactly double. To understand this behavior the position of the two cells must be considered. An assumption – deriving from the observation that the speed of propagation increases with the amount of plants at setup time – is that the farther the both plants are located from each other, the faster the spreading occurs.
To test this assumption two more runs of the model are carried out. Each run under same condition, with two plants at initialization. The difference of both

runs is their location. In the first test both individuals will be placed on two cells next to each other. In the second run both cells will be placed as far as possible from each other. Of course, the torus shape of the lattice is considered). So in a 100 x 100 torus the maximum distance is 50 cells. For instance, the position x=25 and y=25 is in maximum distance to cell x=75 and y=75.
In the first run – where the two cells are neighbors – all cells are occupied after 209 time steps. In the second run – with the maximum distance – only 170 time steps are necessary for the a fully occupation of the lattice. Several repetitions – five for each of both settings – show an average of circa 208 time steps for the setting with neighbored individuals, and circa 171 periods for the plants with the distance in between.
For a more detailed analysis every location between the two extrema – neighborhood and most remote – will be measured. Plant one (P_1) will stay at the same position (x=75 and y=75) during the whole experiment. Plant two (P_2) will be located concerning to the values in the first column (with x=y) of table 6.2.

Position of P_2	run 1	run 2	run 3	run 4	run 5	Average
25	170	170	169	173	172	170.8
26	171	173	173	175	174	173.2
27	173	170	172	169	169	170.6
28	173	170	170	168	171	170.4
29	171	171	172	173	170	171.4
30	172	176	172	173	177	174.0
31	171	172	173	174	172	172.4
32	175	172	171	169	175	172.4
33	176	172	170	174	175	173.4
34	173	174	172	177	174	174.0
35	174	172	172	171	171	172.0
36	173	176	176	169	174	173.6
37	175	173	172	170	172	172.4
38	174	172	175	175	174	174.0
39	174	176	177	178	174	175.8
40	175	177	177	177	173	175.8
41	172	174	172	175	171	172.8
42	176	176	174	178	179	176.6
43	175	180	175	174	175	175.8
44	179	178	177	173	177	176.8
45	174	175	176	181	180	177.2
46	175	173	179	174	174	175.0
47	179	180	172	177	176	176.8
48	177	177	178	179	176	177.4
49	176	178	178	180	181	178.6
50	179	183	177	175	179	178.6
51	183	177	178	181	180	179.8
52	185	179	178	180	180	180.4
53	183	180	177	181	182	180.6
54	183	183	182	181	181	182.0
55	182	182	184	187	177	182.4

56	182	182	183	187	183	183.4
57	186	188	182	186	182	184.8
58	186	185	188	185	186	186.0
59	186	188	187	187	188	187.2
60	191	186	187	190	186	188.0
61	187	188	184	188	190	187.4
62	190	188	191	189	190	189.6
63	195	194	191	190	191	192.2
64	189	192	191	192	189	190.6
65	195	195	194	190	192	193.2
66	194	197	194	192	202	195.8
67	198	194	196	199	194	196.2
68	199	198	198	198	197	198.0
69	197	197	198	196	199	197.4
70	200	196	198	199	198	198.2
71	201	202	203	201	202	201.8
72	202	202	201	201	201	201.4
73	207	206	206	205	209	206.6
74	209	208	206	210	206	207.8

Table 6.2: Growing speed for different positions of two plants

Result, of the more detailed analysis, is that the decrease in time – that is needed for a completely occupation of the lattice – by an increment of distance, is not proportional to the distance of both individuals. With a higher distance the increment of speed per cell declines (see figure 6.5). The limiting value, is of course, zero. That means that the increase of occupation speed is non-negative on each position change of P_2 related to P_1. The volatility in figure 6.5 is reasoned by noise. It is an empirical investigation with only five measurements per setting. With more repetitions the variance would reduce.
The reason for the declining increase with more distance is explicable by thinking about two overlapping circles. Because of the optimal and homogeneous conditions, the plants are spreading in all directions with a very similar speed, therefore we can compare the islands of plants with circles. The amount of overlapping space inside two circles is higher when the centers of the circles are close together(a); the opposite is the case when there is a higher distance (b) (in figure 6.6). Therefore the spreading possibilities are restricted. For an islands of plants it is solely possible to breed at its edge. Shortly after the fusion of both islands happens the growth of total circumference slows down. Hence, the growth of population slows down as well.

The next step will be a switching back to random positions and numbers for the plants at setup time while increasing the value for *setup ratio*.
The *setup ratio* determines the amount of a species at the initialization for a simulation run (see table 5.1). It is the probability for each cell whether a plant will be placed there or not. Table 6.3 shows the average amount of individuals for ten runs at initialization (right column) linked to the *setup ratio* (left column). Again a lattice with 100 x 100 cells is used.

A proportional correlation between the *setup ratio* and total amount of plants

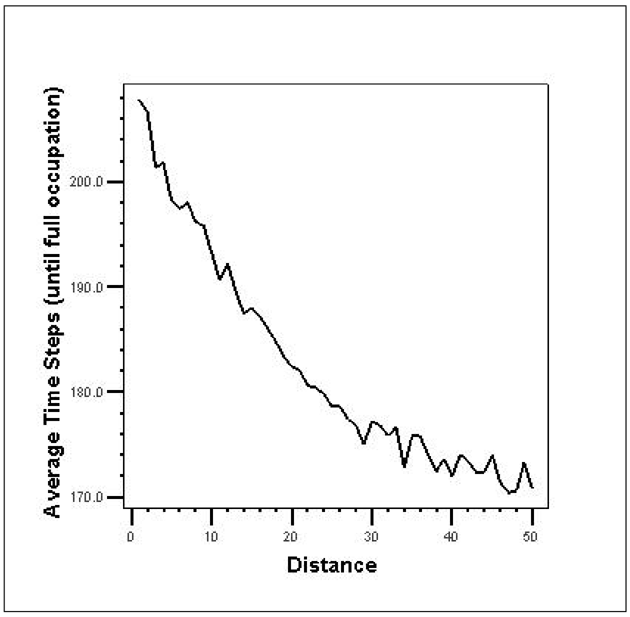


Figure 6.5: Growth of species amount dependent on time

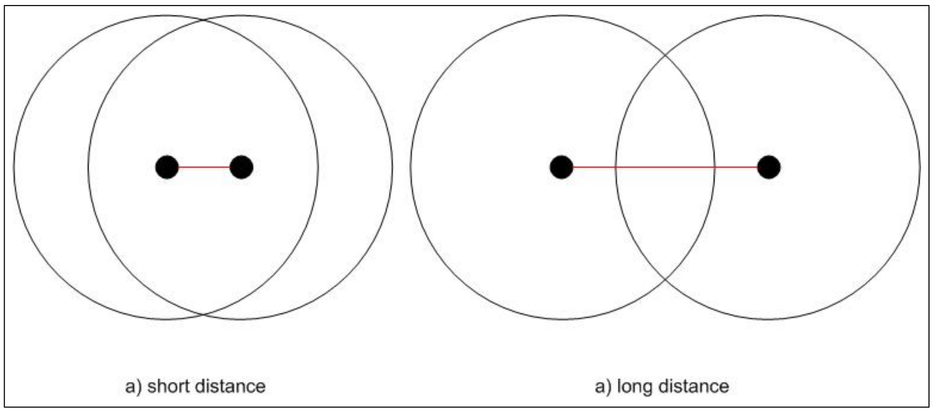


Figure 6.6: Overlapping space correlates with distance

Setup ratio	Average amount
0.001	9.0
0.002	18.3
0.003	32.6
0.004	40.7
0.005	47.2
0.006	62.2
0.007	71.4
0.008	84.2
0.009	89.4
0.010	99.4
0.020	203.5
0.030	301.8
0.040	389.2
0.050	505.9
0.060	605.0
0.070	697.5
0.080	792.3
0.090	904.1
0.100	994.0
0.200	2011.7
0.300	3002.5
0.400	3998.4
0.500	4975.7
0.600	5999.7
0.700	6985.5
0.800	8008.6
0.900	9001.5
1.000	1000.0

Table 6.3: *Setup ratio* of one species correlating with total amount at time step zero

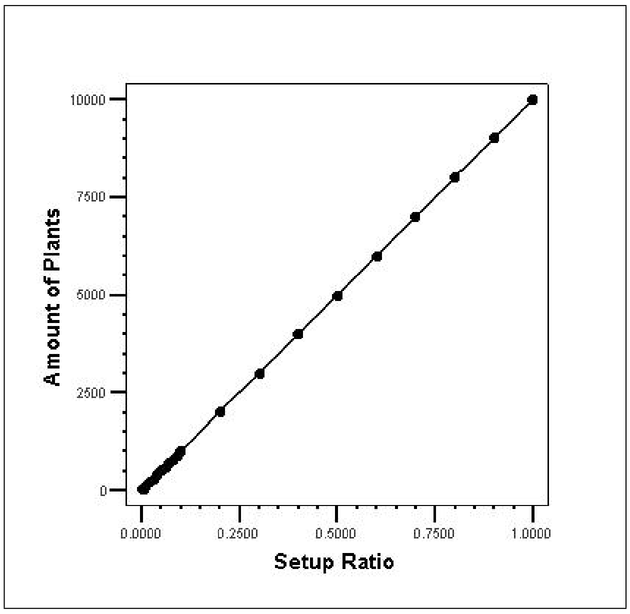


Figure 6.7: Amount of plants depending on *setup ratio* for one species

at initialization of the program is given[2] (see also figure 6.7). The allocation of the plants on the surface seems to be random.
This behavior changes, of course, with more than one species. Competition for cells is the reason. At initialization the same competition rules apply as by competition during a simulation run. To test the dependency of the amount of plants on the number of species, another species (S_2) is added to the model (see table 6.4) and a third one S_3 subsequently (see table 6.6). The new species have the same parameter values (exception is the *setup ratio* according to the tables) as the original one (S_1).

Figure 6.8 shows the declining but positive margin of the average amount of plants for both species by an increase of *setup ratio.* The data is the same as the content out of table 6.4 (except the last two rows where the *setup ratio* from S_1 is not equal to the one of S_1). It is generated from a Monte-Carlo-Simulation experiment with five runs for each configuration of the *setup ratio.* Compared with scenario for one species (see figure 6.7), the dependency between *setup ratio* and amount of plants is now under proportional.
The last two rows of table 6.4 expose a dissimilar value for *setup ratio* for both species. One configuration is S_1 with a ratio of 5% and S_2 with 100%. Its result shows that S_1 has an amount of only 243 plants on average after the initialization of the model. Compared to the setting of the fourteenth row of table 6.4 (where both species have the ratio of 5%) the total amount of plants is lower (than 505 for S_1 and 481 for S_2, repectively).
To explain this result the structure of the *setup ratio* is crucial. The *setup ratio* is not directly oriented to the amount of cells which a species occupies at time

[2]This is actually no property of models behavior, it is more likely to show that the model works correctly.

Setup ratio S_1	*Setup ratio* S_2	Average amount of S_1	Average amount of S_2
0.001	0.001	9.8	10.2
0.002	0.002	20.2	19.4
0.003	0.003	29.2	33.6
0.004	0.004	38.6	43.0
0.005	0.005	52.8	47.2
0.006	0.006	62.8	56.2
0.007	0.007	78.2	75.4
0.008	0.008	75.8	78.6
0.009	0.009	86.4	79.6
0.010	0.010	103.6	100.2
0.020	0.020	192.4	202.2
0.030	0.030	291.8	287.4
0.040	0.040	389.4	386.6
0.050	0.050	505.0	481.0
0.060	0.060	591.8	568.4
0.070	0.070	683.0	680.4
0.080	0.080	749.8	773.4
0.090	0.090	863.4	862.8
0.100	0.100	963.2	950.2
0.200	0.200	1796.6	1806.0
0.300	0.300	2537.6	2551.6
0.400	0.400	3195.6	3214.4
0.500	0.500	3734.0	3773.6
0.600	0.600	4206.2	4181.2
0.700	0.700	4580.2	4524.2
0.800	0.800	4791.2	4805.6
0.900	0.900	4952.8	4946.2
1.000	1.000	4994.4	5005.6
0.050	1.000	243.0	9757.0
0.100	0.200	893.0	1901.8

Table 6.4: *Setup ratios* of two species correlating with total amounts

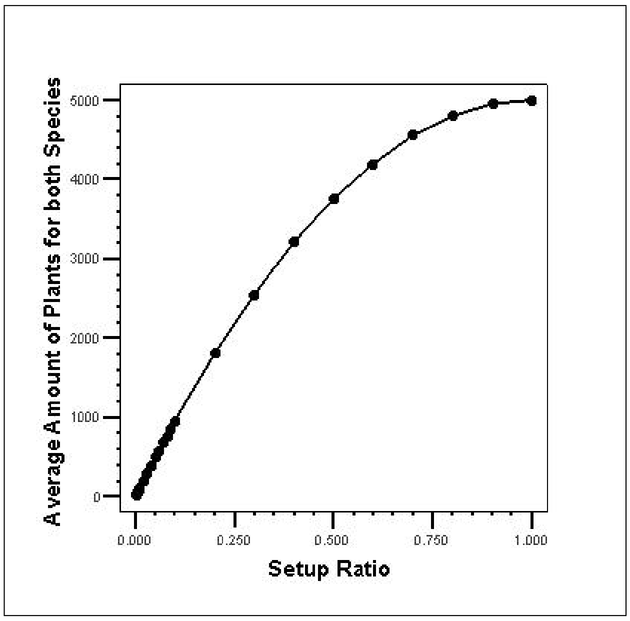


Figure 6.8: Amount of plants depending on *setup ratio* for two species

of initialization. Furthermore it is directed to every single cell. So, some cells are aimed by both species to be occupied and in that case the competition rules are intervening. With low *setup ratios* for both species the intervening of competition rules are rare. However, with an increase of the *setup ratio* else cells are aimed for occupation and therefore more competition happens. More competition leads to more lost competition games. Of course, it leads also to more won competitions games, but these wins are nothing more than what would happen if there is no competitor at all. So it is obvious – and again not surprising – that the total amount of individuals at initialization of a species is not only depending on its own *setup ratio*, but also on the *setup ratios* of all other species. To develop an equation to explain this observation two elements must be considered. Basically the *setup ratio* of the considered species ($\mathrm{S_a(sr)}$) and the probability that the other species $\mathrm{S_b}$ tries to occupy a cell that $\mathrm{S_a}$ also strives for, are crucial. The probability for such an incidence is ($\mathrm{S_a(sr) \cdot S_b(sr)}$). Equation 6.1 follows the presumption that the cases of competition must be subtracted from the basic probability, but only half of it, because half of the competitions wins $\mathrm{S_a}$ on average. A necessary assumption is that the *competition strength* is equal for both species. The terms *width* and *height* mean the width and the height of the whole lattice measured in number of cells.

$$\mathrm{TA(S_a)} = \mathit{with} \cdot \mathit{height} \cdot (\mathrm{S_a(sr)} - \frac{\mathrm{S_a(sr) \cdot S_b(sr)}}{2}) \tag{6.1}$$

Using different values for *competition strengths* is basically not different from the statement of equation 6.1. Only the ratio of $\frac{1}{2}$ must be replaced with ratio of both *competition strengths* (see equation 6.2, with $\mathrm{S_x(cs)}$ as *competition strength*

$S_a(sr)$	$S_b(sr)$	Avg. $TA(S_a)$ measured	$TA(S_a)$ calculated	Relative error
0.200	0.400	1615.2	1600	0.9%
0.300	0.400	2394.2	2400	-0.2%
0.200	0.020	1975.8	1980	-0.2%
0.001	0.200	8.0	9	-12.5%
0.020	0.300	169.4	170	-0.4%
0.070	0.060	662.6	679	-2.5%
0.500	0.200	4498.2	4500	-0.1%
1.000	0.020	9900.2	9900	0.0%
0.800	0.400	6393.6	6400	-0.1%
0.001	0.800	4.8	6	-25.0%
0.100	0.500	762.2	750	1.6%
0.500	0.700	3245	3250	-0.2%

Table 6.5: Comparison of empirical data with results from an equation for *setup ratio* for two species

for any species).

$$\mathrm{TA}(\mathrm{S_a}) = \mathit{with} \cdot \mathit{height} \cdot (\mathrm{S_a(sr)} - \frac{\mathrm{S_a(sr)} \cdot \mathrm{S_b(sr)} \cdot \mathrm{S_a(cs)}}{2 \cdot \mathrm{S_b(cs)}}) \tag{6.2}$$

To see how exact the formula (equation 6.1) works, new empirical values are created by application of the simulation model, and compared with the results of the equation (see table 6.5[3]). The values for the *setup ratios* are chosen randomly, but under consideration of inequality. For the measurement the mean of five single runs is used. Again all other species and environmental parameters are unchanged.
Table 6.5 shows an acceptable accordance of the equation (6.1) with empirical data. Exceptions are low *setup ratios* for one of two species. However, this is reasoned by the fact that for minor values even a small divergence leads to a large relative error.
In summary, it is possible to argue a certain correctness of equation 6.1. However, it can be useful to adapt the equation for the application of more than two species.

Designing an experiment with three species, can help to find out whether a mathematical model can describe the dependency from the *setup ratio* to the total amount of plants, at time step zero. Because of the infinite number of possible combinations, it is necessary to find an acceptable solution for that experiment.
With three species it may be useful to double the *setup ratio* for the second species (S_2) from the first species (S_1) and triple it for the third species (S_3). So the differences between the *setup ratios* are not too small. Table 6.6 shows in the first three columns all applied values and the consequential results, or to be more precise, the average results for five runs. Of course, all species parameters

[3]The abbreviation *sr* stands for *setup ratio*; and *TA*(S_x) means the average total amount of plants for a certain species at initialization.

S_1(sr)	S_2(sr)	S_3(sr)	TA(S_1)	TA(S_2)	TA(S_3)
0.001	0.002	0.003	9.6	17.6	30.6
0.002	0.004	0.006	19.4	38.0	58.4
0.010	0.020	0.030	95.6	209.8	293.2
0.020	0.040	0.060	189.2	415.2	563.0
0.100	0.200	0.300	759.8	1841.2	2566.8
0.200	0.400	0.600	1101.0	2903.2	4444.0
0.015	0.030	0.045	137.0	324.2	444.0
0.150	0.300	0.450	973.6	2429.2	3606.6
0.030	0.060	0.090	278.8	629.0	844.2
0.300	0.600	0.900	1158.8	3423.2	5736.6
0.033	0.066	0.099	304.6	692.2	951.2
0.333	0.666	0.999	1109.2	3480.0	6102.4

Table 6.6: *Setup ratio* correlation with total amount of plants for three species

are the same (except the *setup ratio*). The environmental parameters are also unchanged.
The supposition is that the total amount of a species depends on the *setup ratios* of all species and the size of the lattice. This is not new, but important is how the dependency between all *setup ratios* works. Equation 6.3 shows an approach to model it mathematically and it is basically reasoned on the equation for two plants (equation 6.1). (The term *size* means only *width* · *height* and stands again for the dimension of the lattice. It better fits into the length of the equation.)

$$\mathrm{TA}(\mathrm{S_a}) = \mathit{size} \cdot (\mathrm{S_a}(\mathrm{sr}) - \frac{\mathrm{S_a}(\mathrm{sr}) \cdot \mathrm{S_b}(\mathrm{sr})}{2} - \frac{\mathrm{S_a}(\mathrm{sr}) \cdot \mathrm{S_c}(\mathrm{sr})}{2} + \frac{\mathrm{S_a}(\mathrm{sr}) \cdot \mathrm{S_b}(\mathrm{sr}) \cdot \mathrm{S_c}(\mathrm{sr})}{3}) \tag{6.3}$$

Equation 6.3 is very similar to equation 6.1. The difference is the consideration of two competitors, therefore the probability for competition with all other species is minded ($\frac{\mathrm{S_a}(\mathrm{sr}) \cdot \mathrm{S_b}(\mathrm{sr})}{2}$ and $\frac{\mathrm{S_a}(\mathrm{sr}) \cdot \mathrm{S_c}(\mathrm{sr})}{2}$). It is possible to shorten equation 6.3 to equation 6.4 and argue the last part of it.

$$\mathrm{TA}(\mathrm{S_a}) = \mathit{size} \cdot (\mathrm{S_a}(\mathrm{sr}) - \frac{\mathrm{S_a}(\mathrm{sr}) \cdot (\mathrm{S_b}(\mathrm{sr}) + \mathrm{S_c}(\mathrm{sr}))}{2} + \frac{\mathrm{S_a}(\mathrm{sr}) \cdot \mathrm{S_b}(\mathrm{sr}) \cdot \mathrm{S_c}(\mathrm{sr})}{3}) \tag{6.4}$$

The last part ($\frac{\mathrm{S_a}(\mathrm{sr}) \cdot \mathrm{S_b}(\mathrm{sr}) \cdot \mathrm{S_c}(\mathrm{sr})}{3}$) possibly seems to be against intuition. It has to do with the competition game of plants. If plants of three different species are trying to occupy one cell then the chances for each of the species are basically the same. The competition game does not work the way that the first plant has to compete against the second one, and the winner of this game has to prevail over the third one. This would be an advantage of the third, because it would have to fight only once. So the implemented algorithm assures the same chances in one game for all (in the current case three). However, an equation without adding the shared set of all *setup ratios*, must be wrong – for more than two species – , because it would assume that cells that are sought by two other species (out of the view of the considered species ($\mathrm{S_a}$(sr)) triggering two separate competition games.

$S_a(sr)$	$S_b(sr)$	$S_c(sr)$	$TA(S_a)$ m.	$TA(S_a)$ c.	Rel. error
0.001	0.002	0.003	30.6	29.95502	2.1%
0.002	0.004	0.006	58.4	59.82016	-2.4%
0.010	0.020	0.030	293.2	295.52000	-0.8%
0.020	0.040	0.060	563.0	582.16000	-3.4%
0.100	0.200	0.300	2566.8	2570.0000	-0.1%
0.200	0.400	0.600	4444.0	4360.00000	1.9%
0.015	0.030	0.045	444.0	439.94250	0.9%
0.150	0.300	0.450	3606.6	3555.00000	1.4%
0.030	0.060	0.090	844.2	860.04000	-1.9%
0.300	0.600	0.900	5736.6	5490.00000	4.3%
0.033	0.066	0.099	951.2	941.71374	1.0%
0.333	0.666	0.999	6102.4	5738.51574	6.0%

Table 6.7: Comparison of empirical data with equation calculations for three species

The exact structure of the added part ($\frac{S_a(sr) \cdot S_b(sr) \cdot S_c(sr)}{3}$) is designated because of the fact that the probability of all cases, where three species are trying to occupy the same cell must be treated separately. Therefore the term $S_a(sr) \cdot S_b(sr) \cdot S_c(sr)$ – which expresses exact this probability – is added, or rather: Is not subtracted from the *setup ratio* of the considered species. (All probabilities of occupation attempts of different species are subtracted from the *setup ratio* (see equation 6.3 or 6.4).) Thereafter the probability for losing competition games with three species participated must be subtracted. The term $\frac{2 \cdot (S_a(sr) \cdot S_b(sr) \cdot S_c(sr)}{3}$ expresses that. Eventually, the summarizing calculation leads to the added term: $S_a(sr) \cdot S_b(sr) \cdot S_c(sr) - \frac{2 \cdot (S_a(sr) \cdot S_b(sr) \cdot S_c(sr)}{3} = \frac{S_a(sr) \cdot S_b(sr) \cdot S_c(sr)}{3}$.
Of course equation 6.4 also based on the assumption of equal values for *competitions strengths.*

An experiment should test equation 6.4 empirically. Again everything is unchanged from the beginning of this chapter. The exception is the use of three equate species, but with dissimilar *setup ratios.* Table 6.7[4] illustrates the results with a measurement of errors. The empirical data is the same as in table 6.6. The considered species is S_3 ($S_a = S_3$).
The results are acceptable. Of course, it is impossible to find an exact method, which predicts the total amount of plants at initialization of the program by using randomly influenced values like the *setup ratio.* Therefore the equations for two and three species (equation 6.1 and 6.4) provide a useful heuristic for the user and show that the model does not behave unpredictably because of software errors, programming mistakes or the like.

The last step related to the *setup ratio* should be a generalization of equation 6.4 for any species numbers. The model basically provides an infinite number of species. The models application in practice with higher amount of species may be rare. However, a heuristic can be useful.
Cases for more than three species are just a generalization of the three species

[4] "$TA(S_a)$ m." stands for the average total amount of $Species_a$ measured and "$TA(S_a)$ c." is the calculated result.

case. Therefore an abstract expression of equation 6.4 with k species is equation 6.5.

$$\mathrm{TA}(\mathrm{S_a}) = size \cdot (\mathrm{S_a}(\mathrm{sr}) - \sum_{i=2}^{k} \frac{\mathrm{S_a}(\mathrm{sr}) \cdot \mathrm{S_i}(\mathrm{sr})}{2} + \frac{\mathrm{S_a}(\mathrm{sr})}{k} \cdot \prod_{i=2}^{k} \mathrm{S_i}(\mathrm{sr})) \tag{6.5}$$

6.4.2 Exploration of the *humidity mean*

Humidity mean is the next parameter to test. It determines the optimal humidity for an individual of a species (see table 5.1). For these tests it is possible to speculate that a change of the mean at constant environmental conditions[5] leads to decreasing breeding abilities of the species. At the moment the *humidity mean* of the species is 50% (see table 6.1). The humidity map determines the same value to all cells.
A basic assumption of the model is the existence of a certain degree individuality for the plants. Each species has only a *mean* and *standard derivation* for *humidity* and *temperature*. The same is true for *life expectation* and *maturity*. On level of entity, concrete values for theses parameters are calculated. That means as soon as a seedling reaches a cell, the *birth* happens and an exact life expectancy, maturity, preferential temperature, and – needless to say – also a preferential humidity will be determined. This happens even before the competition for the cell starts. This assumption is based on the point of view that each individual plant is unique and slightly different from other entities of its species[6]. Reason can be mutation or similar effects that differentiate individuals. So a basic rule for propagation is that the characteristics of the parent plant determine whether a cell is eligible for its successors or not.
An experiment is designed to figure out what relationship between propagation and *humidity mean* exists. It is predictable that not all runs of the experiment will lead to a full occupation of the whole lattice. Therefore the dynamics over a certain amount of time will be measured. The *humidity mean* will be changed in steps of 5%. All other conditions and parameters are unchanged (see table 6.1). Also the *humdity standard deviation* is still 0.05%. The time step when the measurement happens is 50. For each setting of the parameter, ten runs are carried out. The results of this experiment are shown in table 6.8.
A verification, based only on the percentage from time step zero to 50 show that the number of plants increase, however this is slightly confusing (see right column in table 6.8). This is reasoned by the fact that the amount of plants at time step zero is crucial for the resulting percentage. For instance at a *humidity mean* of 50% a relative high average amount of plants initially occurs. This circumstance has a strong influence on the percentage of increase. For the mathmatical result it is crucial, but it does not show the model's behavior regarding the change of *humidity mean* without bias. Hence, it is useful to consult the absolute values at time step 50 to get an overview of the influence (see figure 6.9).
The absolute numbers are more realistic for the purpose of that experiment. Alternatively it could be useful to repeat the experiment with fixed numbers of plants at the model's initialization. However, this would be an intervention in

[5]The map for humidity is still the same (and also the temperature map).
[6]By discussing the *standard deviations* for *humidity*, *temperature*, *maturity*, and *life expectancy* it becomes clear how the user can lever out this assumption.

Humidity mean	TA(S_a) at t=0	TA(S_a) at t=50	increase
0%	21.6	21.6	0.0%
5%	19.2	19.2	0.0%
10%	21.0	21.0	0.0%
15%	16.8	16.8	0.0%
20%	19.5	24.3	24.6%
25%	20.5	142.6	595.6%
30%	17.4	3286.9	18790.2%
35%	19.1	6799.1	35497.4%
40%	19.2	7444.1	38671.4%
45%	19.7	7692.3	38947.2%
50%	22.3	8113.2	36282.1%
55%	20.5	7766.2	37783.9%
60%	19.3	7176.4	37083.4%
65%	19.5	6777.3	34655.4%
70%	19.4	1647.5	8392.3%
75%	18.8	69.7	270.7%
80%	20.8	22.4	7.7%
85%	21.2	22.0	3.8%
90%	19.5	19.5	0.0%
95%	20.4	20.4	0.0%
100%	18.5	18.5	0.0%

Table 6.8: Results of a changing *humidity mean*

the basic settings of the model without a big advantage. The repetition of ten times for each run should be enough to avoid volatility based on noise.
The plant's tolerance regarding humidity is ±20%. The data from table 6.8 (see also figure 6.9) shows a strong reproduction exaclty for values inside the tolerance interval. Therefore the model behaves appropriately regarding the *humdity mean*.
It is certainly possible to change this tolerance interval in both directions, broadening or reducing. For the experiments, however, it can be useful to leave it as it is for the moment.
The sharp rise from 25% to 30% is surely also dependend on the *humdidty standard deviation* of species. A higher standard deviation would probably dimish this rise. This is the next part of the verification.

6.4.3 Exploration of the *humidity standard deviation*

The *humidity standard deviation* influences the deviation of the individual plants preferred humidity. While a high *humidity standard deviation* leads to more individuality of a species entities, results a low deviation in more unitary plants of a species. The assumption that each plant is slightly unique can be leveraged out by a value of zero. If the *humidity standard deviation* is zero, the plant's value for optimal humidity is exactly equal to the *humidity mean*.
By testing the *humidity standard deviation* a concentration on relative high values combined with *borderline* values for *humidity mean* seems to be the most interesting experimental design. Borderline values for *humidity mean* are values

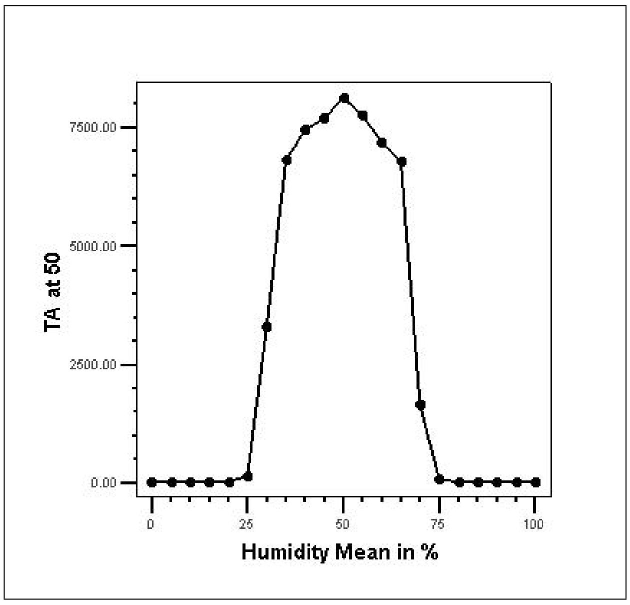


Figure 6.9: Average amount at time step 50 depending on *humidity mean*

that permit both, entities that are unable to breed as well as individual plants that are capable of breeding, because of their humidity preferences. In the next section (6.4.4) a combination of both parameters (*humidity mean* and *humidity standard deviation*) will be implemented.
However, the current experiment is only concerned with *humidity standard deviation*. Basically there is no upper limit for the preferred humidity.
Table 6.9 shows the results of the experiment, that consists again of ten runs for each parameter setting. The main criteria, to estimate the influence of *humidity standard deviation* to the model's behavior, is again (see section 6.4.2) the amount after 50 time steps. The *humidity mean* is constant by 50% during the whole experiment and all other parameters are unchanged from the basic configuration, that means, in particular, the humidity map determines the environment to 50% humidity for all cells. Because of the fact that it does not matter which *standard deviation* for *humdity* is chosen, there is always a probability for an optimal humidity fitting to the environmental conditions. The spectrum of tested values is 0% to 120%.
Figure 6.10 shows a decreasing total amount of plants by an increase of *humidity standard deviation*, and therefore the model behaves as expected. The prediction from the last section (6.4.2) that the sharp rises in figure 6.9 will flatten is demonstrated by this experiment. An increasing standard deviation leads to a more wide spread of the values for optimal humidity. Therefore less plants are characterized with value inside the optimal spectrum.
Another observation is shown at the leveling off of the graph, at about 45% and more for *humidity standard deviation* (see figure 6.10). This may be understood by the fact that the flatten of the gauss function itself. The higher the standard deviation the flatter the graph. The increase of the amount of plants that does not fit into the environmental conditions is almost saturated at this point.

Humidity standard deviation	TA(S_a) at t=0	TA(S_a) at t=50	increase
0%	21.7	7981.5	36681.1%
5%	20.4	7458.1	36459.3%
10%	20.1	7394.7	36689.6%
15%	20.5	6625.5	32219.5%
20%	19.7	4990.1	25230.5%
25%	19.9	3546.6	17722.1%
30%	21.0	2362.9	11151.9%
35%	20.4	1953.8	9477.5%
40%	20.6	1085.6	5169.9%
45%	19.8	525.9	2556.1%
50%	20.2	443.7	2096.5%
55%	20.0	325.1	1525.5%
60%	20.0	206.1	930.5%
65%	19.6	140.3	615.8%
70%	20.2	137.4	580.2%
75%	19.7	93.3	373.6%
80%	19.6	89.8	358.2%
85%	19.7	92.0	367.0%
90%	19.4	92.1	374.7%
95%	20.2	98.1	385.6%
100%	20.2	74.0	266.3%
105%	20.2	66.5	229.2%
110%	19.9	65.6	229.6%
115%	20.3	56.7	179.3%
120%	20.1	56.0	178.6%

Table 6.9: Results of a changing *humidity standard deviation*

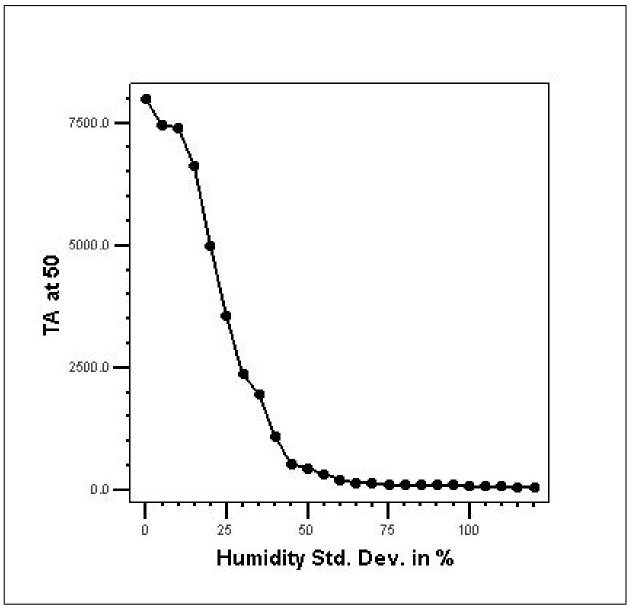


Figure 6.10: Average amount at time step 50 depending on *humidity standard deviation*

With support of the user interface – that allows the observation of the plants on the cellular automata's lattice – it is possible to notice which plants have the right preferred humidity to breed. These plants breed, and islands arise. Others with unfitting preferred humidity cannot reproduce and are isolated until they die (see figure 6.11).

6.4.4 Experimental design: *Humidity mean* and *humidity standard deviation*

By doing experiments with optimal humidity an opportunity is given to test how accurate the gaussian function works. Therefore five runs with a *humidity mean* of 50% and a *humidity standard deviation* of 0.1% are carried out. All of the five runs show at measure time a fully occupied surface. So data of 10,000 entities per run is available. The average of all plants humidity is 0.500174687% and the standard deviation amounts 0.099795991% this means a divergence of only 0.02% for each of both values.

More interesting are experiments that – as mentioned before (see section 6.4.3) – combine values for *humidity mean*, with an almost equal likelihood for fitting and unfitting plants, and a high *humidity standard deviation.* A fast occupation of the "fittest" entities is to presume, but the optimal humidity for a plant is determined individually, so there will occur delays. Some plants will not succeed when they try to spread to neighbor cells, but possibly successors from more remote plants may vegetate them after a while.
The parameter values regarding the preferred humidity for one species are shown in table 6.10. Each line is one test. All other parameter values are unchanged. The results of the first test of table 6.10 are relatively unspectacular. Only one

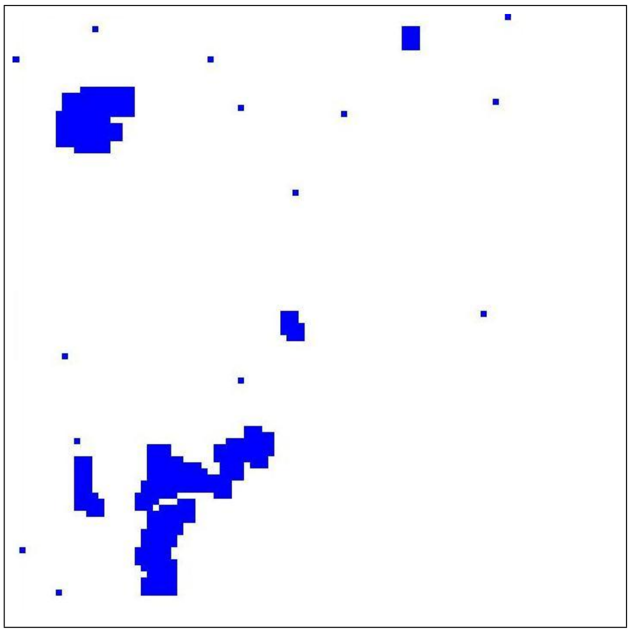


Figure 6.11: Different propagation behavior by a *humidity standard deviation* of 40%

Test	*humidity mean*	*humidity standard deviation*
1	70%	100%
2	70%	20%
3	30%	100%
4	30%	20%

Table 6.10: Parameter values for testing behavior regarding the preferred humidity

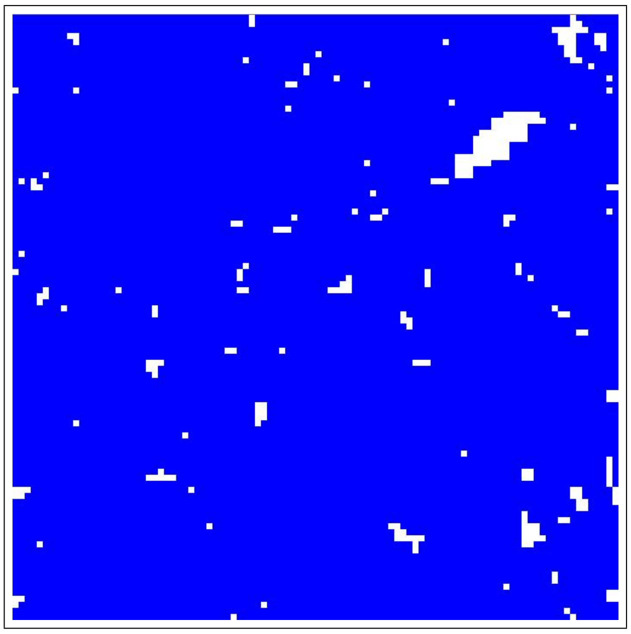


Figure 6.12: Test two at time step 311

plant can breed and occupies its Moore neighborhood completely in one time step, thereupon nothing more happens, except for the dying of all plants at about time step 400.
Test number two is more interesting, because of the smaller standard deviation more plants are able to breed. Growth happens relatively fast. However, a few cells remain unoccupied (see figure 6.12).
Directly after the death of some plants new occupations happen. The reason is that some of the plants that are close to the cells, formerly occupied, are likely to have the right humidity preferences to breed. Consequently they can reoccupy these cells. Furthermore, they and their successors have the chance to occupy the cells that had been free for the whole time. Some changes are visible on the visualization of the lattice (see figure 6.13). Because of relative low *standard deviation* for *life expectancy* – with a value of just two periods – plenty of plants die at the same – or almost at the same – time.
If a plant, which is located directly to an area of unoccupied cells, dies, and its cell will be reoccupied by a new plant, then it is also very likely that parts of – or even the whole – unoccupied area will be vegetated. This has been mentioned before and it is reasoned by another – possibly fitter – plant's humidity preference that is now located directly to the unoccupied cells. The better fitting individuals succeed. Comparing the figures 6.12 and 6.13 shows that some of the unoccupied areas have been vegetated, but that happens not until the first plants have died. Only the reoccupation of formerly vegetated cells leads to new individuals and these new plants can fit into the environment.
However, this is a slow process. The *mean life expectancy* is 400 periods, therefore it takes about 400 time steps until a new generation of individuals comes. Even after 7543 time steps the lattice is not fully occupied by vegetation (see figure 6.14).

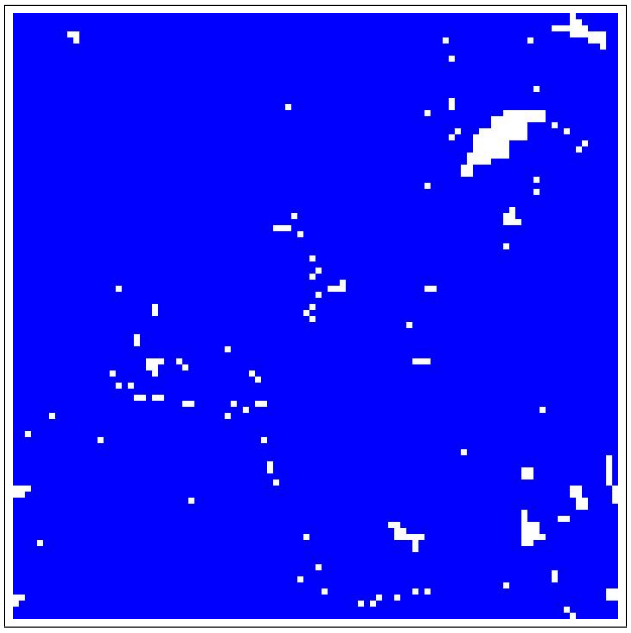


Figure 6.13: Test two at time step 564

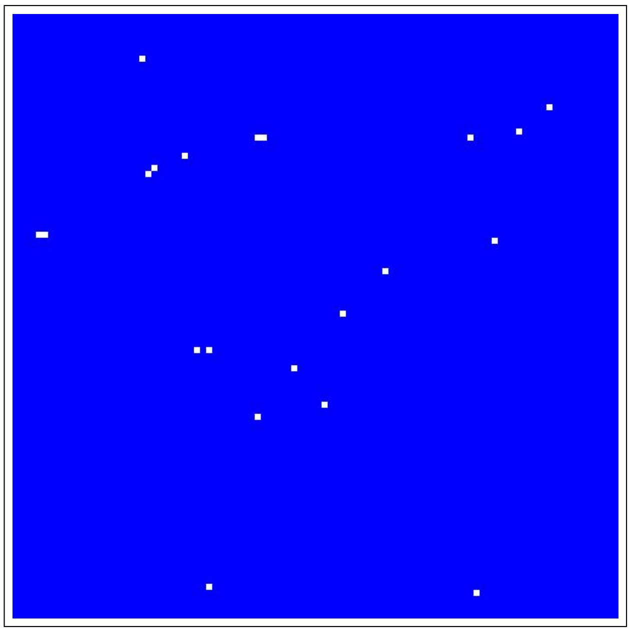


Figure 6.14: Test two at time step 7543

Test three shows similar results as test one and the same is true for test four and two.

6.4.5 Exploration of the *temperature mean*

Temperature – by its similar character and definition – behaves like humidity. Both are deviated according to the gaussian function. Humidity and temperature are combined regarding Liebig's Law. That means, as mentioned before that they are in a non-substitutive relationship.
By verifying the temperature, similar experimental designs as by humidity can be applied.
The individuality for the plants preferences for temperature are basically the same as for humidity. Also a *temperature mean* and a *temperature standard deviation* can be determined for each species. Based on both parameters an optimal (or preferred) temperature is calculated at initialization of the plant (see also section 6.4.2).
The map that determines the environmental temperature is still the same as before. It appoints a temperature of 10.4° Celsius homogeneously to all cells.

To test the dependency between *temperature mean* and the propagation, the temperature is changed in 1.0° Celsius steps[7]. The other conditions remain constant, particularly the *temperature standard deviation* is still 1.0° Celsius. Five runs for each setting are carried out and the times of measurement are time step zero and 50. The criteria is the amount of plants at these times, in particular at time step 50.
The tolerance for temperature swings is currently – and also in all other tests – ±4° Celsius. The measured deviation fits into that interval (see figure 6.15). From about 6° Celsius to about 15° Celsius for *temperature mean* the plants are able to breed.

6.4.6 Exploration of the *temperature standard deviation*

The *temperature standard deviation* determines the plants preferred temperature in the same way as the *humidity standard deviation* influences the optimal humidity. Therefore the test design is similar to that in section 6.4.3. The experiment consists of 29 different settings and each setting is repeated five times. The *temperature standard deviation* changes in steps of 0.1° Celsius. After 21 different settings the step is increased to 0.5° Celsius and eight more values are tested (see table 6.12). The average amount of plants out of the five runs is measured at time step zero and 50. All parameters have the values of table 6.1 – except, of course – the *temperature standard deviation.*
The results of the test for the *temperature standard deviation* show that the breeding capabilities of the species depends on the standard deviation. However, the dependency is not as strong as species breeding depends on *humidity standard deviation.* That is probably reasoned by the rather big tolerance interval of ±4° Celsius. Figure 6.16 shows a quite successful behavior, measured by amount of plants at time step 50, until a *temperature standard deviation* of 5.5° Celsius. Therefore the question comes up whether this tolerance interval

[7]The highest temperature value (see last row of table 6.11) of 23.2° Celsius is an exception of this stepping, reasoned by the limited spectrum for temperature.

Temperature mean	TA(S_a) at t=0	TA(S_a) at t=50	increase
−2.3° Celsius	20.0	20.0	0.0%
−1.3° Celsius	20.2	20.2	0.0%
−0.3° Celsius	21.4	21.4	0.0%
0.7° Celsius	19.8	19.8	0.0%
1.7° Celsius	20.4	20.4	0.0%
2.7° Celsius	21.0	21.0	0.0%
3.7° Celsius	21.0	22.6	7.6%
4.7° Celsius	19.8	29.4	48.5%
5.7° Celsius	19.8	159.6	706.1%
6.7° Celsius	19.6	4097.2	20804.1%
7.7° Celsius	19.8	7088.2	35699.0%
8.7° Celsius	20.6	7533.6	36470.9%
9.7° Celsius	20.2	7572.8	37389.1%
10.7° Celsius	20.6	7804.0	377.8%
11.7° Celsius	20.2	7792.0	38474.2%
12.7° Celsius	20.2	7670.8	37874.3%
13.7° Celsius	19.8	6118.4	30801.0%
14.7° Celsius	20.2	1400.2	6831.7%
15.7° Celsius	20.4	57.0	179.4%
16.7° Celsius	20.4	22.0	7.9%
17.7° Celsius	20.2	20.2	0.0%
18.7° Celsius	19.8	19.8	0.0%
19.7° Celsius	20.0	20.0	0.0%
20.7° Celsius	19.4	19.4	0.0%
21.7° Celsius	19.8	19.8	0.0%
22.7° Celsius	19.6	19.6	0.0%
23.2° Celsius	20.2	20.2	0.0%

Table 6.11: Results of a changing *temperature mean*

Temperature standard deviation	TA(S_a) at t=0	TA(S_a) at t=50	increase
0.0°	20.6	7686.6	37213.6%
0.1°	20.4	7698.4	37637.2%
0.2°	20.4	7644.8	37374.5%
0.3°	20.2	7440.6	36734.7%
0.4°	20.2	7937.2	39193.1%
0.5°	20.4	7845.4	38357.8%
0.6°	20.2	7621.0	37627.7%
0.7°	20.2	7769.2	38361.4%
0.8°	19.4	7411.4	38103.1%
0.9°	20.0	7659.0	38195.0%
1.0°	20.0	7708.0	38440.0%
1.1°	19.8	8018.4	40397.0%
1.2°	19.8	7699.2	38784.9%
1.3°	20.2	7777.0	38400.0%
1.4°	20.4	7230.2	35342.2%
1.5°	19.8	7558.4	38073.7%
1.6°	20.0	7080.2	35301.0%
1.7°	19.8	7501.8	37787.9%
1.8°	19.6	7266.2	36972.5%
1.9°	20.0	7632.0	38060.0%
2.0°	20.2	7089.6	34997.0%
2.5°	20.2	7002.8	34567.3%
3.0°	20.0	6438.0	32090.0%
3.5°	20.4	5947.6	29054.9%
4.0°	19.8	5124.2	25779.8%
4.5°	20.0	4239.2	21096.0%
5.0°	19.8	4058.4	20397.0%
5.5°	19.8	3583.8	18000.0%
6.0°	19.8	2399.4	12018.2%

Table 6.12: Results of a changing *temperature standard deviation*

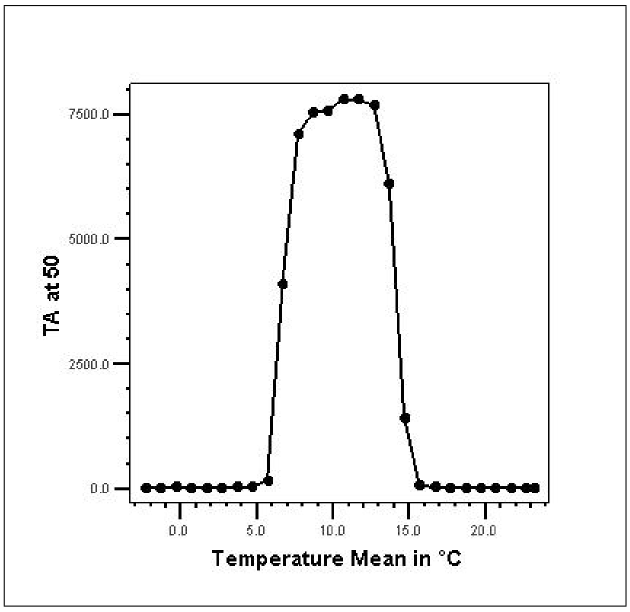


Figure 6.15: Average amount at time step 50 depending on *temperature mean*

is close to reality or not. This can only be answered for real existing and well researched plant species.
The volatile part in figure 6.16 is more reasoned by random influences than by the change of *temperature standard deviation.*

6.4.7 Experimental design: *Temperature mean* and *temperature standard deviation*

The *temperature standard deviation* works similar to the *humidity standard deviation.* A high standard deviation leads to more individuality. It implies that the likelihood for the survival of a plant species is higher if its *temperature mean* does not fit to the environmental temperature. On the other hand, a low *temperature standard deviation* leads to better survival chances if it is combined with a fitting *temperature mean.* For instance a *temperature mean* of 10.4° Celsius is combined with the current map optimal, but a very high *standard deviation* for *temperature* can scatter the value for preferred temperature and a few plants are not able to breed in that environment. Of course, in nature it is not very likely that plants from one generation to the next change significantly, but for the verification it can be an interesting point.
Therefore a test is realized. A *temperature mean* of 10.4° Celsius and a *temperature standard deviation* of 50° Celsius are applied. All other parameter values are again unchanged, except for the *setup ratio.* The *setup ratio* is 1.0 for this experiment. The reason for choosing the maximum *setup ratio* is that the analysis can be conducted without running the simulation program. The initialization is enough. It is very likely that the species is not very successful in breeding with those parameters, but it is useful to get as many plants as possible. For summary of the important parameter values see test one in table

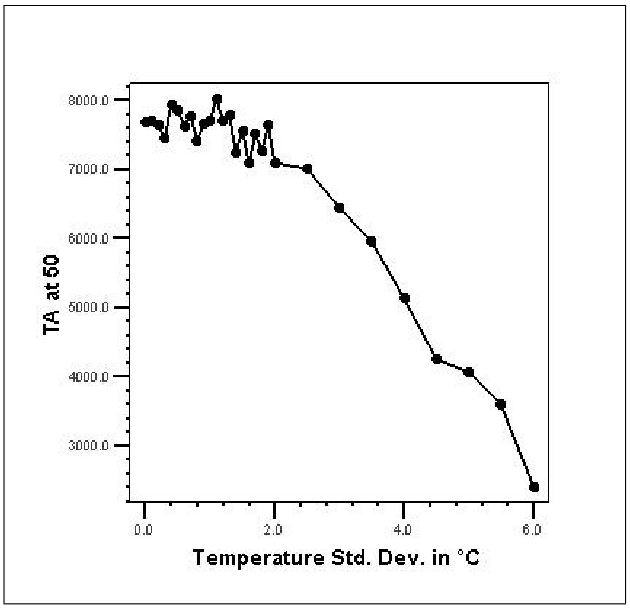


Figure 6.16: Average amount at time step 50 depending on *temperature standard deviation*

Test	*humidity mean*	*humidity standard deviation*	*setup ratio*
1	10.4° Celsius	50.0° Celsius	1.00
2	14.4° Celsius	0.0° Celsius	0.05
3	6.4° Celsius	0.0° Celsius	0.05
4	14.4° Celsius	0.1° Celsius	0.01

Table 6.13: Parameter values for testing behavior regarding the preferred temperature

6.13.
Another test is a combination of the optimal temperature (on the basis of the temperature of the environment, that is 10.4° Celsius) plus the tolerance interval (that is ±4° Celsius) and a low *temperature standard deviation*, for example 0° Celsius (see test two and three in table 6.13).

Test one shows a very heterogeneous scattering regarding the preferred temperature of the individual plants. The values are between −181.44° Celsius and +189.16° Celsius. Of course, the mean preferred temperature of all plants is about 10.4° Celsius (exactly at 10.38° Celsius) and the standard deviation is 50.95° Celsius[8].
Running the model with that configuration can be interesting, although it was not planned. The plants on the fully occupied lattice start to die after about 400 periods. When the first plants are dead, free space for the next generation is available and the plants with a slightly higher life expectancy start to breed. Of course, they breed only in the case of fitting the temperature preferences.

[8]For this test only one run was executed. Hence, the values are not as close to the parameter inputs. In contrast see the results in section 6.4.4 where five runs were carried out.

At time step 462 a surprisingly high quantity of plants (2413) are living on the lattice (see figure 6.17).

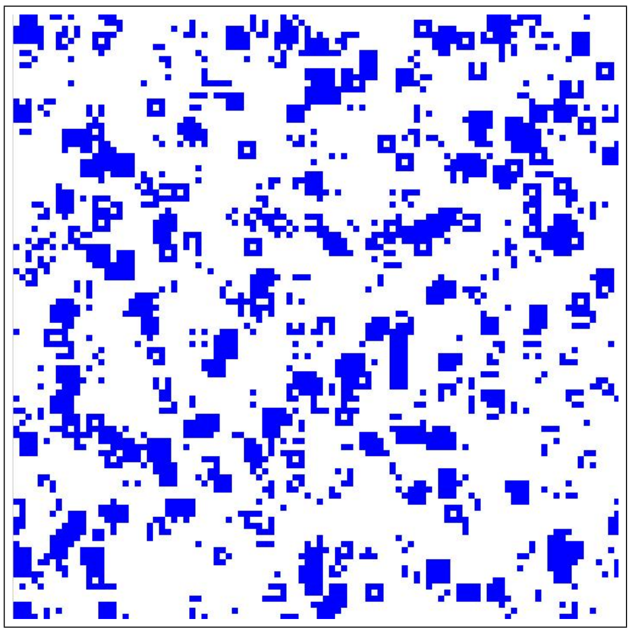


Figure 6.17: Test one at time step 462

At time step 860 all plants of the second generation are dead and the new population has an amount of 453 (see figure 6.18). From the beginning two phases of dying happens until this point. At each time less than one quarter of the population could create one successor on average. The test will go to the next dying phase and about 100 plants should exist thereafter.
The presumption that circa 100 plants are existent, after the third generation is completely disappeared, fits more or less to the empirical result. After 1298 time steps 76 plants are on the surface. The allocation on the lattice shows that only a few plants bred, the concentration on certain areas is obvious (see figure 6.19). After the fourth phase of dying only seven plants are left, and at time step 2009 the species is extinct.
The result of test one is explainable by a closer look to the underlying gaussian distribution. The most interesting point is the probability, that the distribution provides for breeding. Each plant needs a preferred temperature that fits into the tolerance interval of ±4° Celsius to the environmental conditions. That are circa 10.4° Celsius for each cell. From the view point of the environment, it means a plant must prefer an interval of [6.4° Celsius, 14.4° Celsius]. The values for mean (μ) and for standard deviation (σ) are 10.4° Celsius and 50° Celsius. An integral of the gaussian distribution shows that just a probability of about 6.4% is left for each plant to achieve this interval (see equation 6.6 and figure 6.20).

$$\int_{6.3}^{14.3} \frac{1}{\sqrt{2\pi} \cdot 50} \cdot e^{\frac{-(x-10.3)^2}{2 \cdot 50^2}} = 0.0637627 \tag{6.6}$$

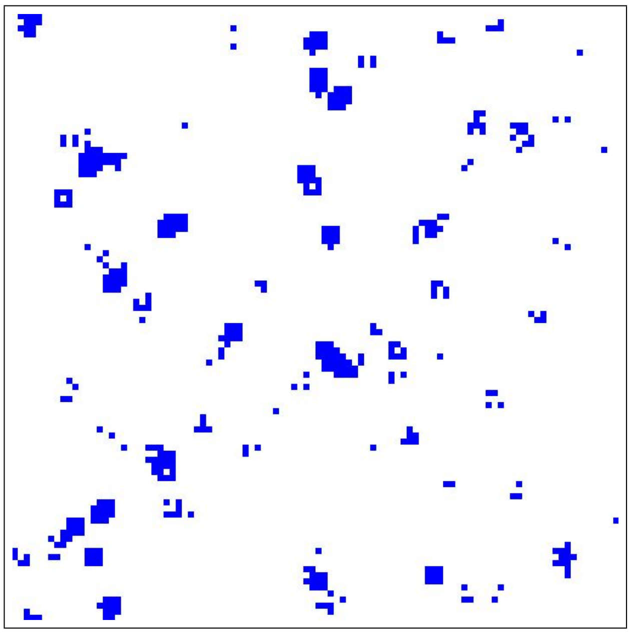


Figure 6.18: Test one at time step 860

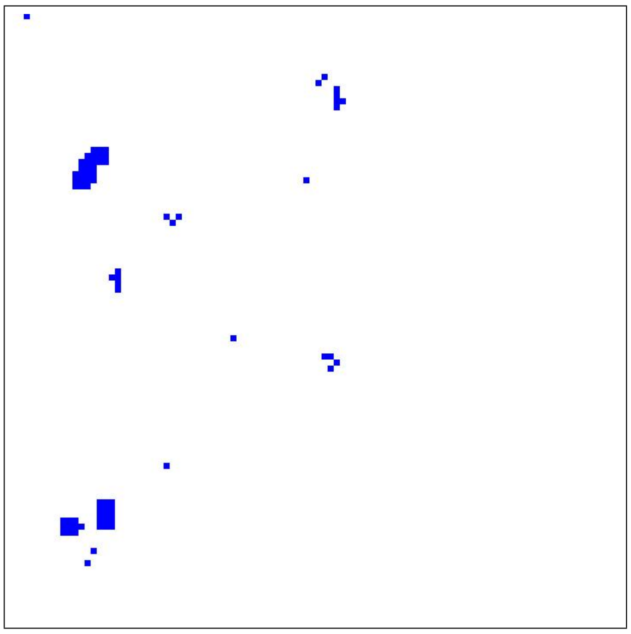


Figure 6.19: Test one at time step 1298

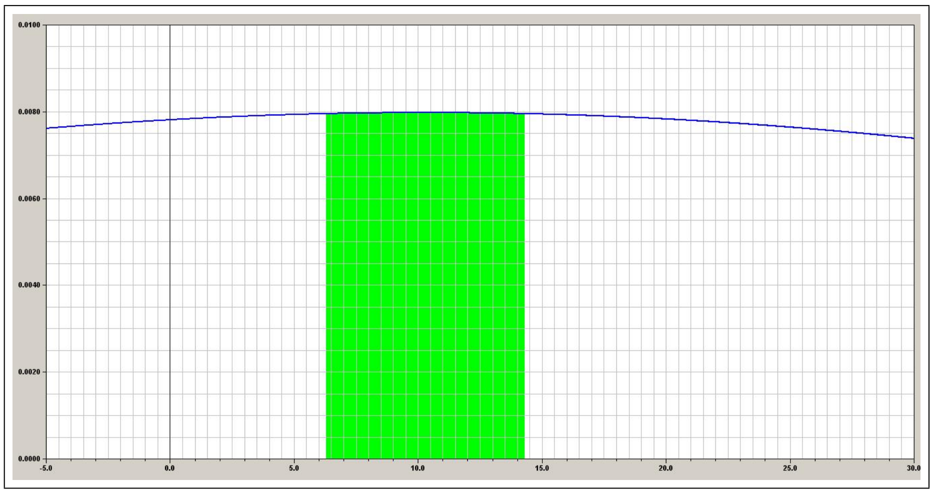


Figure 6.20: Tolerance interval for test one

Test number two uses a zero standard deviation and a *setup ratio* of 0.05 (see table 6.13). It is to assume that a growing of species' population occurs at this test, hence, a small *setup ratio* is applied.
The growth of population in test two is enormous. All plants breed. That behavior is reasoned by the fact that all values for preferred temperature are still inside the tolerance interval. Because of this behavior a new test is designed. The test has the same configuration as test number two, but the *temperature standard deviation* is increased to 0.1° Celsius and the *setup ratio* is decreased to 0.01 to get a better overview (see test four in table 6.13).
The result of test four is that only circa 19 out of 118 plants start to breed (see figure 6.21). These are exactly the plants that had a preferred temperature inside the tolerance interval. Repetitions of this test show a strong fluctuation of the amount of plants that are breeding.
Test three shows a very similar behavior to test two and a separate discussion is not needed.

6.4.8 Exploration of the *maturity mean*

Maturity is the state that enables a plant to breed. Hence, it is a fundamental parameter. An assumption of the simulation model is that plants cannot loose their ability to breed once it is achieved. The argumentation is that, firstly, for plants it makes no sense to stay alive and secondly, many examples exist of plants that die very soon after their reproduction.
A plant that cannot breed, because it lost for example its reproductiveness through age, is useless from an evolutionary view point. Social species – like humans or great apes – can still support their relatives or other members of their species in many different ways, but plants cannot.
Examples for plants that are dying even directly after their breeding are *Tozzia alpina* an alpine plant (Abderhalden and Schmidt 1924), or *Ficaria verna* (Esdorn 1961).

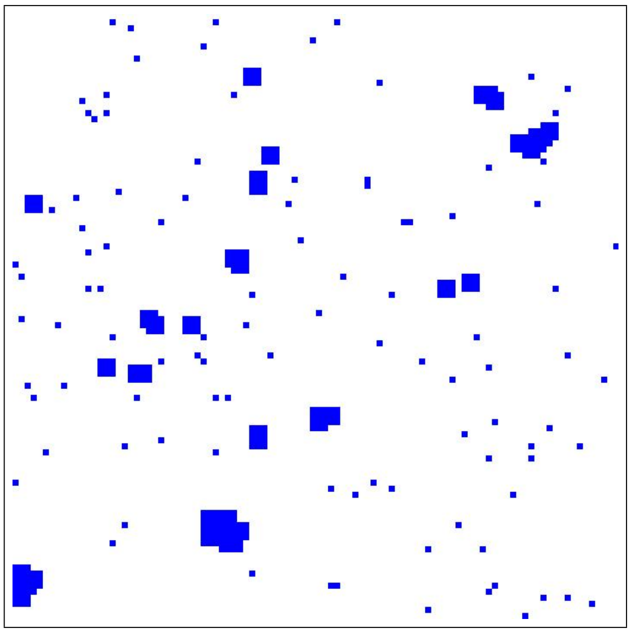


Figure 6.21: Test four at time step 50

However, fluctuations of plants' breeding ability over time are certainly possible. Therefore this is incorporated in the model, but not included in the tests.

The *maturity mean* is the average age for plants of a species to become able to breed. The maturity is based on a gaussian distribution, as well as the preferred humidity, temperature and the life expectancy.
The test design, that has been already used for *humidity* and *temperature mean* (see section 6.4.2 and 6.4.5) can be a simple and useful start for testing the *maturity mean*. One would expect a fast growth at small values for *maturity mean*, hence, the amount of species will be already measured after 25 periods. A fixed amount of ten plants at fixed positions is used for this test. The *maturity mean* will increase during the experiment by steps of 0.5 periods. All other conditions are unchanged (see table 6.1). The experiment is five times repeated for each configuration and the average results are structured in table 6.14.

Maturity mean	TA(S_a) at t=0	TA(S_a) at t=25	increase
0.0 periods	10.0	10000.0	99900.0%
0.5 periods	10.0	9966.8	99568.0%
1.0 periods	10.0	9357.4	93474.0%
1.5 periods	10.0	7686.2	76762.0%
2.0 periods	10.0	5905.0	58950.0%
2.5 periods	10.0	4514.6	45046.0%
3.0 periods	10.0	3222.4	32124.0%
3.5 periods	10.0	2273.0	22630.0%
4.0 periods	10.0	1720.0	17100.0%
4.5 periods	10.0	1324.0	13140.0%

5.0 periods	10.0	1042.0	10320.0%
5.5 periods	10.0	851.8	8418.0%
6.0 periods	10.0	721.2	7112.0%
6.5 periods	10.0	607.4	5974.0%
7.0 periods	10.0	518.4	5084.0%
7.5 periods	10.0	480.8	4708.0%
8.0 periods	10.0	427.6	4176.0%
8.5 periods	10.0	351.4	3414.0%
9.0 periods	10.0	279.2	2692.0%
9.5 periods	10.0	256.2	2462.0%
10.0 periods	10.0	250.0	2400.0%
10.5 periods	10.0	249.8	2398.0%
11.0 periods	10.0	248.6	2386.0%
11.5 periods	10.0	240.6	2306.0%
12.0 periods	10.0	226.6	2166.0%
12.5 periods	10.0	164.4	1544.0%
13.0 periods	10.0	130.4	1204.0%
13.5 periods	10.0	108.0	980.0%
14.0 periods	10.0	93.8	838.0%
14.5 periods	10.0	90.0	800.0%
15.0 periods	10.0	90.0	800.0%
15.5 periods	10.0	90.0	800.0%
16.0 periods	10.0	90.0	800.0%
16.5 periods	10.0	90.0	800.0%
17.0 periods	10.0	90.0	800.0%
17.5 periods	10.0	90.0	800.0%
18.0 periods	10.0	90.0	800.0%
18.5 periods	10.0	90.0	800.0%
19.0 periods	10.0	90.0	800.0%
19.5 periods	10.0	90.0	800.0%
20.0 periods	10.0	90.0	800.0%
20.5 periods	10.0	90.0	800.0%
21.0 periods	10.0	90.0	800.0%
21.5 periods	10.0	90.0	800.0%
22.0 periods	10.0	90.0	800.0%
22.5 periods	10.0	90.0	800.0%
23.0 periods	10.0	85.2	752.0%
23.5 periods	10.0	80.4	704.0%
24.0 periods	10.0	74.0	640.0%
24.5 periods	10.0	61.2	512.0%
25.0 periods	10.0	45.2	352.0%
25.5 periods	10.0	32.4	224.0%
26.0 periods	10.0	22.8	128.0%
26.5 periods	10.0	14.8	48.0%
27.0 periods	10.0	10.0	0.0%
27.5 periods	10.0	10.0	0.0%
28.0 periods	10.0	10.0	0.0%

Table 6.14: Results of a changing *maturity mean*

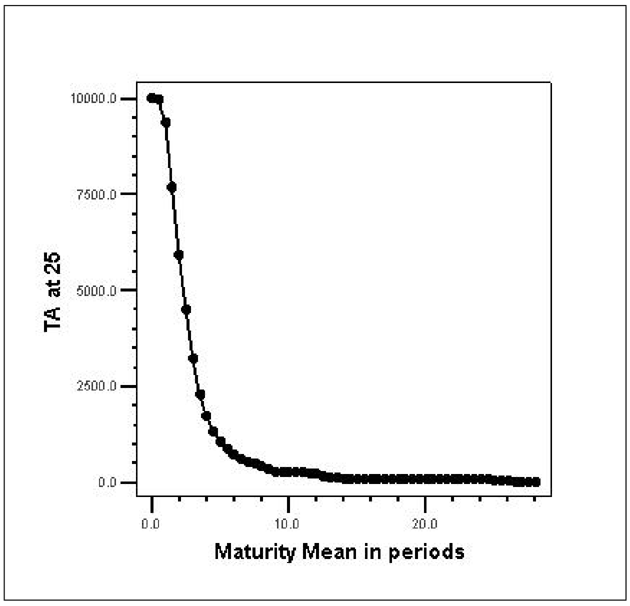


Figure 6.22: Average amount at time step 25 depending on *maturity mean*

Table 6.14 gives an interesting overview of the dependency from *maturity mean* and the amount of plants. The plants of the runs that are summarized in the first row have all achieved a number of 10,000 at time step 25 but not earlier. All in all it seems that the amount of plants decreases disproportionally with a raise of the *maturity mean.* That is reasoned by the exponentially growth of plants. The delay of reproduction – reasoned by an increasing *maturity mean* – does not affect only the direct successors of the current generation. It also influences the generation after that. The successors of the successors have to wait on average double the time of the delay. To get a better overview figure 6.22 represents the data of table 6.14 graphically.
A more detailed observation shows that there are some *plateaus* when the *maturity mean* is increased. Figure 6.23 shows an extraction of the data from table 6.14, that means only the results from a *maturity mean* of nine periods and more are shown. This allows a smaller scaling and therefore a better resolution and discovers some plateaus.
The plateaus are in the interval 10 to 12 periods and 14 to 22.5 periods for *maturity mean.* They are reasoned by the fact that the individuals, that are mature, after their first breeding are not neighbored by free cells. The optimal environmental conditions give them the opportunity to breed directly after their maturity. Then the circled expansion happens (see section 6.2) and the mature individuals are surrounded by their own successors. The younger plants that surround the older ones are not able to breed, yet. Therefore all plants have to wait until the plants on the periphery are mature.
Hence, the reason for the first plateau is that the plants can propagate only twice during the whole simulation run. In the time span of 25 periods the plants have only two phases of breeding, it does not matter whether the plants wait ten, eleven or twelve periods on average. In other words: The plants have

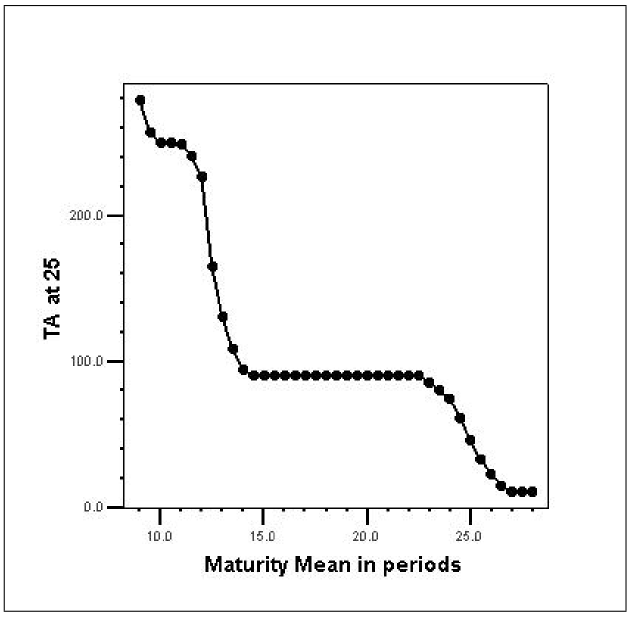


Figure 6.23: Average amount at time step 25 depending on *maturity mean* (extraction)

no additional benefit regarding their amount, if they have two and a half phases of breeding compared to only two phases.
With a short *maturity mean* there is no delay in breeding and hence no plateau occurs. The longer the time to maturity takes, the longer is the plateau. This can be explained by dividing the average duration, until a plant is able to breed – and that is nothing else than the *maturity mean* – by the length of the run, and that is in this experiment 25 time steps. An example can be a comparison of values of both plateaus. The *maturity mean* of eleven is part of the first plateau. Dividing eleven by 25 gives the idea that about two breeding phases happen and three periods remain.
A *maturity mean* of 15 inside the second – the longer – plateau (see figure 6.23) is another example. The rest of 15 divided by 25 is ten, but in ten periods no further breeding is possible. Hence, with an increasing *maturity mean* the plateaus are longer, because of the increasing amount of time for *waiting.*
The emergence of plateaus can only happen because of the relative low *maturity standard deviation* of one period.

6.4.9 Exploration of the *maturity standard deviation*

The *maturity standard deviation* embodies the deviance from the average age when a plant is enabled to breed.
To test the *maturity standard deviation* the criterion of the amount of individuals is used again. The point of time to measure is 50 time steps and all parameter settings are reset to default, except for *maturity standard deviation* and the *setup ratio*. Hence, the *maturity mean* has the value of four periods during the whole experiment. The *setup ratio* is used the same way as in the

Maturity standard deviation	$TA(S_a)$ at t=0	$TA(S_a)$ at t=50	increase
0.0 periods	10.0	5000.0	49900.0%
0.5 periods	10.0	4971.6	49616.0%
1.0 periods	10.0	5491.4	54814.0%
1.5 periods	10.0	6193.8	61838.0%
2.0 periods	10.0	7266.8	72568.0%
2.5 periods	10.0	8088.6	80786.0%
3.0 periods	10.0	8858.0	88480.0%
3.5 periods	10.0	9448.8	94388.0%
4.0 periods	10.0	9837.2	98272.0%
4.5 periods	10.0	9897.8	98878.0%
5.0 periods	10.0	9990.8	99808.0%
5.5 periods	10.0	9979.6	99696.0%
6.0 periods	10.0	9987.6	99776.0%
6.5 periods	10.0	9997.4	99874.0%
7.0 periods	10.0	9998.8	99888.0%
7.5 periods	10.0	10000.0	99900.0%
8.0 periods	10.0	10000.0	99900.0%

Table 6.15: Results of a changing *maturity standard deviation*

experiment for *maturity mean* (see section 6.4.8).
The experiment is implemented in half period steps for the *maturity standard deviation*. And for each setting the simulation program runs five times. Table 6.15 shows the results of this experiment, but figure 6.24 makes the positive dependency more obvious. With an increasing *standard deviation* for *maturity*, the amount of plants at 50 time steps increases, too.

This can be surprising, because the *maturity mean* is four periods and with an increasing *standard deviation* the probability spreads in both direction. Higher as well as lower ages than the *mean* for *maturity* are supported. Some plants get a maturity of more than four periods, and about the same amount become mature before they have reached the fourth period of life. Therefore the chances should be balanced.
However, the shape of the islands that the plants develop is much more jagged (see figure 6.25) than the non-jagged shapes by a small *standard deviation* for *maturity*. With a low *maturity standard deviation* the propagation happens more steady. The jagged shape allows the occupation of more empty cells in one period, because more empty cells are in the neighborhood, as shown in (b) in figure 6.26. Vegetation of a non-jagged island is limited in its breeding (see (a) in figure 6.26[9]). Therefore the amount of empty cells, neighbored by existing individuals is higher than in a non-jagged island by the same amount of plants and hence, the propagation can occur faster.

Another reason for the increasing amount of plants with an increasing *maturity standard deviation* is the possibility of negative values for the maturity. If the *standard deviation* for *maturity* is relative high, and the *maturity mean* low

[9]Both examples in figure 6.26 have an equal amount of plants.

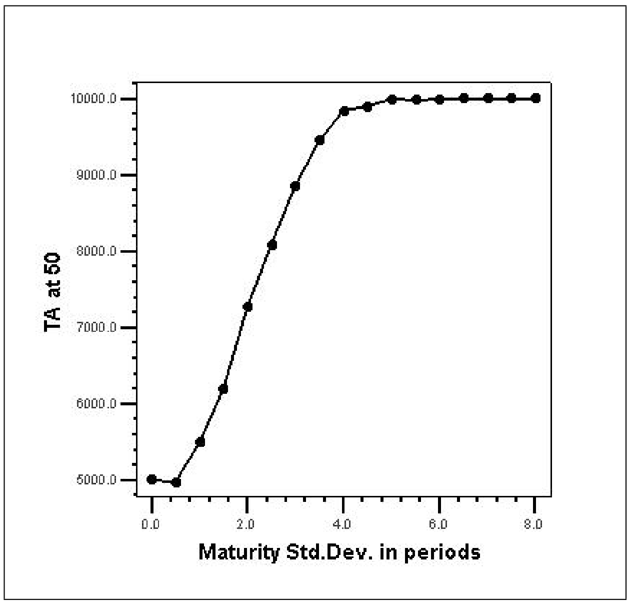


Figure 6.24: Average amount at time step 50 depending on *maturity standard deviation*

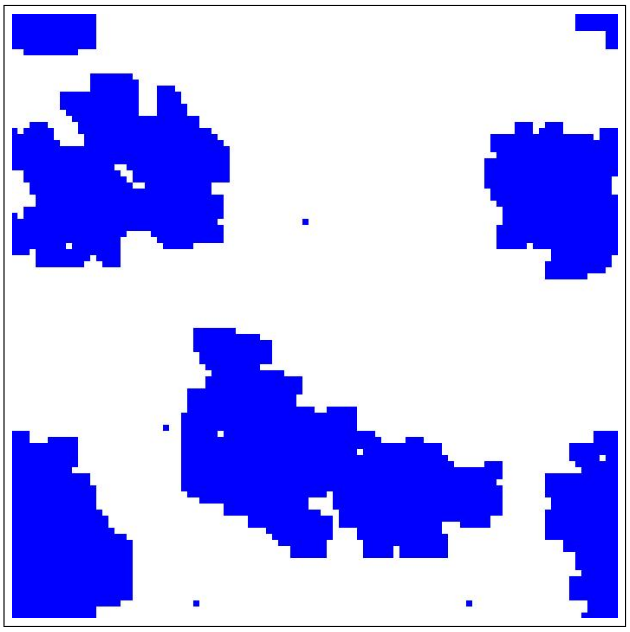


Figure 6.25: Scatter of plants at time step 14 depending on *maturity standard deviation*

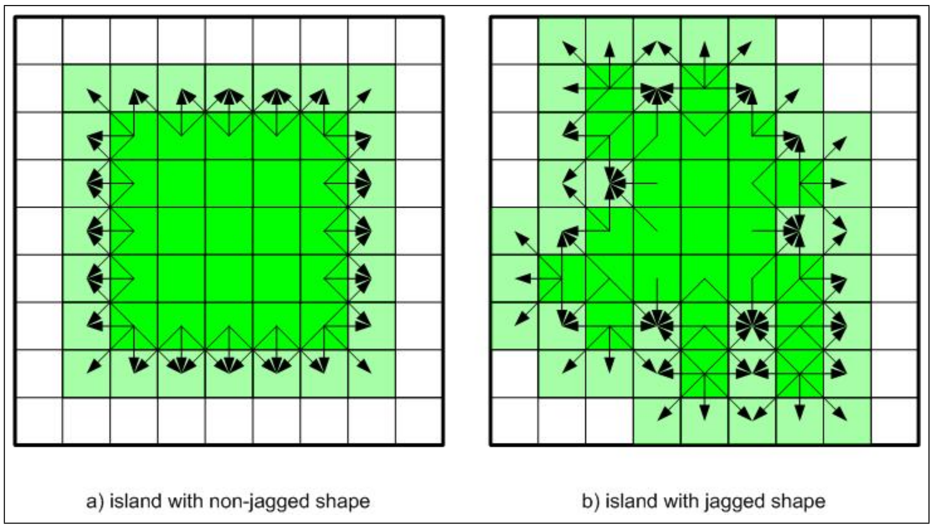


Figure 6.26: Shape of island as crucial factor for propagation

(for instance, four periods as in this experiment), then some plants get negative values for their maturity age. Therefore they are able to breed directly after they are born (more precisely one period later, because of the model's process (see section 5.2.3)). Hence, by an increasing *maturity standard deviation* many plants exist that take a long time until they develop their ability to breed. On the other hand, many other plants can breed directly.
Combining both findings – firstly, the heterogeneous pattern regarding maturity, and secondly, the realization that a more riven shape of vegetation triggers an increase of reproduction speed – leads to the explanation of the positive dependency between *maturity standard deviation* and the amount of plants. The breeding capacity of plants, that are slow in achieving maturity are easily compensated by those plants that have matured fast. The reason is that after a short while a jagged structure of islands is established, and the fast plants are able to substitute the slow ones lack of breeding with ease.

6.4.10 Exploration of the *life expectancy mean*

The value for life expectancy is determined at the time when a plant is born (see also section 5.2.2). According to table 6.1 the currently applied value of the *mean* of *life expectancy* is 400 periods. When we assume that one period is equal to a year's length, it seems to be an unrealistic value[10]. The application of such a high value is reasoned by the argumentation that a *ceteris paribus* approach has advantages for the analysis. If the plants' death happen too early a disadvantageous intervention occurs and the results of measurements for other parameters are influenced (see section 6.4).
By testing the influence of *life expectancy mean*, however, lower values should by applied. It can be even useful to start with a value of zero. Table 6.16 shows in the left column the applied values for this experiment. All other values are

[10]The indications in botany literature, for instance, estimate that the maximum age for tussocks of *Eriophorum vaginatum* (Alaska) is about 16 years (Fetcher and Shaver 1983).

shown in table 6.1, hence, the *life expectancy standard deviation* is two periods. An exception is again the *setup ratio*. The initial amount of species is exactly 20, and the time step to measure is 25. Each setting runs five times and the average is used as result.

Life expectancy mean	TA(S_a) at t=0	TA(S_a) at t=25	increase
0.0 periods	20.0	1.4	-93.0%
0.5 periods	20.0	4.2	-79.0%
1.0 periods	20.0	11.2	-44.0%
1.5 periods	20.0	30.8	54.0%
2.0 periods	20.0	142.2	611.0%
2.5 periods	20.0	348.6	1643.0%
3.0 periods	20.0	591.2	2856.0%
3.5 periods	20.0	1032.4	5062.0%
4.0 periods	20.0	1423.8	7019.0%
4.5 periods	20.0	1854.6	9173.0%
5.0 periods	20.0	2007.4	9937.0%
5.5 periods	20.0	2291.6	11358.0%
6.0 periods	20.0	2313.8	11469.0%
6.5 periods	20.0	2534.2	12571.0%
7.0 periods	20.0	2795.6	13878.0%
7.5 periods	20.0	2930.6	14553.0%
8.0 periods	20.0	2988.0	14840.0%
8.5 periods	20.0	3157.0	15685.0%
9.0 periods	20.0	3125.2	15526.0%
9.5 periods	20.0	3115.6	15478.0%
10.0 periods	20.0	3190.6	15853.0%
10.5 periods	20.0	3254.2	16171.0%
11.0 periods	20.0	3287.0	16335.0%
11.5 periods	20.0	3298.6	16393.0%
12.0 periods	20.0	3301.2	16406.0%
12.5 periods	20.0	3285.4	16327.0%
13.0 periods	20.0	3313.0	16465.0%
13.5 periods	20.0	3325.6	16528.0%
14.0 periods	20.0	3360.4	16702.0%
14.5 periods	20.0	3334.6	16573.0%
15.0 periods	20.0	3304.8	16424.0%
15.5 periods	20.0	3254.8	16174.0%
16.0 periods	20.0	3368.6	16743.0%
16.5 periods	20.0	3368.0	16740.0%
17.0 periods	20.0	3439.8	17099.0%
17.5 periods	20.0	3254.8	16174.0%
18.0 periods	20.0	3385.2	16826.0%
18.5 periods	20.0	3401.6	16908.0%
19.0 periods	20.0	3368.8	16744.0%
19.5 periods	20.0	3386.0	16830.0%
20.0 periods	20.0	3424.2	17021.0%
20.5 periods	20.0	3378.4	16792.0%
21.0 periods	20.0	3412.6	16963.0%

21.5 periods	20.0	3328.6	16543.0%
22.0 periods	20.0	3401.0	16905.0%
22.5 periods	20.0	3452.2	17161.0%
23.0 periods	20.0	3402.8	16914.0%
23.5 periods	20.0	3480.8	17304.0%
24.0 periods	20.0	3407.0	16935.0%
24.5 periods	20.0	3375.8	16779.0%
25.0 periods	20.0	3404.0	16920.0%
25.5 periods	20.0	3404.0	16920.0%
26.0 periods	20.0	3364.8	16724.0%

Table 6.16: Results of a changing *life expectancy mean*

The results of the experiment with a changing *life expectancy mean* discover a dependency to the amount of plants in a positive relationship. An increase of the *life expectancy mean* leads to more individuals. This is absolutely obvious, and it only shows that the model works right regarding the parameter *life expectancy mean.*
Much more interesting is the diminishing marginal growth, easier to recognize in figure 6.27. By a raise of the *life expectancy mean* of circa ten periods and more, the amount of plants does not increase any more. Even with a value of 400 periods there is only an average amount of 3402.8 plants at time step 25. That is less than the maximum from the experiment (see the row for *life expectancy mean* of 23.5 in table 6.16). Spreading is still constrained by other parameters, in particular the *maturity mean.*
The diminishing marginal growth is reasoned by the high probability of reoccupation inside the islands. Older plants are principally located inside the islands and not at the borders. When they die, the released cells are very close to cells occupied with mature plants and therefore a reoccupation is very likely. However, this does not increase the whole population. The growing of the circumference (and also the diameter) of the islands stagnates. It seems to be independent from *life expectancy mean.* It depends much more on the maturity (see also section 6.4.8 and 6.4.9).
However, at lower values for *life expectancy mean* many plants cannot breed, because their maturity is higher than their life expectancy. Figure 6.28 shows that a sparsely declining marginal population growth happens by an increase of *life expectancy mean* for values of 4.5 periods and higher. That is in the first view contra intuitive, because this is almost exactly the same time for maturity (the *maturity mean* is four periods). By considering the concentration of plants, however, it is explainable. It is reasoned by substitution that happens, because of the dense coexistence in islands. Early dying plants can be replaced by successors of long living plants. The *life expectancy standard deviation* is, of course, crucial for this behavior. In this experiment its value is two periods.

The next section will provide new information about the behavior concerning a change of the *standard deviation* for *life expectancy.*

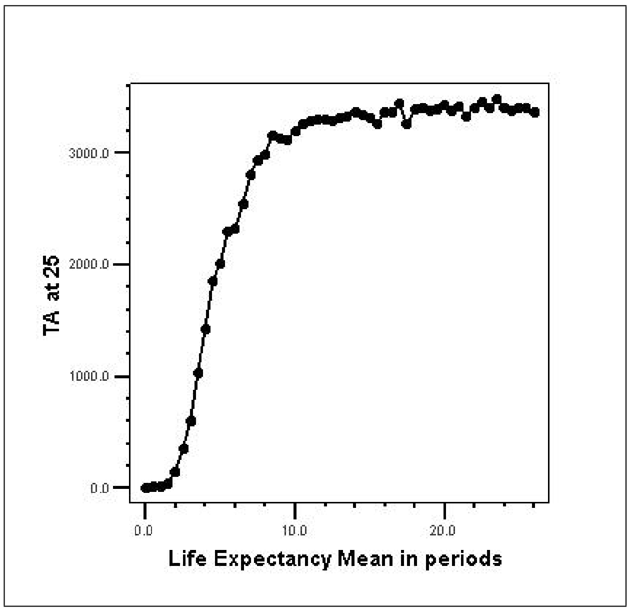


Figure 6.27: Average amount of plants depending on *life expectancy mean* at time step 25

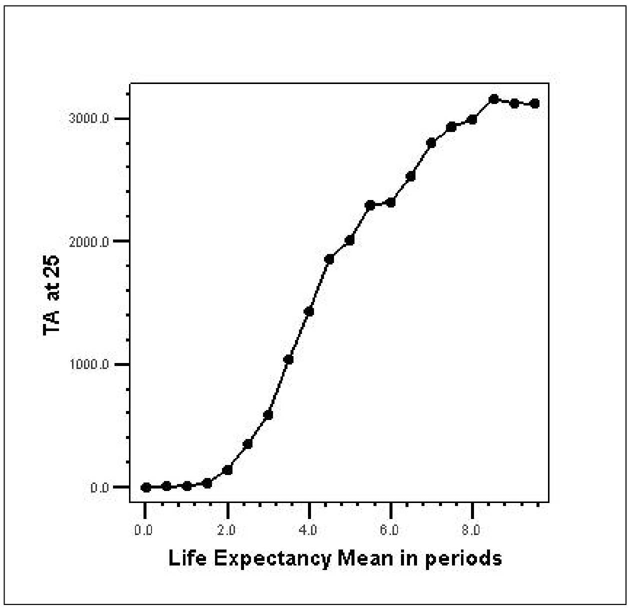


Figure 6.28: Average amount of plants depending on *life expectancy mean* at time step 25 (extraction)

Life expectancy standard deviation	$TA(S_a)$ at t=0	$TA(S_a)$ at t=25	increase
0.0 periods	20.0	3050.0	15150.0%
0.5 periods	20.0	3062.6	15213.0%
1.0 periods	20.0	2874.6	14273.0%
1.5 periods	20.0	2713.8	13469.0%
2.0 periods	20.0	2492.4	12362.0%
2.5 periods	20.0	2467.8	12239.0%
3.0 periods	20.0	2666.8	13234.0%
3.5 periods	20.0	2060.0	10200.0%
4.0 periods	20.0	1926.2	9531.0%
4.5 periods	20.0	1683.2	8316.0%
5.0 periods	20.0	1843.6	9118.0%
5.5 periods	20.0	1836.2	9081.0%
6.0 periods	20.0	1698.4	8392.0%
6.5 periods	20.0	1827.6	9038.0%
7.0 periods	20.0	1631.6	8058.0%
7.5 periods	20.0	1634.2	8071.0%
8.0 periods	20.0	1494.6	7373.0%

Table 6.17: Results of a changing *life expectancy standard deviation*

6.4.11 Exploration of the *life expectancy standard deviation*

In the table (6.1) for basic parameter settings, the *life expectancy standard deviation* is two periods. Changing that, provides possibly, a better understanding of the data from the experiment concerning *life expectancy mean* (see section 6.4.10).

Again five runs for each setting are executed and aggregated to an average value. The *life expectancy mean* should not be as high as in table 6.1, but on the other hand, not too low. So a value of six periods could be useful. The amount of plants at time step zero is 20, exactly as in the experiment for *life expectancy mean* (see section 6.4.10). Hence, it is easier to combine the results from the different experiments for standard deviation and mean.
All other parameters are set to the values of table 6.1. Table 6.17 shows the average total amount of plants after 25 periods.
The result is a declining amount of plants if the *life expectancy standard deviation* is increased. Figure 6.29 graphically shows that dependency.
The dwindling population, with an increase of *life expectancy standard deviation*, is triggered by the fact, that too many plants are dying early, often before they are mature. A high *standard deviation* for *life expectancy* leads to a small number of plants that are able to breed, and the more it is increased, the lower are the chances that this amount of plants is big enough to keep the population alive.
The *life expectancy mean*, of course, is crucial. Weakened is the dependency of *life expectancy standard deviation* with a considerably higher value for the mean.

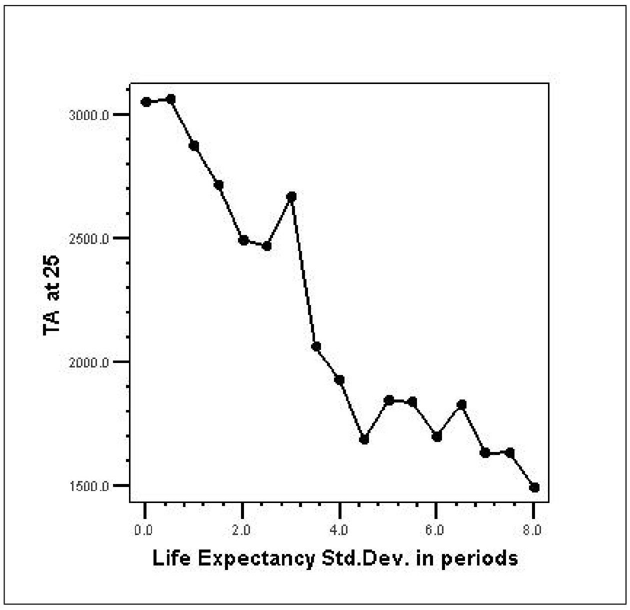


Figure 6.29: Average amount of plants at time step 25 depending on *life expectancy standard deviation*

The last four sections (6.4.8, 6.4.9, 6.4.10, and this section) are showing a strong interdependence of the researched values. These are the *maturity mean*, *maturity standard deviation*, *life expectancy mean*, and *life expectancy standard deviation*. The dependency between these parameters is realistically implemented. In section 6.5 we will see an experimental design, which considers changes of all parameters at the same time, and hence, also settings, that seems to be natural for these four parameters. For example, the *maturity mean* should be lower than the *life expectancy mean*.

6.4.12 Exploration of species competition

Competition strength is an abstract value. The higher it is, the more likely the plant wins a competition. It is not the only determinant for the result of a competition game. Each competition consists of at least of two values. That is the case for a competition of two plants. For a competition between three plants, three values exist, and for n plants n values determine the *winner*.
Hence, the relative *competition strength* is crucial. Always the strength of all involved plants must be considered.
The *competition strength* is multiplied with a random value and the result is the current strength of a certain plant. The plant with the highest current strength wins the competition, and this individual occupies the considered cell.
By applying that algorithm the *competition strength* is much less important than it could be. For instance a solution without the random value – that only uses the *competition strength* to determine the winner – would be much simpler, but would also give species with a higher *competition strength* – even it is a very small difference – the absolute advantage. The strongest species would always win and the others would have not the slightest chance to win a competition.

Competition strength S_b	TA(S_a) at t=0	TA(S_b) at t=0	TA(S_a) at t=25	TA(S_b) at t=25
0	16.2	25.2	1980.8	3174.0
1	20.4	15.8	2620.4	2017.4
2	17.6	20.6	2244.0	2580.0
3	20.6	23.8	2260.8	2742.4
4	18.8	16.0	2453.6	2028.8
5	22.4	22.0	2848.8	2702.4
6	19.0	18.2	2383.0	2326.6
7	16.4	16.6	2156.0	2262.4
8	17.0	18.0	2193.8	2240.4
9	17.0	18.6	2247.8	2334.4
10	19.8	20.0	2571.0	2475.4

Table 6.18: Results of a changing *competition strength* for two species at time step 25

This cannot be seen as a realistic behavior.
The solution with the random value, on the other hand, leads to weakening of the factor *competition strength.* Is the random value for one plant only slightly above the random value for the rival plant, than the first plant can win, even with a relative low *competition strength.* The multiplication of both values – *competition strength* and random value – by the algorithm is the reason.
However, in a long term the plants of the strongest species will prevail, but the random characteristic of the value that is multiplied with the *competition strength* leads to a compensation of extreme results. With the algorithm a *fair*[11] and realistic mixture out of random determination and user influence is provided.

To measure effects of *competition strength* at least two species are needed. Of course, competition happens in simulation runs with only one species, too. If more than one plant tries to place a seedling on the same cell at the same time step, the plants are competing, and following the same rules, even though they are from the same kind of species. If plants from the same species compete, they have an equal *competition strength*, because the parameter is a characteristic of the species and not specific for the individual.
For the experiment two species (S_a and S_b) with equal parameter settings are used. The only difference is the value for *competition strength.* While species S_a has a constant *competition strength* of exactly ten, changes the *competition strength* of S_b in steps of one. Table 6.18 shows the values for *competition strength* of S_b and the average results of five runs for the different settings at time step 25 analogously.
The results are surprising, it is not possible to recognize a tendency or dependency. The reason is that competition between plants of different species did not happen often enough to show a significant influence. The low *setup ratio* of 0.002 and the short time of only 25 periods does not lead to many cases where plants of different species are approaching each other locally. So the influence

[11]If more extreme competition is requested, it is still possible to set the values for *competition strength* with such extreme differences, that a certain species always prevails and another loses.

Competition strength S_b	TA(S_a) at t=0	TA(S_b) at t=0	TA(S_a) at t=25	TA(S_b) at t=25
0	497.5	498.4	5366.2	4622.8
1	497.8	489.8	5375.3	4624.7
2	485.8	484.1	5306.5	4694.0
3	497.5	481.2	5286.4	4713.6
4	490.6	491.4	5262.3	4737.7
5	485.5	485.6	5081.2	4918.8
6	483.9	484.0	5073.0	4927.0
7	486.4	487.5	5061.6	4938.4
8	481.0	481.3	5047.1	4952.9
9	496.2	496.3	5038.4	4961.6
10	478.0	475.0	4999.6	5000.4

Table 6.19: Results of a changing *competition strength* for two species at time step 25 (*setup ratios*=0.05)

of the *competition strength* is only important in the case of a certain density of populations or by allowing wind borne seedings.
Hence, it is necessary for the next experiment to increase the measure time or the *setup ratio*. An increase of *setup ratio* seems to be a useful method. If it is high enough, competition will happen from the first time steps on. By repeating the last experiment all conditions are unchanged, but the *setup ratio* is increased to 0.05 for both species (S_a and S_b). In this experiment the model runs for each setting ten times.
The result (see table 6.19 and figure 6.30) of the second experiment is a clearly dependency between *competition strength* and the breeding. The amount of all plants (from S_a and S_b) is always 10,000 at time step 25. That is the size of the whole lattice. Because of the relative high *setup ratio* of 0.05 the occupation of all cells happens very fast and both different species come into conflict. The *competition strength* supports S_a (except of the last setting, when both species have a *competition strength* of ten (see table 6.19)). S_a is superior, but with a declining difference in *competitive strength* the edge decreases as well. Figure 6.30 shows S_b and its increasing amount of plants with an elevation of its *competitive strength*. The analysis of S_a is actually only the opposite of the one for S_b, because of the total amount of 10,000 individuals.
To test how sensitive species behave, a long term experiment is useful. It is crucial for that experiment that one species occupies the whole lattice. That means it has to crowd out all plants of the other species and occupy 10,000 cells. If a species is eliminated once, it can never come back in the same run, hence the decision is clear and the point of time is recorded. All conditions are equal for both species, only the *competition strength* is for S_a exactly ten and 9.999 for S_b. Table 6.20 shows that the behavior of the model is not sensitive enough to trigger a visible effect from the slightly different *competition strengths* in ten runs.
The difference of the *competition strengths* must be increased. For the next experiment S_b has a value of 9.99.
This time a reaction is measured. Table 6.21 shows, that in six of ten runs S_a is predominant.

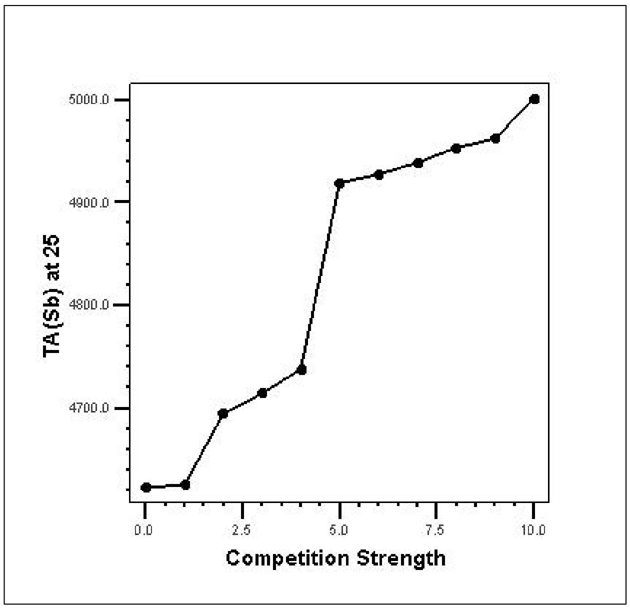


Figure 6.30: Average amount at time step 25 depending on *competition strength*

Number of run	survived species	time step of fully occupation
1	S_b	124043
2	S_a	125288
3	S_a	1704932
4	S_a	463491
5	S_b	293482
6	S_b	373846
7	S_a	234948
8	S_a	411197
9	S_b	175573
10	S_b	465710

Table 6.20: Results of a difference in *competition strength* of 0.001

Number of run	survived species	time step of fully occupation
1	S_b	426799
2	S_b	737911
3	S_a	208366
4	S_a	278573
5	S_a	452110
6	S_a	537616
7	S_b	409201
8	S_b	404235
9	S_a	77528
10	S_a	181945

Table 6.21: Results of a difference in *competition strength* of 0.01

Number of run	survived species	time step of fully occupation
1	S_a	110000
2	S_a	84760
3	S_a	129306
4	S_a	92814
5	S_a	300888
6	S_a	231642
7	S_a	79930
8	S_a	153460
9	S_a	180251
10	S_a	55862

Table 6.22: Results of a difference in *competition strength* of 0.1

S_a(*competition strength*)	mode survived species	average time step of full occupation
9.999	S_a, S_b	437251.0
9.99	S_a	371428.4
9.9	S_a	141891.3

Table 6.23: Summary of the detailed experiments regarding the *competition strength*

Additionally a third experiment is implemented. The difference of both species *competition strength* is 0.1, hence the value for S_b is changed to 9.9. It is not only expected that in the most – or probably even in all – runs S_a will prevail; additionally the time that is needed for S_a to occupy the whole lattice will be clearly shorter than in the both experiments before. The results in table 6.22 show the realization of both predictions.
Table 6.23 summarizes the values of the last three experiments. To get an overview, the mean for the amount of periods until full occupation is achieved, and the mode of the survived species is shown.
The very sensitive reaction of the model – shown by the last experiments – is reasoned by the enormous number of competition that happen in each run. On average each 400 time steps all cells are unoccupied, because that is the *mean* for *life expectancy* for both species. Therefore 10,000 competitions happen each 400 time steps. Of course, only a small amount of that is competition between plants of different species. However, the more plants of different species are neighbored to each other, the more effective is the *competition strength.*

Regarding *competition strength* the assumption has been made that the competition between different species is a one-dimensional construct. A species with the value ten for *competition strength* always has that value and it is an immaterial fact which species is the competitor. Because of different natural aspects of breeding it can be useful to define a *competition strength* for each possible combination of species competition. If there are n species, each species can compete with all other species. Therefore n^2 combinations are possible. It can be that a species is quite weak in competing with a second species, but compared with a third species it is relatively strong. A medium value for species one (S_1(cs)),

	S_a	S_b	S_c	...	S_n
S_a	0	S_b to S_a	S_c to S_a	...	S_n to S_a
S_b		0	S_c to S_b	...	S_n to S_b
S_c			0	...	S_n to S_c
...	...	...	...	...	...
S_n					0

Table 6.24: *Competition strength* as matrix

a high value for species two (S_2(cs)), and low value for species three (S_3(cs)) is only useful if S_3 is weaker than S_2. Thinking in one dimension for *species competition* determines exactly this result (see also equation 6.7).

$$S_2(cs) > S_1(cs) > S_3(cs) \tag{6.7}$$

However, it might be possible that S_3 is stronger than S_2. For example, S_3 is weaker than S_1 because a plant seedling of S_1 can establish itself and develop roots before a plant of S_3 can settle. Hence, S_1 supplants the seedling of S_3 and therefore the *competition strength* of S_1 is higher than the one of S_3. Additionally, S_2 is even faster in seedling than S_1. However, lets assume that S_3 is the only of the three species that has the ability to store much water[12]. That gives it a competitive advantage only over S_2, if an additionally assumption is, that the seedling of S_2 needs a lost of water to establish, but the seedling of S_1 does not.
In that example two dimensions for competing are mentioned, water and speed of establishment. Examples for much more dimensions are possible, but basically it is enough to enable the opportunity to define a *competition strength* for all combinations of species. That can be solved by using a n^2 matrix (see table 6.24 as a scheme). Reading the matrix works *from row to column*, that means the value, for instance, of S_b to S_a is the strength of S_b in a competition with S_a. If the S_a is stronger than S_b the value is negative. The competition matrix for the model can be changed via an Excel-file which is called *"competitionMatrix.xls"* and stored in the same folder as the files for the maps.

Eventually both opportunities for defining *competition strength* are provided by the model's framework. Firstly, the way that has been tested before, with absolute values for each species' *competition strength* and secondly the matrix solution. In the simulation program basically the solution with absolute values is activated, but a change to the matrix solution is simple.
Controlling the content of the matrix – and therefore the values for competition for the matrix solution – is possible by using Microsoft Excel or StarOffice.

6.5 Experimental design: Experiments with maps for environmental parameters

The framework of the simulation model provides the opportunity to decide freely over the environmental conditions (see section 5.2.1). Using different maps in

[12]Like Agaves and Cacti (Nobel 2003).

some experiments gives the chance for gaining new insights into the models behavior. The "perfect" maps that provided homogeneously a temperature of exact 10.4° and a humidity of 50% have been used in all of the tests before, except in section 6.3. With the achieved knowledge about the model by the completed test series, it is now time to roll out experiments with heterogenous environmental conditions and different plant species. Creating maps for this purpose is quite simple. It can be done with a simple drawing tool or by scanning maps. In that test series drawn maps are applied.

Experiment one: Heterogeneous landscape

For the fist experiment, the humidity and the temperature map are showing a dry and hot area in the north west. An average humidity of 10% and a temperature of 21° Celsius characterizes this "desert". Close to that area a small and lathy "valley" is denoted with a high humidity (about 80%) and slightly a lower temperatures (19° Celsius) than in the desert. In the east a hilly area has very cold temperatures (4° Celsius) and is also relatively dry (30% humidity). A mountain range is placed inside the hills with most cold and arid climate (−2.3° Cesius and 0%). The south west is an area that is similar to the maps that has been used in the test series before. It has an average temperature and humidity of 10.4° Celsius and 50%, respectively. Figure 6.31 shows the map to get an overview about these areas, but it is not the map that is used for the experiment to directly determine temperature and humidity. The input maps for the model will be created separately, based on this overview map. The values for temperature and humidity are not as homogeneous inside the areas, as the overview map may suggests. This landscape is, of course, quite variable for such a small space. However, it can be a useful test environment.

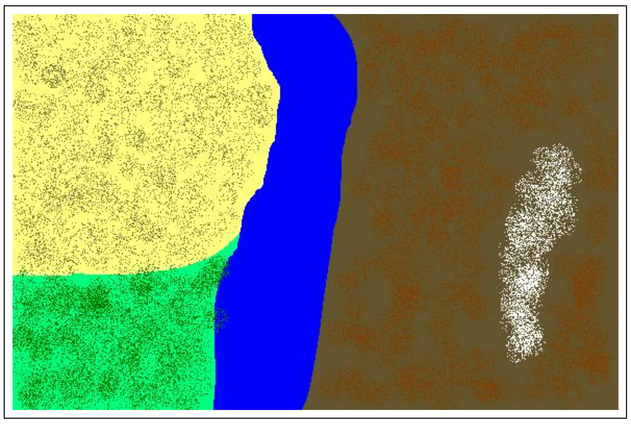


Figure 6.31: Overview map for environment

Different plant species are placed randomly at initialization of the first run. The exact description of these species are represented by their parameter values (see table 6.25). The *setup ratio* for all species has exactly the same value. The *competition strength* is also equal for each species. Competition should happen on the basis of the different environmental conditions, but not by different values for *competition strength.* The goal of that experiment is to figure out, whether

	S_d	S_t	S_h	S_v
Setup Ratio	0.05	0.05	0.05	0.05
Humidity Mean in%	0.1	0.5	0.3	0.8
Humidity Standard Deviation in%	0.05	0.05	0.1	0.05
Temperature Mean in ° Celsius	21.0	10.4	4.0	19.0
Temperature Standard Deviation in ° Celsius	1.0	1.0	1.5	1.0
Maturity Mean in periods	6.0	4.0	4.0	4.0
Maturity Standard Deviation in periods	1.0	1.0	1.0	1.0
Life Expectancy Mean in periods	20.0	30.0	30.0	30.0
Life Expectancy Standard Deviation in periods	2.0	2.0	2.0	2.0
Competition Strength	10.0	10.0	10.0	10.0

Table 6.25: Species parameter for experiment one

the plants are seeding in the regions that provide the specific conditions for their species.
The first species (S_d) fits best into the desert region in the north west, it likes aridness and warmth. However, to get mature it needs a relatively long time (its *maturity mean* is six periods) and also the life expectancy is just 20 periods on average. The second species is basically the test species (S_t) from table 6.1. The values for *temperature mean* and *humidity mean* are the same as in most of the implemented tests (see section 6.4). It is to presume that this species will occupy the south west. The third species (S_h) will probably achieve the hilly area. It prefers cold and arid environments. However, it is not highly specialized on that clime (hence, it has relative high *standard deviations* for *temperature* and *humidity*). S_v is an organism that fits very well into the valley. However, it can be difficult to survive for that species, because the valley is a small area, and it is surrounded by a dry desert and cold hills.
The results of the first run show almost exactly the predicted behavior. Already after 50 periods no plant is left at an area where it does not fit into regarding the prediction. The only exception is a small stripe east of the valley, the western part of the hilly land. Here S_t is established after a while (see figure 6.32). The maps are modeled with continuous transitions from one area to another. Hence, on this stripe over 60% of the cells have a humidity between 40% and 60%. Furthermore, the temperature is at the same cells between 11.9° Celsius and 13.6° Celsius. Hence, S_t is in that small stripe between hills and valley in advantage to the hill preferring species S_h, because of the temperature; and in terms of humidty to S_v, that fits best into the valley.
Additionally, figure 6.32 shows also, that the mountains and some borderland are completely unoccupied by any species.

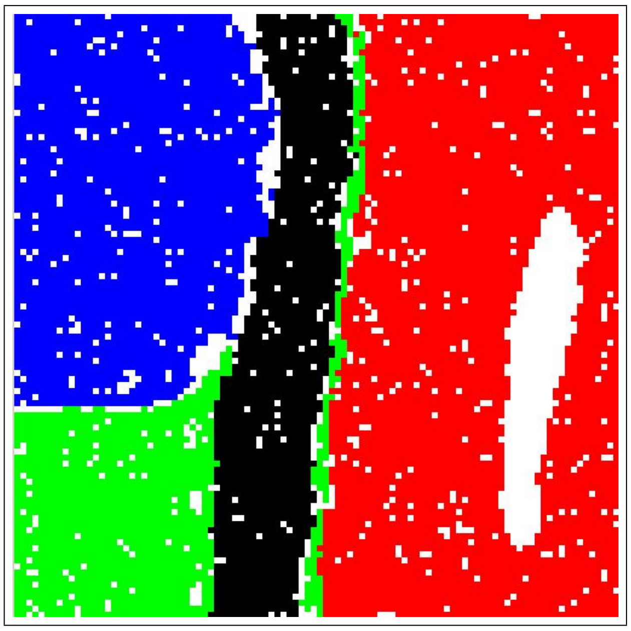


Figure 6.32: Vegetation distribution at time step 50 (blue: S_d; green: S_t; red: S_h; black: S_t)

Experiment two: Wind borne seeding in a heterogeneous landscape

Testing the wind – which is implemented in the model – gives a reason to repeat experiment one, but with a very low *setup ratio* for all species. Hence, the chances are not high that a plant of a species in its preferred area starts by initialization of a simulation run.
A *setup ratio* of 0.0002 is applied for all species. All other conditions are the same as in experiment one, except of the wind. Each mature plant gets the chance to send one seed per period with the wind. To compare the results, the experiment is repeated, but without wind.
Each run, where a species survives for 100 periods is counted (see table 6.26). For each of both settings – with and without wind – 50 runs are executed.

	S_d	S_t	S_h	S_v
with wind	48	44	47	42
without wind	19	16	33	12

Table 6.26: Species survival in 50 runs, experiment with wind

The advantage of the possibility to use the wind is enormous. The survival of the plants is much higher with wind, than without assuming remote breeding. S_t and particularly S_v seem to be the species with the lowest success. This is reasoned by the size of areas that they are specialized for (see figure 6.31).
Another interesting observation of experiment two is the occupation of free cells with plants of "foreign species", if the "domestic species" is completely dead. For example, in some runs S_h does not survive and S_t starts to breed on some cells in the middle of the hilly area. However, if both plants survive S_t has no

chances to populate the hills.
If S_h survives, but S_t does not, the opposite happens. S_h occupies some cells of the area in the south west. That behavior is reasoned by the similarity of both species. They are more specialized for their own area, but if the other species is not existent any more, they have chances, because of the lack of competition. This effect also depends on the *standard deviations* for *humidity* and *temperature.* The higher the values for these parameters, the more likely the considered species occupies cells in a foreign region. On the other hand, the smaller the value for both parameters, the more effective a species defends its region against invaders.

Experiment three: Evolution in a heterogeneous landscape

The results from the last experiment – considering the possibility for wind borne seeding – lead to the idea for a test design regarding the trade-off between specialization and diversification in a heterogeneous landscape. Therefore only two species are necessary. Table 6.27 shows a specialized species (S_s) and a relatively diversified (or non-specialized) species (S_n). The species are quite similar, but S_s is more specialized regarding the temperature. S_n is characterized by a *temperature standard deviation* of 2.5° Celsius, but S_s has a value of 0.05° Celsius. for this parameter. The maps from experiment one and two are used for this experiment. Therefore S_s will probably populate the hilly area. This species has similar preferences as S_h from the experiments one and two[13]. S_n does not fit in a certain region on the map. However, with its relative high *standard deviation* for *temperature* is could have a chance to compete with S_s.

	S_s	S_n
Setup Ratio	0.03	0.03
Humidity Mean in%	0.3	0.3
Humidity Standard Deviation in%	0.05	0.05
Temperature Mean in ° Celsius	4.0	8.0
Temperature Standard Deviation in ° Celsius	0.05	2.5
Maturity Mean in periods	6.0	6.0
Maturity Standard Deviation in periods	0.05	0.05
Life Expectancy Mean in periods	30.0	30.0
Life Expectancy Standard Deviation in periods	1.0	1.0
Competition Strength	10.0	10.0

Table 6.27: Species parameter for experiment three

S_n survived in the hilly area in all of the 50 runs with these setting for more than 100 periods. The number of plants from S_n at time step 100 in the hilly area is relative small (on average 788.7) compared with the number of entities from S_s (on average 2744.3). Hence S_s is predominant in the hilly region. However, the benefit of diversification is obvious. Other regions can be achieved additionally. So occupies S_n also large parts of the area in the south east.
Another 50 runs with the same species and conditions, but a warmer temperature in the hills (an increase by 4° Celsius to 8° Celsius) leads to a prevail of

[13]Both species (S_s and S_h) are characterized by the same values for the *mean* of *humidity* and *temperature.*

S_n. S_s does not survive for 100 periods in 88% of the 50 runs.
In summary, it is possible to argue that the advantages far outweigh the disadvantages of specialization, if the considered species can find a region with its preferred conditions. On the other hand, is diversification of species much more useful, if such conditions are not existent.

Experiment four: Climate change in a heterogeneous landscape

The possibility to change the conditions of the environment, while a simulation runs, is implemented in the model and an experiment can possibly discover some new behavior. The heterogeneous landscape that has been designed for the last three experiments can also be applied for this test. The species from the first experiment (see table 6.25) are also useful.
The first test assumes a warming of all regions. Therefore an increase of 3.0° Celsius is simulated steadily over the time of 300 periods. So a warming of 0.01° Celsius per period characterizes the regions of the map. Wind borne seeding is allowed.
During the first 50 periods the simulation run behaves not differently from experiment one. However, thereafter S_t invades some parts of the hilly area, but the hills seem to be too arid to allow S_t an occupation of large amounts of space in this region. More significant is the propagation of S_h. This species occupies almost 30% of the mountains after 100 time steps. Only an increase of 1.0° Celsius occurred so far. At time step 300 S_h has populated more than 85% of the mountain area.

The same experiment will be repeated with an unlimited increase of humidity, additionally. The humidity increase is 0.1% per period. The temperature increase is also without limit. That mean that the increase will continue until the defined limit of 23.2° Celsius. One would expect that in a long term S_v will prevail. This species prefers hot and humid environments (see table 6.25).
Figure 6.33 shows that the propagation of S_v is not as significant as other observations. The invading of S_h into the mountain area is more obviously; as well as the spreading of S_t into the hills. The most disadvantaged species by the environmental change is S_d. It loses its biosphere because of the increase of humidity, not the warming is the reason for the decreasing number of this species.

The last test regarding the verification of the model is an experiment with decreasing temperature and humidity. Therefore the last test setting is used, but the environment cools by 0.01° Celsius period and it becomes also more arid (0.1% per period).
Figure 6.34 shows the vegetation distribution after 300 periods. The not-vegetated mountain area enlarges. The number of plants of S_v and S_t declines. S_h displaces S_t almost completely from the south west. This is not surprising, because the warmth preferring species (S_v and S_t) need also humidity. An exception is S_d. This species needs warm and arid conditions. Hence, its number declines because of the cooling. S_v and S_h are affected most extremely – one negatively and the other positively – from the decrease of temperature and humidity. S_v becomes almost extinct, but S_h is propagating successfully.

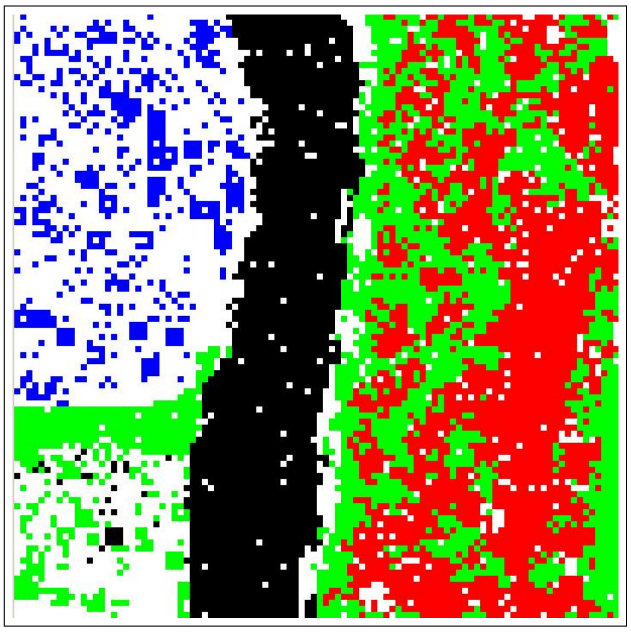


Figure 6.33: Vegetation distribution with increasing temperature and humidity at time step 300 (blue: S_d; green: S_t; red: S_h; black: S_t)

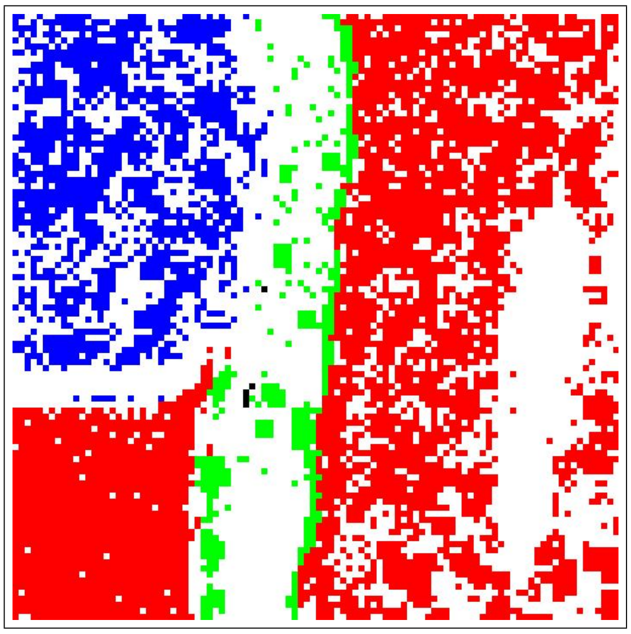


Figure 6.34: Vegetation distribution with decreasing temperature and humidity at time step 300 (blue: S_d; green: S_t; red: S_h; black: S_t)

Chapter 7

Criticism and conclusion

The computer-based simulation model for grassland in the Southern Island of New Zealand has been explored and some interesting details discovered.
However, computer-based simulation is only a research method and has some disadvantages that should be mentioned – in particular against the backdrop of the fact that probably users with little or no experiences in computer-based simulation will apply the model – to facilitate the right use.
From a certain point of view computer-based simulation seems to be outlandish, but in the same way fascinating. To develop a little universe consisting of plain information, and consequently trying to rebuild a part of the real world, which consists of matter and energy, or sometimes even living organisms, is in the first view very peculiar. However, the field of information science provides powerful capabilities, and results from simulation models are often (surprisingly) exact. One indicator for this are the enormous research activity. Another one is the numerous attempts in business to support decision-making, even in conservative fields like finance.
However, for the interpretation of results it must always be kept in mind that, if a model is applied, the results are always influenced by it[1]. Comparison with observations of reality is important, even though data is rarely availability for vegetation, in particular for longer terms.
The critique of computer-based simulation approaches applies basically to all computer-based simulation models and therefore for the general grassland distribution model for the Southern Island of New Zealand, too.
However, the model for New Zealand is developed on a conceptual base. Its research results are exclusively from an academic interest until an application with empirical data occurs.
Because of the detailed verification, the model's dynamics are documented. Behavior of population – like the establishment of plants at locations very close to each other – are natural and can also be noticed in the real world. Also the observation that a population of plants cannot grow anymore when they are surrounded by plants that are too young to breed can be reconsidered by botany scientists.

[1]A sound and basic example for the influence of each computer-based simulation model on its result is the fact, that random values – and these are very often used – are not random in a strict sense. They are always influenced by the deterministic algorithms of finite-state machines and therefore they have periodical properties (Hellekalek 1998).

The ability to send seedlings to distant locations changes this behavior. What is called *wind borne seeding* allows the population of remote places. The advantages are shown by the results of the experiments. In nature plenty of plant species are able to send their seeds with wind, water, or by animal species to populate new areas and this has also strong advantages on a regional base (Soons and Ozinga 2005). The developed model shows exactly this behavior.
The assumption that each individual of a species is different from the other plants of same the species brings useful and interesting dynamics into the model's behavior. An adaption of the species to environmental conditions is possible and therefore properties of evolution are included. Two extremes can be found in the experiments' results. One is, that specialization has the advantage to populate regions with appropriate environmental conditions effectively. In this case it is possible to talk about the "survival of the fittest". On the other hand, the population of unsuitable environments is easier for species that are not specialized. This is particularly important in a changing environment.
Specialization and generalizations regarding the environment in the simulation model is determined by the standard deviation of temperature and humidity.

Further research opportunities are given by an application of empirical data. Additionally further development of the model is desired, so different specialized successors of the model can be developed. This can only happen, if the mentioned empirical data is available and therefore an involvement of botanists or ecologists is essential.
Propagation of invading species can be examined by simulate them and synchronously considering indigenous plants. Therefore the model is eligible as invasion model (see section 3.2.5).
The simulation model developed in this thesis should enhance further development. That means further research with empirical data as input for this model, but also advancement of the model itself by using it and enhancing its functions. Further development of the model for more specialized applications is enabled because of the documentation.
In chapter 3 a lack of application of simulation models in some fields of botany, in particular grassland vegetation, has been shown. However, vegetation modeling is an appropriate field for computer-based simulation. Empirical research as input for such models is possible. The effort for collecting empirical data seems to be very large in botany[2]. However, the scientific gain is relatively high. Social aspects, by comparison, are always in some manner vague. Hence, compensation by assumptions are often necessary. Of course, vegetation modeling also needs some assumptions, but plants characteristics seem not to be as complicated as social interactions. Latent variables seem to be no big issue for botany. Therefore it might be possible to compensate the assumptions by detailed research.
Another – and more important – argument for the fit of botany and simulation science is the scaling of variables in botany. The quality of (almost) all variables can be scaled highly. The data collection methods allow this to be achieved. In social science often interviews are applied as method for data collection and the answers of the important questions are often ordinally scaled. Plants responses

[2]For details of data collection see some studies of botany (Smale, Ross, and Arnold 2005), (Sullivan, Timmins, and Williams 2005), (Dickinson, Mark, and Lee 1992).

to experiments and data from observations can be encoded metrically. This provides a high data quality and that is an indispensable condition for reliable and valid research.

In conclusion vegetation research is a field with plenty of opportunities to apply computer-based simulation. Both fields, computer science and botany would benefit from it.

Acknowledgments

I would like to express my appreciation to both of my supervisors. The project to develop a simulation model would have never been possible without the initiative, support, and ideas from Prof. Dr. Peter Whigham, from Dunedin (New Zealand) and Prof. Dr. Klaus G. Troitzsch, from Koblenz (Germany).

To Verena Götz – my girlfriend – from Koblenz (Germany) I would like to express my gratitude. She helped to check all the table contents and moreover I thank Verena very much for her patience during the last months.

I am very grateful to Peter Whigham and Ute Riechert, from Koblenz (Germany) for spell-checking and grammar correction.

Of course, I have to be blamed for all mistakes left inside the thesis.

Michael Zaggl

Bibliography

Abderhalden, E. and J. Schmidt (1924). *Handbuch der biologischen Arbeitsmethoden.* München: Urban & Schwarzenberg.

Acevedo, M. and J. Raventòs (2002). Growth dynamics of three tropical savanna grass species: an individual-module model. *Ecological Modelling 154*(1), 45–60.

Anselin, L. (1992). *Spatial data analysis with GIS: An introduction to application in the social sciences.* Santa Barbara, CA: NCGIA. p. 1092.

Aussenac, G. (2000). Interactions between forest stands and microclimate: Ecophysiological aspects and consequences for silviculture. *Annals of Forest Science 57*(3), 287–301.

Berk, R. (2003). *Regression Analysis: A Constructive Critique.* London: Sage.

Birdsall, C. and A. Langdon (2004). *Plasma physics via computer simulation.* London: Taylor&Francis.

Bolker, B. and S. Pacala (1999). Spatial moment equations for plant competition: Understanding spatial strategies and the advantages of short dispersal. *The American Naturalist 153*(6), 575–602.

Botkin, D., J. Janak, and J. Wallis (1972a). Rationale, limitations, and assumptions of a northeastern forest growth simulator. *IBM Journal of Research and Development 16*(2), 101–116.

Botkin, D., J. Janak, and J. Wallis (1972b). Some ecological consequences of a computer model of forest growth. *The Journal of Ecology 60*(3), 849–872.

Burke, I., W. Lauenroth, M. Vinton, P. Hook, R. Kelly, H. Epstein, M. Aguiar, M. Robles, M. Aguilera, K. Murphy, and R. Gill (1998). Plant-soil interactions in temperate grasslands. *Biogeochemistry 42*(1&2), 121–143.

Calder, J., J. Wilson, A. Mark, and G. Ward (1992). Fire, succession and reserve management in a new zealand snow tussock grassland. *Biological Conservation 62*(1), 35–45.

Cowling, R., D. Richardson, and S. Pierce (1997). *Vegetation of southern Africa.* Cambridge: Cambridge University Press.

Davis, M. (1982). *Computability and Unsolvability.* New York: Dover Publication.

Deutschman, D., C. Devine, and L. Buttel (2000). The role of visualization in understanding a complex forest simulation model. *Computer Graphics 34*(1), 51–55.

Deutschman, D., S. Levin, C. Devine, and L. Buttel (1997). Scaling from trees to forests: Analysis of a complex simulation model. *Science Online 277*(5332), 1688. http://www.sciencemag.org/feature/data/deutschman/index.htm.

DeVelice, R. (1988). Test of a forest dynamics simulator in new zealand. *New Zealand Journal of Botany 26*, 387–392.

Dickinson, K., A. Mark, and W. Lee (1992). Long-term monitoring of non-forest communities for biological conservation. *New Zealand Journal of Botany 30*(2), 163–179.

Dijkstra, J., H. Timmermans, and A. Jessurun (2000). A multi-agent cellular automata system for visualising simulated pedestrian activity. In S. Bandini and S. Worsch (Eds.), *Theoretical and Practical Issues on Cellular Automata, Proceedings of the Fourth International Conference on Cellular Automata for Research and Industry*, Berlin, pp. 29–36. Springer.

Doyle, T. (1982). Description of forico: a tropical gap dynamics model of the lower montain rain forest of puerto rico. Technical Report 1875, Environmental Sciences Division. Environmental Sciences Division Publication.

Doyle, T., H. Shugart, and D. West (1982). Forico: gap dynamics model of the lower montane rain forest in puerto rico. Technical report, Oak Ridge National Lab., TN (USA).

Eddy, D. (2007). Linking electronic medical records to large-scale simulation models: Can we put rapid learning on turbo? *Health Affairs 26*(2), 125–136.

Emanuel, W., H. Shugart, and M. Stewenson (1985). Climatic change and the broad-scale distribution of terrestrial ecosystem complexes. *Climatic Change 7*(1), 29–43.

Engel, A., M. Möhring, and K. Troitzsch (1994). *Sozialwissenschaftliche Datenanalyse.* http://kola.opus.hbz-nrw.de/volltexte/2004/17/pdf/sdabuch.pdf.

Ermentrout, G. and L. Edelstein-Keshet (1993). Cellular automata approaches to biological modeling. *Journal of Theoretical Biology 160*(1), 97–133.

Esdorn, I. (1961). *Die Nutzpflanzen der Tropen und Subtropen in der Weltwirtschaft.* Jena: Fischer.

Fenner, M. and W. Lee (2001). Lack of pre-dispersal seed predators in introduced asteraceae in new zealand. *New Zealand Journal of Ecology 25*(1), 95–99.

Fetcher, N. and G. Shaver (1983). Life histories of tillers of eriophorum vaginatum in relation to tundra disturbance. *The Journal of Ecology 71*(1), 131–147.

Feynman, R. (1982). Simulating physics with computers. *International Journal of Theoretical Physics 21*(6&7), 467–488.

Flache, A. and R. Hegselmann (2001). Do irregular grids make a difference? relaxing the spatial regularity assumption in cellular models of social dynamics. *Journal of Artificial Societies and Social Simulation 4*(4), http://www.soc.surrey.ac.uk/JASSS/4/4/6.html.

Flanagan, D. (2005). *Java in a nutshell.* Beijing: O'Reilly.

Focks, D., E. Daniels, D. Haile, and J. Keesling (1995). A simulation model of the epidemiology of urban dengue fever: Literature analysis, model development, preliminary validation, and samples of simulation results. *The American Journal of Tropical Medicine and Hygiene 53*(5), 489–506.

Gates, D. (1980). *Biophysical Ecology.* Berlin: Springer.

Georgrii, H. (2004). *Stochastik: Einführung in die Wahrscheinlichkeitstheorie und Statistik* (2 ed.). Berlin: de Gruyter.

Gilbert, N. and K. Troitzsch (1999). *Simulation for the Social Scientist.* Buckingham: Open University Press.

Goldberg, D. and P. Werner (1983). The effects of size of opening in vegetation and litter cover on seedling establishment of goldenrods (solidago spp.). *Oecologia 60*(2), 149–155.

Gonzalez, R. and R. Woods (2007). *Digital Image Processing* (3 ed.). Upper Saddle River, NJ: Pearson Prentice Hall.

Grimm, V., U. Berger, F. Bastiansen, S. Eliassen, V. Ginot, J. Giske, J. Guss-Custard, T. Grand, S. Heinz, G. Huse, A. Huth, J. Jepsen, C. Jørgenson, W. Mooij, B. Müller, G. Pe'er, C. Piou, S. Railsback, A. Robbins, M. Robbins, E. Rossmanith, N. Rger, E. Strand, S. Souissi, R. Stillman, R. Vabø, U. Visser, and D. DeAnglis (2006). A standard protocol for describing individual-based and agent-based models. *Ecological Modelling 198*(1&2), 115–126.

Grove, P., A. Mark, and K. Dickinson (2002). Vegetation monitoring of recently protected tussock grasslands in the southern south island, new zealand. *Journal of The Royal Society of New Zealand 32*(3), 379–414.

Gutowitz, H. (1991). Introndduction (of cellular automata - theory and experiment). In H. Gutowitz (Ed.), *Cellular Automata - Theory and Experiment*, pp. vii–xiv. Cambridge, MA: MIT Press.

Halloy, S., A. Mark, and K. Dickinson (2001). Management of new zealands terrestrial biodiversity as a complex adaptive system. *Complexity International 8*, 1–19.

Hamann, O. (2001). Demographic studies of three indigenous stand-forming plant taxa (scalesia, opuntia, and bursera) in the galpagos islands, ecuador. *Biodiversity and Conservation 10*(2), 223–250.

Hassink, J. and A. Whitmore (1997). A model of the physical protection of organic matter in soils. *Soil Science Society of America Journal 61*(1), 131–139.

Hastings, A., K. Cuddington, K. Davies, C. Dugaw, S. Elmendorf, A. Freestone, S. Harrison, M. Holland, J. Lambrinos, U. Malvadkar, B. Melbourne, K. Moore, C. Taylor, and D. Thomson (2005). The spatial spread of invasions: New developments in theory and evidence. *Ecology Letters 8*(1), 91–101.

Hellekalek, P. (1998). Don't trust parallel monte carlo! In *Twelfth Workshop on Parallel and Distributed Simulation*, Banff, Canada, pp. 82–89. IEEE Computer Society Technical Comittee on Simulation, ACM, and SCS.

Horn, H., H. Shugart, and D. Urban (1989). Simulators a models of forest dynamics. In J. Roughgarden, R. May, and S. Levin (Eds.), *Perspectives in ecological theory*, pp. 256–267. Princeton: Princeton University Press.

Huhns, M. and M. Singh (1998). *Readings in Agents*. San Francisco: Morgan Kaufmann.

Humphreys, L. (2007). *The evolving science of grassland improvement*. Cambridge: Cambridge University Press.

Kelton, W. (2000). Experimental design for simulation. In J. Joines, R. Barton, K. Kang, and P. Fishwick (Eds.), *Proceedings of the 2000 Winter Simulation Conference*, pp. 32–38.

Kier, L. and T. Witten (2005). Cellular automata models of complex biochemical systems. In D. Bonchev and D. Rouvray (Eds.), *Complexity in chemistry, biology, and ecology*, pp. 237–302. Berlin: Springer.

Kirkby, M. (1999). Landscape modelling at regional to continental scales. In S. Hergarten and H. Neugebauer (Eds.), *Process Modelling and Landform Evolution*. Heidelberg: Springer.

Kirkby, M. (2001, May 24-29). From plot to continent: Reconciling fine and coarse scale erosion models. In D. Scott, R. Mohtar, and G. Steinhardt (Eds.), *Sustaining the global farm*, 10th International Soil Conservation Organization Meeting, West Lafayette, pp. 860–870.

Kirkby, M. and M. McMahon (1999). Medrush and the catsop basin - the lessons learned. *CATENA 37*(3&4), 495–506.

Kleijnen, J. (1998). Experimental design for sensitivity analysis, optimization, and validation of simulation models. In J. Banks (Ed.), *Handbook of Simulation: Principles, Methodology, Advances, Applications, and Practice*, pp. 173–224. New York: Wiley.

Köhler, P., T. Ditzer, R. Ong, and A. Huth (2001). Comparison of measured and modelled growths on permanent plot in sabahs rain forests. *Forest Ecology Management 144*(1-3), 101–111.

Köhler, P. and A. Huth (1998). The effects of tree species grouping in tropical rainforest modelling: Simulation with the individual-based model formind. *Ecological Modelling 109*(3), 301–321.

Lantuèjoul, C. (2001). *Geostatistical simulation: models and algorithms*. Berlin: Springer.

Leishman, M., L. Hughes, K. French, D. Armstrong, and M. Westoby (1992). Seed and seedling biology in relation to modelling vegetation dynamics under global climate change. *Australian Journal of Botany 40*(4&5), 599–613.

Leisnham, P., C. Cameron, and I. Jamieson (2003). Life cycle, survival rates and longevity of an alpine weta hemideina maori (orthoptera: Anostostomatidae) determined using markrecapture analysis. *New Zealand Journal of Ecology 27*(2), 191–200.

Lemaire, G., R. Wilkins, and H. J. (2005). Challenges for grassland science - managing research priorities. *Agriculture, Ecosystems & Environment 108*(2), 99–108.

Lieberman, M., S. Chellamma, B. Varughese, Y. Wang, C. Lent, G. Bernstein, G. Snider, and F. a. Peiris (2002). Quantum-dot cellular automata at a molecular scale. *Annals of the New York Academy of Sciences 960*(MOLECULAR ELECTRONICS II), 225–239.

Lindenmayer, D. (1996). *Wildlife and woodchips: Leadbeater's Possum - A test case for sustainable forestry.* Sydney: UNSW Press.

Lough, T., J. Wilson, A. Mark, and A. Evans (1987). Succession in a new zealand alpine cushion community: a markovian model. *Plant Ecology 71*(3), 129–138.

Mark, A. (1969). Ecology of snow tussocks in the mountain grasslands of new zealand. *Plant Ecology 18*(1-6), 289–306.

Mark, A., G. Scott, F. Sanderson, and P. James (1964). Forest succession on landslide above lake thomson, fiorland. *New Zealand Journal of Botany 2*, 60–89.

Martin, J. (1991). Iron, liebig's law and the greenhouse. *Oceanology 4*(2), 52–55.

Martin, P. (1992). Vegetation responses and feedbacks to climate: a review of models and processes. *Climate Dynamics 8*(4), 201–210.

McPherson, G. and S. DeStefano (2003). *Applied ecology and natural resource management.* Cambridge: Cambridge University Press.

Miller, A. and R. Duncan (2004). The impact of exotic weed competition on a rare new zealand outcrop herb, pachycladon cheesemanii (brassicaceae). *New Zealand Journal of Ecology 27*(1), 113–124.

Miller, C. and D. Urban (1999). A model of surface fire, climate and forest pattern in sierra nevada, california. *Ecological Modelling 114*(2&3), 113–135.

Miller, J., J. Franklin, and R. Aspinall (2007). Incorporating spatial dependence in predictive vegetation models. *Ecological Modelling 202*(3&4), 225–242.

Moore, A. and I. Noble (1990). An individualistic model of vegetation stand dynamic. *Journal of Environmental Management 31*(1), 61–81.

Münzbergová, Z., M. Křivaneká, A. Bucharová, J. Juklička, and T. Herben (2005). Ramet performance in two tussock plants do the tussock-level parameters matter? *Flora 200*(3), 275284.

Neilson, R. (1992). Toward a rule based biome model. *Landscape Ecology 7*(1), 27–43.

Neilson, R. (1995). A model for predicting continental-scale vegetation distribution and water balance. *Ecological Applications 5*(2), 362–385.

Neilson, R. and D. Marks (1994). A global perspective of regional vegetation and hydrologic sensitivities from climatic change. *Journal of Vegetation Science 5*(5), 715–730.

NIWA (2008). National institute of water & atmospheric research. http://www.niwascience.co.nz/.

Nobel, P. (2003). *Environmental Biology of Agaves and Cacti.* Cambridge: Cambridge University Press. p. 67.

Noble, I., H. Shugart, and J. Schauer (1980). *Description of BRIND, a computer model of succession and fire response of the high altitude Eucalyptus forests of the Brindabella Range, Australian Capital Territory.* Oak Ridge National Lab.

O'Connor, K. (1982). The implications of past exploitation and current developments to the conservation of south island tussock grasslands. *New Zealand Journal of Ecology 5*, 97–107.

Ord, T. and T. Kieu (2003). On the existence of a new family of diophantine equations for ω. *Fundamenta Informaticae 56*(3), 273–284.

Pacala, S., C. Canham, J. Saponara, J. Silander, R. Kobe, and E. Ribbens (1996). Forest models defined by field measurements: Estimation, error analysis and dynamics. *Ecological Monographs 66*(1), 1–43.

Pacala, S. and D. Deutschman (1995). Details that matter: The spatial distribution of individual trees maintains forest ecosystem function. *Oikos 74*(3), 357–365.

Patel, A., E. Gawlinski, S. Lemieux, and R. Gatenby (2001). A cellular automaton model of early tumor growth and invasion: The effects of native tissue vascularity and increased anaerobic tumor metabolism. *Journal of Theoretical Biology 213*(3), 315–331.

Pausas J., Austin, M. and I. Noble (1997). A forest simulation model for predicting eucalypt dynamics and habitat quality for arboreal marsupials. *Ecological Applications 7*(3), 921–933.

Pavlidou, V., J. Kuijpers, L. Vlahos, and H. Isliker (2001). A cellular automaton model for the magnetic activity in accretion discs. *Astronomy and Astrophysics 372*(1), 326–337.

Perrings, C. (1995). *Biodiversity loss: Economic and ecological issues.* Cambridge: Cambridge University Press.

Perry, G. and N. Enright (2006). Spatial modelling of vegetation change in dynamic landscape: a review of methods and applications. *Progress in Physical Geography 30*(1), 47–72.

Podlich, D. and M. Cooper (1998). Qu-gene: a simulation platform for quantitative analysis of genetic models. *Bioinformatics 14*(7), 632–653.

Polack, F. and S. Stepney (2005). Emergent properties do not refine. *Electronic Notes in Theoretical Computer Science 137*(2), 163–181.

Raines, G., M. Zientek, J. Causey, and D. Boleneus (2002). Preliminary cellular-automata forecast of permit activity from 1998 to 2010, idaho and western montana. *Natural Resources Research 11*(3), 167–186.

Rodriguez Iglesias, R. and M. Kothmann (1997). Structure and causes of vegetation change in state and transition model applications. *The Journal of Range Management 50*(4), 399–408.

Rose, A. and K. Platt (1987). Recovery of northern fiordland alpine grasslands after reduction in the deer population. *New Zealand Journal of Ecology 10*, 23–33.

Rutter, A., A. Morton, and P. Robins (1975). A predictive model of rainfall interception in forests. ii. generalization of the model and comparison with observations in some coniferous and hardwood stands. *The Journal of Applied Ecology 12*(1), 367–380.

Sackville Hamilton, L. (2002). Measurement of competition and competition effects on pastures. In P. Tow and A. Lazenby (Eds.), *Competition and succession in pastures*, pp. 15–42. New York: CAB International.

Samaschke, K. (2004). *Java: Einstieg für Anspruchsvolle*. München: Pearson.

Shugart, H. (1984). *A Theory of Forest Dynamics*. New York: Springer.

Shugart, H. (1987). Dynamic ecosystem consequences of tree birth and death patterns. *BioScience 37*(8), 596–602.

Shugart, H. (1998). *Terrestrial ecosystems in changing environments*. Cambridge: Cambridge University Press.

Shugart, H., A. Mortlock, M. Hopkins, and I. Burgess (1980). Computer simulation model of ecological succession in australian subtropical rainforest. environmental sciences division publication no. 1407. Technical report, Oak Ridge National Lab., TN (USA).

Shugart, H. and I. Prentice (1992). Individual-tree-based models of forest dynamics and their application in global change research. In H. Shugart, R. Leemans, and B. G.B. (Eds.), *A Systems Analysis of the Global Boreal Forest*, pp. 313–333. Cambridge: Cambridge University Press.

Shugart, H., T. Smith, and W. Post (1992). The potential for application of individual-based simulation models for assessing the effects of global change. *Annual Review Of Ecology And Systematics 23*(1), 15–38.

Silvertown, J. and M. Tremlett (1989). Interactive effects of disturbance and shade upon colonization of grassland: An experiment with anthriscus sylvestris (l.) hoffm., conium maculatum l., daucus carota l. and heracleum sphondylium l. *Functional Ecology 3*(2), 229–235.

Sklar, F. and R. Costanza (1991). The development of dynamic spatial models for landscape ecology: a review and prognosis. In M. Turner and R. Gardner (Eds.), *Quantitative methods in landscape ecology. The analysis and interpretation of landscape heterogeneity*, pp. 239–288. New York: Springer.

Smale, M., C. Ross, and G. Arnold (2005). Vegetation recovery in rural kahikatea (dacrycarpus dacrydioides) forest fragments in the waikato region, new zealand, following retirement from grazing. *New Zealand Journal of Ecology 29*(2), 261–269.

Soons, M. and W. Ozinga (2005). How important is long-distance seed dispersal for the regional survival of plant species? *Diversity & Distributions 11*(2), 165–172.

Stephenson, N. (1990). Climatic control of vegetation distribution: The role of water balance. *American Naturalist 135*(5), 659–670.

Stewart, I. (1990). *Does God Play Dice?: The Mathematics of Chaos*. Oxford: Blackwell.

Stringham, T., W. Krüger, and P. Shaver (2003). State and transition modeling: An ecological process approach. *Journal of Range Management 56*(2), 106–113.

Sullivan, J., S. Timmins, and P. Williams (2005). Movement of exotic plants into coastal native forests from gardens in northern new zealand. *New Zealand Journal of Ecology 29*(1), 1–10.

Terradas, J. (2005). Forest dynamics: A broad view of the evolution of the topic, including some recent regional contributions. *Invest Agrar Sist Recur For 14*(3), 525–537.

Thornley, J. (1998). *Grassland dynamics: An Ecosystem simulation model.* Wallingford: CAB International.

Toffoli, T. and N. Margolus (1987). *Cellular Automata Machines - A new Environment for Modeling.* Cambridge, MA: MIT Press.

Toffoli, T. and N. Margolus (1994). Invertible cellular automata: A review. *Physica D: Nonlinear Phenomena 45*(1-3), 229–253.

Tomlinson, K., J. Dominy, J. Hearne, and T. OConnor (2007). A functional-structural model for growth of clonal bunchgrasses. *Ecological Modelling 202*(3&4), 243–264.

Troitzsch, K. (1990). *Modellbildung und Simulation in den Sozialwissenschaften.* Opladen: Westdeutscher Verlag.

Urban, D., G. Bonan, T. Smith, and H. Shugart (1991). Spatial applications for gap models. *Forest Ecology and Management 42*(1&2), 95–110.

van Daalen, J. and H. Shugart (1989). Outeniqua a computer model to simulate succession in the mixed evergreen forests of the southern cape, south africa. *Landscape Ecology 2*(4), 255–267.

Vanclay, J. (1995). Growths models for tropical forests: A synthesis of models and methods. *Forest Science 41*(1), 7–24.

Walker, P. and K. Cocks (1991). Habitat: A procedure for modelling a disjoint environmental envelope for a plant or animal species. *Global Ecology and Biogeography Letters 1*(4), 108–118.

Wang, J., M. van Ginkel, D. Podlich, G. Ye, R. Trethowan, W. Pfeiffer, I. DeLacy, M. Cooper, and S. Rajaram (2003). Comparison of two breeding strategies by computer simulation. *Crop Science 43*(5), 17641773.

Westoby, M., B. Walker, and I. Noy-Meir (1989). Opportunistic management for rangelands not at equilibrium. *Journal of Range Management 42*(4), 266–274.

Winglee, R., G. Dulk, P. Bornmann, and J. Brown (1991). Interrelation of soft and hard x-ray emissions during solar flares: Ii - simulation model. *The Astrophysical Journal 375*(1), 382–402.

Winkler, E. and S. Klotz (1997). Long-term control of species abundances in a dry grassland: spatially explicit model. *Journal of Vegetation Science 8*(2), 189–198.

Wolfram, S. (1998). Cellular automata as models of complexity. In L. Lam (Ed.), *Nonlinear Physics for Beginners: Fractals, Chaos, Solitons, Pattern Formation, Cellular Automata, Complex Systems*, pp. 197–202. London: World Scientific.

Woodward, F. (1987). *Climate and plant distribution.* Cambridge: Cambridge University Press.

Woodward, F. and L. Rochefort (1991). Sensitivity analysis of vegetation diversity to environmental change. *Global Ecology and Biogeography Letters 1*(1), 7–23.

Wooldridge, M. and N. Jennings (1995). Intelligent agents: theory and practice. *Knowledge Engineering Review 10*(2), 115–152.

Zahar, E. (2007). Falsifiability. In H. Keuth (Ed.), *Karl Popper, Logik der Forschung* (3rd ed.)., pp. 103–124. Berlin: Akademie Verlag.

Zhang, Q., Y. Sato, J. Takahashi, K. Muraoka, and N. Chiba (1999). Simple cellular automaton-based simulation of ink behaviour and its application to suibokuga-like 3d rendering of trees. *The Journal of Visualization and Computer Animation 10*(1), 27–37.

Appendix A

Source code

```
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.Dimension;

import javax.swing.JButton;
import javax.swing.JComponent;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JCheckBox;
import javax.swing.JSlider;

import java.awt.Image;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;


public class TussockSimulator extends JComponent
        implements ActionListener, Runnable {

            private static final long serialVersionUID
                = -681953814763757147L;



//BASIC CONFIGURATION:

boolean dynamicEnvironmentYes = false;
```

[1]The source code is available in digital form. Contact: mzaggl@uni-koblenz.de

```
//Should the environment be dynamic (or static) ??

int amountOfWindBorneSeedings = 0;

boolean competitionMatrixYes = false;
//Should the matrix competition be applied
//  (or the individual values) ??



            int x, y;
                // number of cells in x- and y-direction
            int stepCounter;
                // Counts the Steps

            Cell[][] grid;

            Species[] existingSpecies;
                //Array of all existing species

            Counter[] speciesCounter;

        JButton buttonChangeView =
                new JButton("  Vegetation(1/3)  ");
        JButton buttonSymbols =
                new JButton("Symbols on");
        JButton buttonOneStep =
                new JButton("OneStep");
        JCheckBox checkBoxGraphic =
                new JCheckBox("Show Animation", true);
        JCheckBox checkBoxPlot =
                new JCheckBox("Show Plot", true);
        JCheckBox checkBoxHistogram =
                new JCheckBox("Show Histogram", true);
        JSlider sliderSpeed =
                new JSlider(JSlider.HORIZONTAL, 0,
                                        300, 100);

        int viewLevel = 1;
                //which view, Plant (1) or Humidity (2),..
        boolean symbols = false;
                //Symbols for plants will be shown, or not

            JFrame mainFrame =
                new JFrame("Tussock Simulation 1.0");
        JLabel labelCounter =
                new JLabel("Step Count: "+stepCounter
                                            +"  ");
        JPanel panelSpeciesCounter =
                new JPanel();
```

```
Plot plotSpecies;
Histogram histogram;

FileOperator fileOperatorInTussockSimulator =
        new FileOperator();
boolean useTemperatureMap, useHumidityMap;

OutputCreator output = new OutputCreator();
String speciesCounterForOutput = "";

String path = "C:\\Programme\\TussockSimulator";

    Thread thread;          // Thread for running
    int sleep_time;


    public TussockSimulator(int x, int y) {

        this.x = x;
        this.y = y;

        grid = new Cell[x][y];

        //Create Lattice:

        SetupEnvironmentControl readPicture =
                new SetupEnvironmentControl();

        PictureView picTemperature =
                new PictureView
                (readPicture.
                getPathTemperature());
        PictureView picHumidity =
                new PictureView
                (readPicture.getPathHumidity());

        this.checkUsageOfMaps();

            for (int i = 0; i < x; i++) {
            for (int j = 0; j < y; j++) {

                Cell a = new Cell(i, j,
                        useTemperatureMap,
                        useHumidityMap,
                        picTemperature,
                        picHumidity);

                grid[i][j] = a;

                if(grid[i][j] == null) System.err
                .println ("Error in TussockSimul
                ator.java!!! "+i+" "+j+" Empty
                array field.");

            }
            }

            //Create Species:
            this.createGenericSpecies();

        //Create Counters:
                speciesCounter =
                new Counter
                [existingSpecies.length];

                for(int k = 0;
                k < existingSpecies.length;
                 k++) {

                        speciesCounter[k] =
                new Counter(existingSpecies[k]);

                }

                plotSpecies =
                new Plot(existingSpecies);
                histogram = new Histogram();
                histogram.setHistogram(grid);

        this.setFirstConfiguration();

    }

    public void setFirstConfiguration() {

                stepCounter = 0;

        //Finding Candidates for Cells;
        for (int i = 0; i < x; i++) {
            for (int j = 0; j < y; j++) {

                grid[i][j].setSpecies(null);

                for(int k = 0 ;
                k < existingSpecies.length;
                k++) {

                    if(Math.random() <
                    existingSpecies[k]
                    .startRatio) {
```

```
//if((i == 74 && j == 74) || (i == 75 && j == 75)) {

                         Species tmp =
                         existingSpecies[k].clone();
                         tmp.initializeSpecies();
                         grid[i][j].addCompetingSpecies
                         (tmp);

                              }
                     }
                 }
             }
             //Competition
             for (int i = 0; i < x; i++) {
                 for (int j = 0; j < y; j++) {

                     if(grid[i][j].isOccupied()


                       if(competitionMatrixYes) {


                         grid[i][j]
                         .competeUsingMatrix();


                        } else {


                         grid[i][j].compete();


                        }

                     }

                 }
             }

             this.count();

             String headline = "StepCounter: "+",";

             for(int i = 0;
             i < existingSpecies.length; i++) {

                     headline +=
                     existingSpecies[i]
                     .getSpeciesName()+",";
```

```
        }
        output.saveOutput(headline+"\n");

    }

        public void createGenericSpecies() {

                String all
                = fileOperatorInTussockSimulator
                .readSpecies();

                if(all != null) {

                  String[] line =
                  all.split(" endline2 ");
                  existingSpecies =
                  new Species[line.length];

                  for(int i=0;i<line.length;i++) {

                          existingSpecies[i] =
                          new Species();

                          String[] column =
                          line[i].split(" ; ");

                          existingSpecies[i]
                          .number = i;
                          existingSpecies[i]
                          .name = column[0];
                          existingSpecies[i]
                          .startRatio = Double
                          .parseDouble
                          (column[1]);
                          existingSpecies[i]
                          .optimalHumidityMean
                          = Double.parseDouble
                          (column[2]);
                          existingSpecies[i]
                          .optimalHumidityStdDev
                          = Double.parseDouble
                          (column[3]);
                          existingSpecies[i]
                          .optimalTemperatureMean
                          = Double.parseDouble
                          (column[4]);
                          existingSpecies[i]
                          .optimalTemperatureStd
                          Dev = Double.parse
                          Double(column[5]);
```

```
                            existingSpecies[i]
                            .maturityMean =
                            Double.parseDouble
                            (column[6]);
                            existingSpecies[i]
                            .maturityStdDev =
                            Double.parseDouble
                            (column[7]);
                            existingSpecies[i]
                            .dieMean = Double
                            .parseDouble(column[8]);
                            existingSpecies[i]
                            .dieStdDev = Double
                            .parseDouble(column[9]);
                            existingSpecies[i]
                            .competitionStrength
                            = Double.parseDouble
                            (column[10]);
existingSpecies[i].specColor
= new Color(Integer.parseInt(column[11]),
Integer.parseInt(column[12]), Integer
.parseInt(column[13]), 255);

          }

        } else {

          System.err.println
          ("No Species configured!");

        }
}

public void checkUsageOfMaps() {

        String all =
        fileOperatorInTussockSimulator
        .readEnvironment();

          if(all != "") {

            String[] column
            = all.split(";");

            if(column[0].charAt(0) == 't')

                {

                useTemperatureMap = true;
```

```
                                } else {

                                useTemperatureMap = false;

                                }

                                if(column[1].charAt(0)
                                == 't') {

                                useHumidityMap = true;

                                } else {

                                useHumidityMap = false;

                                }
                             }
                }

        //Counter:
            public void count() {

                for(int k = 0;
                k < existingSpecies.length ; k++) {

                        int tmpCounter = 0;

                        for (int i = 0; i < x; i++) {
                          for (int j = 0; j < y; j++) {

                            if(grid[i][j].isOccupied()) {

                              if(grid[i][j]
                              .getSpeciesNumber() ==
                              speciesCounter[k]
                              .getSpecies()
                              .getSpeciesNumber()) {

                                          tmpCounter++;

                                   }
                            }
                                }
                        }

                        speciesCounter[k]
                        .setAmountOfSpecies(tmpCounter);

                }
```

```
//Counter Exposure:
int largest = speciesCounter[0]
.getSpecies().getSpeciesNumber();

for(int i = 0;
i < speciesCounter.length; i++) {

  if(speciesCounter[i]
  .getAmountOfSpecies() >
  speciesCounter[largest]
  .getAmountOfSpecies()) {

   largest = speciesCounter[i]
   .getSpecies().getSpeciesNumber();

        }
}

panelSpeciesCounter.removeAll();

for(int i = 0;
i < speciesCounter.length; i++) {

  JLabel tmp =
  new JLabel(speciesCounter[i]
  .getSpecies().getSpeciesName()
  +": "+speciesCounter[i]
  .getAmountOfSpecies());

  tmp.setPreferredSize
  (new Dimension(100, 20));
  tmp.setForeground(speciesCounter[i]
  .getSpecies().getSpeciesColor());

        if(i == largest) {

          tmp.setText(tmp.getText()
          .toUpperCase());

        }

        panelSpeciesCounter.add(tmp);

}
panelSpeciesCounter.validate();

  //Plots:

plotSpecies.setPlot
(stepCounter, speciesCounter);
```

```
            if(checkBoxPlot.isSelected()) {
               plotSpecies.repaint();

            }

            if(checkBoxHistogram
                    .isSelected()) {

               histogram
              .setHistogram(grid);
               histogram.repaint();

            }


}


// change of states.
public void changeConfiguration() {


    // AGEING
    for (int i = 0; i < x; i++) {
            for (int j = 0; j < y; j++) {

              if(grid[i][j].isOccupied()) {

                grid[i][j].aging();

                    }
            }
    }

    //SPREADING:

    double abstractMaturityProbability = 0.0;
    //add. probability, with 0.0 no effects

    for (int i = 0; i < x; i++) {
     for (int j = 0; j < y; j++) {

      int iBorderLeft = (i-1+x)%x;
      int iBorderRight = (i+1)%x;
      int jBorderTop = (j-1+y)%y;
      int jBorderBottum = (j+1)%y;

      if (grid[i][j].isOccupied()
                  == true) {
```

```
if (grid[i][jBorderTop]
.getSeddlingPossibility(grid[i][j]
.getSpecies()) && grid[i][j]
.maturity() && Math.random()
> abstractMaturityProbability) {

  Species tempSpecies = grid[i][j]
  .getSpecies().clone();
  tempSpecies.initializeSpecies();
  grid[i][jBorderTop]
  .addCompetingSpecies(tempSpecies);
  }

 if (grid[iBorderRight][jBorderTop]
 .getSeddlingPossibility(grid[i][j]
 .getSpecies()) && grid[i][j]
 .maturity() && Math.random()
 > abstractMaturityProbability) {

   Species tempSpecies = grid[i][j]
   .getSpecies().clone();
   tempSpecies.initializeSpecies();
   grid[iBorderRight][jBorderTop]
   .addCompetingSpecies(tempSpecies);
   }

  if (grid[iBorderRight][j]
  .getSeddlingPossibility(grid[i][j]
  .getSpecies()) && grid[i][j]
  .maturity() && Math.random()
  > abstractMaturityProbability) {

    Species tempSpecies = grid[i][j]
    .getSpecies().clone();
    tempSpecies.initializeSpecies();
    grid[iBorderRight][j]
    .addCompetingSpecies(tempSpecies);
    }

  if (grid[iBorderRight][jBorderBottum]
  .getSeddlingPossibility(grid[i][j]
  .getSpecies()) && grid[i][j]
  .maturity() && Math.random()
  > abstractMaturityProbability) {

    Species tempSpecies = grid[i][j]
    .getSpecies().clone();
    tempSpecies.initializeSpecies();
```

```
          grid[iBorderRight][jBorderBottum]
          .addCompetingSpecies(tempSpecies);
          }

        if (grid[i][jBorderBottum]
        .getSeddlingPossibility(grid[i][j]
        .getSpecies()) && grid[i][j]
        .maturity() && Math.random()
        > abstractMaturityProbability) {

          Species tempSpecies = grid[i][j]
          .getSpecies().clone();
          tempSpecies.initializeSpecies();
          grid[i][jBorderBottum]
          .addCompetingSpecies(tempSpecies);
          }

        if (grid[iBorderLeft][jBorderBottum]
        .getSeddlingPossibility(grid[i][j]
        .getSpecies()) && grid[i][j]
        .maturity() && Math.random()
        > abstractMaturityProbability) {

          Species tempSpecies = grid[i][j]
          .getSpecies().clone();
          tempSpecies.initializeSpecies();
          grid[iBorderLeft][jBorderBottum]
          .addCompetingSpecies(tempSpecies);
          }

        if (grid[iBorderLeft][j]
        .getSeddlingPossibility(grid[i][j]
        .getSpecies()) && grid[i][j]
        .maturity() && Math.random()
        > abstractMaturityProbability){

          Species tempSpecies = grid[i][j]
          .getSpecies().clone();
          tempSpecies.initializeSpecies();
          grid[iBorderLeft][j]
          .addCompetingSpecies(tempSpecies);
          }

        if (grid[iBorderLeft][jBorderTop]
        .getSeddlingPossibility(grid[i][j]
        .getSpecies()) && grid[i][j]
        .maturity() && Math.random()
        > abstractMaturityProbability){

          Species tempSpecies = grid[i][j]
```

```
    .getSpecies().clone();
    tempSpecies.initializeSpecies();
    grid[iBorderLeft][jBorderTop]
    .addCompetingSpecies(tempSpecies);
}

//wind:
int iterationsForWind = 0;
//amount of remote spreading
int north, south, east, west;

int latitude, longitude;

for (int w = 0;
w < iterationsForWind ; w++) {

    north = (int) (Math.random()*100);
    south = (int) (Math.random()*100);
    west = (int) (Math.random()*100);
    east = (int) (Math.random()*100);

    north = north%100;
    south = south%100;
    west = west%100;
    east = east%100;

    if(north > south) {

        latitude = north - south;

    } else {

        latitude = south - north;

    }

    if(west > east) {

        longitude = west - east;

    } else {

        longitude = east - west;

    }

    if (longitude <= 100
    && longitude >= 0
    && latitude <= 100
    && latitude >= 0) {
```

```
                    if (grid[longitude][latitude]
                    .getSeddlingPossibility(grid[i][j]
                    .getSpecies()) && grid[i][j]
                    .maturity() && Math.random()
                    > abstractMaturityProbability){

                      Species tempSpecies = grid[i][j]
                      .getSpecies().clone();
                      tempSpecies.initializeSpecies();
                      grid[longitude][latitude]
                      .addCompetingSpecies
                      (tempSpecies);

                     }
                   }
                 }


               }
             }
           }

           for (int i = 0; i < x; i++) {
                   for (int j = 0; j < y; j++) {

                           if(grid[i][j].isOccupied()
                           == false) {

                             if(competitionMatrixYes)

                             {

                               grid[i][j]
                               .competeUsingMatrix();

                              } else {

                               grid[i][j].compete();

                              }


                           }
                   }
           }

           //DYING:
           for (int i = 0; i < x; i++) {
                   for (int j = 0; j < y; j++) {
                           grid[i][j].die();

                   }
           }

           //ENVIRONMENT CHANGE
           if(dynamicEnvironmentYes) {

                   double tempHumidity
                    , tempTemperature;

                    for (int i = 0; i < x; i++) {
                      for (int j = 0
                           ; j < y; j++) {

                       tempHumidity = grid[i][j]
                       .getHumidity();
                       //change of Humidity:

                       tempHumidity -= 0.001;

                       grid[i][j].setHumidity
                       (tempHumidity);

                       tempTemperature
                       = grid[i][j]
                       .getTemperature();
                       //change of Temperature:

                       tempTemperature -= 0.01;

                       grid[i][j]
                       .setTemperature
                       (tempTemperature);

                      }
                    }

           }

           //Counters:
           stepCounter++;
   labelCounter.setText
   ("Step Count: "+stepCounter+"   ");
           this.count();
               //updates the Species Counter(s)

           //File Output:
           for(int i = 0;
           i < speciesCounter.length;i++) {

                   speciesCounterForOutput +=
                   speciesCounter[i]
                   .getAmountOfSpecies()+",";

           }

           output.saveOutput(stepCounter+","
           +speciesCounterForOutput);
           speciesCounterForOutput = "";

           //Histogram:
           histogram.setHistogram(grid);

   }

   //simulation process
   public void begin() {
           if(thread == null) {
               thread = new Thread(this);
               thread.start();
           }
   }

   public void oneStep() {
               // stepping
                    this.changeConfiguration();
                       // redrawing
                    super.repaint();
   }

   //break
   public void pause() {

           thread = null;
           buttonOneStep.setEnabled(true);

   }

   // run
   public void run() {

           buttonOneStep.setEnabled(false);

           while (thread != null) {
                   sleep_time = sliderSpeed
                   .getValue();
                   this.oneStep();
               try {
                   Thread.sleep(sleep_time);
               } catch (InterruptedException e) {
                   System.err.println(e);
               }
           }
   }

    public void actionPerformed(ActionEvent ae) {
               String ac = ae.getActionCommand();
               if(ac.equals("Start")) {
                   this.begin();
               } else if(ac.equals("OneStep")) {
                   this.oneStep();
               } else if(ac.equals("Pause")) {
                   this.pause();
               } else if(ac.equals("Reset")) {
                   this.setFirstConfiguration();
                   if(thread == null) {
                       labelCounter.setText
                       ("Step Count: "+stepCounter);
                   }
               this.repaint();
               } else if(ac.equals("End")) {
                   this.pause();
                   System.exit(0);
                     } else if(ac.equals("Setup")) {
                   this.pause();
                   Setup mainSetup = new Setup();
                           mainSetup.draw();
                   } else if(ac.equals
                           ("Symbols on")) {
                           symbols = true;
                           buttonSymbols.setText
                           ("Symbols off");
                           this.repaint();
                   } else if(ac.equals
                           ("Symbols off")) {
                           symbols = false;
                           buttonSymbols.setText
                           ("Symbols on");
                           this.repaint();
               } else if(ac.equals("Print Grid")) {
                   this.pause();
                   OutputCreatorDetailed
                   outputDetailed =
                   new OutputCreatorDetailed(path);
                   outputDetailed
                   .createDetailedOutput
                   (grid, stepCounter);
                   outputDetailed = null;
               //this.begin();
               } else if(ac.equals
               ("     Humidity(2/3)     ")) {
               buttonChangeView.setText
               ("Temperature(3/3)");
               viewLevel = 3;
               this.repaint();
           } else if(ac.equals
               ("Temperature(3/3)")) {
               buttonChangeView.setText
               ("   Vegetation(1/3)   ");
               viewLevel = 1;
               this.repaint();
           } else if(ac.equals
               ("   Vegetation(1/3)   ")) {
               buttonChangeView.setText
               ("     Humidity(2/3)     ");
               viewLevel = 2;
               this.repaint();
           }

   }

public void paintComponent(Graphics g) {

   // Measurement: ds*ds
   int ds = (int) (Math.min((double) super
   .getWidth() / (double) x, (double) super
   .getHeight() / (double) y) + 0.5);

   Color tmp;
   Color backgroundColor =
   new Color(1.0f, 1.0f, 1.0f);
   // all cells ij: coloring

 //Output Vegetation:
 if(viewLevel == 1 && checkBoxGraphic
 .isSelected()) {

   for (int i = 0; i < x; i++) {
           for (int j = 0; j < y; j++) {
             if(grid[i][j].isOccupied()){

               tmp = grid[i][j]
               .getSpeciesColor();
               g.setColor(tmp);
               g.fillRect
               (i * ds, j * ds, ds, ds);

             }
```

```
                if(!grid[i][j].isOccupied()){

                    g.setColor
                    (backgroundColor);
                    g.fillRect
                    (i * ds, j * ds, ds, ds);

                }
            }
        }
    }
    //Output Humidity:
    if(viewLevel == 2) {

     Color tmpColor;
        for (int i = 0; i < x; i++) {
                    for (int j = 0; j < y; j++) {

            tmpColor = new Color
            (0.0f, 0.0f, grid[i][j]
            .getCellsHumidityAsRGB());

            g.setColor(tmpColor);
            g.fillRect(i * ds, j * ds, ds, ds);
                    }
        }
      }
    //Output Temperature:
    if(viewLevel == 3) {

     Color tmpColor;
       for (int i = 0; i < x; i++) {
           for (int j = 0; j < y; j++) {

               tmpColor = new Color(grid[i][j]
               .getCellsTemperatureAsRGB()
               , 0.0f, 0.0f);

               g.setColor(tmpColor);
               g.fillRect
               (i * ds, j * ds, ds, ds);
               }
         }
    }

    if(viewLevel >= 4) {
        System.err.println
        ("Error in TussockSimulator!!!
        The value of the variable view
        ("+viewLevel+") is not defined.");
    }

    //Output Symbols:
    if(symbols) {

        Image[] icon =
        new Image[existingSpecies.length];

        for(int i = 0;
        i < existingSpecies.length; i++) {

          try {

           icon[i] = ImageIO.read
           (new File(path+"\\icons\\"+
           existingSpecies[i]
           .getSpeciesNumber()+".gif"));

          } catch(IOException e) {

             e.printStackTrace(); }

        }
            for (int i = 0; i < x; i++) {
              for (int j = 0; j < y; j++) {

                if(grid[i][j].isOccupied()) {

                 if(icon[grid[i][j]
                 .getSpeciesNumber()]
                 != null) {

                    g.drawImage(icon
                    [grid[i][j]
                    .getSpeciesNumber()],
                    (i * ds)
                    , (j * ds), null);

                  }

             }
        }
   }


  }
  }

  //Frame:
    public JFrame draw(int width, int height) {

        mainFrame.setSize(width, height);
        mainFrame.setDefaultCloseOperation
        (JFrame.EXIT_ON_CLOSE);

        JPanel upperPanel = new JPanel();
        JPanel lowerPanel = new JPanel();
        JPanel rightPanel = new JPanel();
        JPanel sliderPanel = new JPanel();

        //upperPanel (Buttons):
        JButton buttonStart =
        new JButton("Start");
        JButton buttonPause =
        new JButton("Pause");
        JButton buttonReset =
        new JButton("Reset");
        JButton buttonSetup =
        new JButton("Setup");
        JButton buttonEnd  =
        new JButton("End");
        JButton buttonPrintDetailedOutput =
        new JButton("Print Grid");

        //Slider:
        sliderPanel.add(new JLabel
        ("Speed Delay"));
        sliderPanel.add(sliderSpeed);
        sliderSpeed.setMajorTickSpacing(100);
        sliderSpeed.setMinorTickSpacing(25);
        sliderSpeed.setPaintTicks(true);
        sliderSpeed.setPaintLabels(true);

        //upperPanel (Control):
        upperPanel.add(buttonSetup);
        upperPanel.add(buttonStart);
        upperPanel.add(buttonPause);
        upperPanel.add(buttonOneStep);
        //upperPanel.add(buttonReset);
        upperPanel.add(buttonChangeView);
        upperPanel.add(buttonSymbols);
        upperPanel.add(buttonEnd);
        upperPanel.add(checkBoxGraphic);
        upperPanel.add(checkBoxPlot);
        upperPanel.add(checkBoxHistogram);
        upperPanel.setBackground(Color.GRAY);

        //lowerPanel (Counters):
```

```
        lowerPanel.add(sliderPanel);
        lowerPanel.add
        (buttonPrintDetailedOutput);
        lowerPanel.add
        (new JLabel("                    "));
        lowerPanel.add(labelCounter);
        lowerPanel.add(panelSpeciesCounter);
        lowerPanel.setBackground(Color.GRAY);

        //rightPanel:
        rightPanel.setLayout
        (new GridLayout(2, 1));
        rightPanel.setBackground
        (Color.LIGHT_GRAY);
        histogram.setBackground(Color.WHITE);
        rightPanel.add(histogram.getHistogram());

        plotSpecies.setSize
        (plotSpecies.getSizeOfX(), 400);
        plotSpecies.setBackground(Color.WHITE);
        rightPanel.add(plotSpecies.getPlot());

        //Adding everything to the JFrame:
        mainFrame.getContentPane().add
        (this, BorderLayout.CENTER);
        mainFrame.getContentPane().add
        (upperPanel, BorderLayout.NORTH);
        mainFrame.getContentPane().add
        (lowerPanel, BorderLayout.SOUTH);
        mainFrame.getContentPane().add
        (rightPanel, BorderLayout.EAST);

        mainFrame.setVisible(true);

        buttonSetup.addActionListener(this);
        buttonStart.addActionListener(this);
        buttonOneStep.addActionListener(this);
        buttonPause.addActionListener(this);
        buttonReset.addActionListener(this);
        buttonSymbols.addActionListener(this);
        buttonChangeView.addActionListener(this);
        buttonEnd.addActionListener(this);
        buttonPrintDetailedOutput
        .addActionListener(this);

        return mainFrame;


    }
```

```
                public static void main(String[] args) {
                     int height = 100;
                     int width = 100;
                     TussockSimulator mainEntity =
                     new TussockSimulator(width, height);
                     mainEntity.draw(1026, 840)
                     .setVisible(true);

                }
}
```

Listing A.1: TussockSimulator.java

```
import java.awt.Color;
//import utilities.Util;


public class Cell {

        int x;
        int y;

        double humidity;        //0;..;255 [0;..;1]
        double temperature;     //0;..;255 [-2.3;..;+23.2]

        double meanTemperature = 10.4;
        double stdDevTemperature = 3.0;

        CompetitionList competitionListOfThisCell;

        Species occupiedBy;

        public Cell(int x, int y,
        boolean useTemperatureMap, boolean useHumidityMap,
        PictureView temperatureMap,
        PictureView humidityMap) {

                this.x = x;
                this.y = y;

                // corrections:
                x++; y++; int subtractor = 2;

                if(useTemperatureMap == true
                && x <= temperatureMap
                .getScaledImageWidth()
                -subtractor && y <=
                temperatureMap.getScaledImageHeight()
                -subtractor)
                {

                   temperature = convertCellsTemperature
                   InCelsius(temperatureMap
                   .readTemperatureScaledPicture(x, y));

                } else {

                   temperature =
                   convertCellsTemperatureInCelsius
                   ((int)(Math.random() * 255));

                   if(useTemperatureMap == true) {
```

```
                System.out.println("Temperature map
                is smaller than grid (subsituted by
                random value)(x="+x+" y="+y+").");
            }

        /*
        temperature = Util.RAND.random_gaussian
        (meanTemperature, stdDevTemperature);
        while(temperature < 0 || temperature > 20)
        { //assure the spectrum
        temperature = Util.RAND.random_gaussian
        (meanTemperature, stdDevTemperature);
        }
        */

        }

        if(useHumidityMap == true && x <=
        humidityMap.getScaledImageWidth()
        -subtractor && y <=
        humidityMap.getScaledImageHeight()
        -subtractor) {

          humidity =
          convertCellsHumidityToDoubleScale
          (humidityMap.readHumidityScaledPicture
          (x, y));

        } else {

                humidity = Math.random();

                if(useHumidityMap == true) {
                  System.out.println("Humidity
                  map is smaller than grid
                  (subsituted by random value)
                  (x="+x+" y="+y+").");
                }
        }

        //Create competition list for this cell:
        competitionListOfThisCell =
        new CompetitionList();

}

public void setSpecies(Species spec) {

        occupiedBy = spec;
```

```
}

public Species getSpecies() {

        return occupiedBy;

}

public int getSpeciesNumber() {

        return occupiedBy.getSpeciesNumber();
}

public String getSpeciesName() {

        return occupiedBy.name;

}

public Color getSpeciesColor() {

        return occupiedBy.getSpeciesColor();

}

public float getCellsTemperatureAsRGB() {


        float asFloat = (float)(10 *
        temperature + 23);

        return asFloat / 256;

}

public float getCellsHumidityAsRGB() {

        return (float)(humidity);

}

private double convertCellsTemperatureInCelsius
(int temperatureAsRGB) {

        return 0.1 * temperatureAsRGB - 2.3;

}

private double convertCellsHumidityToDoubleScale
```

```
(int humidityAsRGB) {

        double humidityRGBDouble = humidityAsRGB;

        return humidityRGBDouble / 256;

}

public boolean isOccupied() {

        if(occupiedBy == null) {
                return false;
        } else {
                return true;
        }
}

public void aging() {

        occupiedBy.ageing();

}

public boolean getSeddlingPossibility
(Species potentialAncestor) {

        if(this.isOccupied()) {

                return false;

        }

        double lowerLimitPlantsHumidity =
        potentialAncestor.getOptimalHumidity()
        - 0.2;
        double upperLimitPlantsHumidity =
        potentialAncestor.getOptimalHumidity()
        + 0.2;
        double lowerLimitPlantsTemperature =
        potentialAncestor.getOptimalTemperature()
        - 4.0; //in Celsius
        double upperLimitPlantsTemperature =
        potentialAncestor.getOptimalTemperature()
        + 4.0; //in Celsius


        if(!(lowerLimitPlantsHumidity <= this
        .humidity && upperLimitPlantsHumidity
        >= this.humidity)) {
                return false;
        }
        if(!(lowerLimitPlantsTemperature
       <= this.temperature &&
        upperLimitPlantsTemperature >=
        this.temperature)) {

                return false;
        }

        return true;
}

public boolean maturity() {

        if(this.isOccupied()) {
        //this if clause is just for logic,
        actually it is redundant

           return occupiedBy.getMaturity();

        } else {

                return false;
        }

}

public void die() {

    if(this.isOccupied()) {

        if(occupiedBy.getDying() == true) {


                occupiedBy = null;

        }
    }

}

public void addCompetingSpecies
(Species competitor) {

        competitionListOfThisCell
        .addSpecies(competitor);

}
```

```
    public void compete() {

            Species winner;
            int amountOfCandidates =
            competitionListOfThisCell
            .getListLengths();

            switch(amountOfCandidates) {

              case 0:
                      winner = null;
                      break;

              case 1:
                      winner =
                      competitionListOfThisCell
                      .getSpecies(0);
                      break;

              default:
                      Species[] list =
                      competitionListOfThisCell
                      .getSpeciesList();

                      //Competition:

                      winner =
                      competitionListOfThisCell
                      .getSpecies(0);

                      for(int i = 0;
                      i < competitionListOfThisCell
                      .getListLengths(); i++) {

                        if(list[i]
                        .getCompetitionStrength() *
                        Math.random() > winner
                        .getCompetitionStrength() *
                        Math.random() ) {

                         winner =
                         competitionListOfThisCell
                         .getSpecies(i);

                      }
                  }
              }
              occupiedBy = winner;

             /* if(occupiedBy != null) {
                occupiedBy.initializeSpecies();

            }*/

            competitionListOfThisCell.emptyList();
    }


    public double getTemperature() {

            return temperature;

    }

    public void setTemperature
                (double newTemperature) {

            if(newTemperature <= 23.2
                && newTemperature >= -2.3) {

                this.temperature = newTemperature;

            }

    }

    public double getHumidity() {

                return humidity;

    }

    public void setHumidity(double newHumidity) {

            if(newHumidity >= 0.0
                && newHumidity <= 1.0) {

                    this.humidity = newHumidity;

            }

    }


    public void competeUsingMatrix() {

            File matrixSource = new File(path
                    +"\\competitionMatrix.xls");
                Species winner;
                int amountOfCandidates
                    = competitionListOfThisCell
                      .getListLengths();

                switch(amountOfCandidates) {

                  case 0:
                          winner = null;
                          break;

                  case 1:
                          winner =
                          competitionListOfThisCell
                          .getSpecies(0);

                          break;

                  default:

                           try {

                                Workbook matrix =
                                  Workbook
                                  .getWorkbook
                                  (matrixSource);

                                Sheet matrixSheet
                                = matrix.getSheet(0);

                                winner =
                                competitionListOfThisCell
                                .getSpecies(0);

                                int row, column;
                                String fight;

                                for(int i = 0;
                                  i < amountOfCandidates
                                  ;i++) {

                                    row = winner
                                    .getSpeciesNumber()+1;
                                    column =
                                    competitionListOfThis
                                    Cell.getSpecies(i)
                                    .getSpeciesNumber()+1;
                                    fight = matrixSheet
                                    .getCell(column, row)
```

```
                                        .getContents();

                                        if(fight.charAt(0)
                                            == '-') {

                                            winner =
                                            competitionListOf
                                            ThisCell
                                            .getSpecies(i);

                                        }

                                      }

                          } catch(Exception IOE) {

                              System.err.println
                              ("Error reading competition
                                matrix "+ IOE);
                              winner = null;
                          }
                   }
                   occupiedBy = winner;

                   competitionListOfThisCell.emptyList();

         }

}
```

Listing A.2: Cell.java

```
public class CompetitionList {

        int position;
        Species list[];

        public CompetitionList() {

                position = 0;
                list = new Species[9];

        }

        public void addSpecies(Species a) {

                list[position] = a;
                position++;

        }

        public int getListLengths() {

                return position;

        }

        public Species getSpecies(int pos) {

                return list[pos];

        }

        public Species[] getSpeciesList() {

                return list;

        }

        public void emptyList() {

                for(int i=0; i < list.length;i++) {

                        list[i] = null;

                }
                position = 0;
        }
}
```

Listing A.3: CompetitionList.java

```
public class Counter {

        private Species countingSpecies;
        private int amountOfSpecies;

        public Counter(Species countingSpec) {

                countingSpecies = countingSpec;
                amountOfSpecies = 0;

        }

        public void setAmountOfSpecies(int amount) {

                amountOfSpecies = amount;

        }

        public int getAmountOfSpecies() {

                return amountOfSpecies;

        }

        public Species getSpecies() {

                return countingSpecies;

        }

        public void reset() {

                amountOfSpecies = 0;

        }

}
```

Listing A.4: Counter.java

```
import java.io.BufferedReader;
import java.io.File;
import java.io.FileReader;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;


public class FileOperator {

        String path = "C:\\Programme\\TussockSimulator";
        private String fileSpeciesConfiguration
        = "configuration.txt";
        private String fileEnvironmentConfiguration
        = "mapConfig.txt";
        private BufferedWriter writer;
        private BufferedReader reader;
        private File createDir;
        private File createFile;

        public void save(String all) {

        try {
                //Save Configuration to file to harddisk:
            writer = new BufferedWriter(new FileWriter
            (path+"\\"+fileSpeciesConfiguration));
            writer.write(all);

        }

          catch (IOException e) {

                  System.err.println
                  ("Error (FileOperator) writing file.");

                  }

        finally {
            if (writer != null)

                try {writer.close();}

                catch (IOException e) { }

        }

        }

        public String readSpecies() {
                String content = "";

                try {

                  createDir = new File(path);
                  createFile = new File
                  (path+"\\"+fileSpeciesConfiguration);

                        if(!createFile.exists()) {

                           createDir.mkdirs();
                           createFile.createNewFile();

                           this.save("Default ; 0.002 ;
                           0.5 ; 0.2 ; 0.5 ; 0.2 ; 7.0 ;
                           1.0 ; 15.0 ; 2.0 ; 10.0 ; 0 ;
                           0 ; 255 endline2 ");

                           System.out.println("Default
                           Configuration File has been
                           created.");

                        }


            //Read Configuration out of file on harddisk:
            reader = new BufferedReader(new FileReader
            (path+"\\"+fileSpeciesConfiguration));

                    content = reader.readLine();

            }

            catch (IOException e) {


             System.err.println("File not found,
             file creation has been failed.");


            }

        finally {
            if (reader != null)

                try {reader.close();}

                catch (IOException e) { }
        }

        return content;

        }

        public void saveEnvironment
        (boolean mapTemperature, boolean mapHumidity) {

                try {
                        //Save Map-Use to file to disk:
                    writer = new BufferedWriter
                    (new FileWriter(path+"\\"
                    +fileEnvironmentConfiguration));


                    writer.write
                    (mapTemperature+";"+mapHumidity);

                }

                  catch (IOException e) {

                   System.err.println("Error
                   (FileOperator) writing file.");

                }

                finally {
                    if (writer != null)

                        try {writer.close();}

                        catch (IOException e) { }

                }

                }

        public String readEnvironment() {

                String content = "";

                try {

                  createDir = new File(path);
                  createFile = new File(path
                  +"\\"+fileEnvironmentConfiguration);

                        if(!createFile.exists()) {
```

```
                        createDir.mkdirs();

                        createFile.createNewFile();

                        this.save("false;false");

                        System.out.println
                        ("Default Configuration
                        File has been created.");

                    }


            //Read Configuration out of file on harddisk:
            reader = new BufferedReader(new FileReader
            (path+"\\"+fileEnvironmentConfiguration));

                    content = reader.readLine();

            }

            catch (IOException e) {


                    System.err.println
                    ("File not found, file creation
                    has been failed.");


                    }

        finally {
            if (reader != null)

                try {reader.close();}

                catch (IOException e) { }

        }

        return content;

        }

}
```

Listing A.5: FileOperator.java

```
import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;


public class Histogram extends Canvas {

        private static final long serialVersionUID
        = 2054056283306194251L;


        Cell[][] copyOfGrid;

        public void setHistogram(Cell[][] grid) {

                this.copyOfGrid = grid;

        }

         public void paint(Graphics g) {

                 int height = this.getHeight();

                 int age;

                 int smaller5 = 0,  smaller10 = 0
                 , smaller15 = 0, smaller20 = 0
                 , smaller25 = 0 , smaller30 = 0
                 , smaller35 = 0, smaller40 = 0
                 , smaller45 = 0 , smaller50 = 0
                 , higher50 = 0;

                 for(int i=0;i<copyOfGrid.length;i++) {
                         for(int j=0
                              j<copyOfGrid.length;j++) {

                           if(copyOfGrid[i][j]
                              .getSpecies() != null) {

                             age = copyOfGrid[i][j]
                                .getSpecies().getAge();

                                 if(age < 5)
                                  smaller5++;
                                 else if (age < 10)
                                  smaller10++;
                                 else if (age < 15)
                                  smaller15++;
```

```
    else if (age < 20)
     smaller20++;
    else if (age < 25)
     smaller25++;
    else if (age < 30)
     smaller30++;
    else if (age < 35)
     smaller35++;
    else if (age < 40)
     smaller40++;
    else if (age < 45)
     smaller45++;
    else if (age < 50)
     smaller50++;
    else if (age >= 50)
     higher50++;
    }

  }

}

g.setColor(Color.BLACK);

g.drawString
("<5", 28, height -20);

g.drawString
("<10", 49, height -20);

g.drawString
("<15", 74, height -20);

g.drawString
("<20", 99, height -20);

g.drawString
("<25", 124, height -20);

g.drawString
("<30", 149, height -20);

g.drawString
("<35", 174, height -20);

g.drawString
("<40", 199, height -20);

g.drawString
("<45", 224, height -20);
```

```
                              g.drawString
                              ("<50", 249, height-20);

                              g.drawString
                              (">=", 278, height-25);
                              g.drawString
                              ("50", 278, height-15);


                              g.setColor(Color.RED);

                              g.draw3DRect
                              (22, height-smaller5/2,
                              25 , smaller5/2, true);

                              g.draw3DRect
                              (47, height-smaller10/2,
                              25 , smaller10/2, true);

                              g.draw3DRect
                              (72, height-smaller15/2,
                              25 , smaller15/2, true);

                              g.draw3DRect
                              (97, height-smaller20/2,
                              25 , smaller20/2, true);

                              g.draw3DRect
                              (122, height-smaller25/2,
                              25 , smaller25/2, true);

                              g.draw3DRect
                              (147, height-smaller30/2,
                              25, smaller30/2, true);

                              g.draw3DRect
                              (172, height-smaller35/2,
                              25 , smaller35/2, true);

                              g.draw3DRect
                              (197, height-smaller40/2,
                              25 , smaller40/2, true);

                              g.draw3DRect
                              (222, height-smaller45/2,
                              25 , smaller45/2, true);

                              g.draw3DRect
                              (247, height-smaller50/2,
                                25 , smaller50/2, true);

                                g.draw3DRect
                                (272, height-higher50/2,
                                25 , higher50/2, true);


                                if(higher50 >= height) {

                                  g.drawString
                                  (""+higher50, 268, 10);

                                }

                                g.setColor(Color.BLUE);

                                g.drawString
                                ("age", 0, 10);

                                g.drawString
                                ("200", 0, height -100);

                                g.drawLine
                                (0, height -100, this
                                .getWidth(), height -100);

                                g.drawString("400"
                                , 0, height -200);

                                g.drawLine(0, height -200
                                , this.getWidth()
                                , height -200);

        }

        public Canvas getHistogram() {

                this.setMinimumSize
                (new Dimension(200, 200));
                return this;

        }

}
```

Listing A.6: Histogram.java

```
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

import java.util.Date;
import java.text.SimpleDateFormat;


public class OutputCreator {

        BufferedWriter writer;

        SimpleDateFormat currentDate =
        new SimpleDateFormat("yyyy-MM-dd_kk-mm-ss");

        String path = "C:\\Programme\\TussockSimulator";
        //the existence of this folder is assured because
        //of configuration.txt

        String fileOutput;


        public OutputCreator(){

          fileOutput =
          "Output_"+currentDate.format
          (new Date())+".csv";

        }


        public void saveOutput(String outputContent) {

                try {

                  //Save Output to file on harddisk:
                  writer = new BufferedWriter
                  (new FileWriter
                  (path+"\\"+fileOutput, true));
                  writer.write(outputContent);
                  writer.write("\n");

                }

                catch (IOException e) {

                 System.err.println
                 ("Error (FileOperator) writing
                 output file.");
```

```
                    }

                    finally {
                         if (writer != null)

                              try {writer.close();}

                              catch (IOException e) { }

                    }

                    }
}
```

Listing A.7: OutputCreator.java

```
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.File;

import java.util.Date;
import java.text.SimpleDateFormat;

import src.jxl.Workbook;
import src.jxl.WorkbookSettings;
import src.jxl.write.Number;
import src.jxl.write.NumberFormats;
import src.jxl.write.WritableCellFormat;
import src.jxl.write.WritableSheet;
import src.jxl.write.WritableWorkbook;
import src.jxl.write.WriteException;


public class OutputCreatorDetailed {

        SimpleDateFormat currentDate =
        new SimpleDateFormat("yyyy-MM-dd_kk-mm-ss");
        BufferedWriter writer;

        private String path;
        private String fileName;

        int timeStep;
        Cell[][] copyOfGrid;

        public OutputCreatorDetailed(String path) {

                this.path = path;

        }

        public void createDetailedOutput
        (Cell[][] copyOfGrid, int timeStep) {

                this.timeStep = timeStep;
                this.copyOfGrid = copyOfGrid;

                fileName = "Output_"
                +currentDate.format(new Date())
                +"_"+timeStep+"_detailed.xls";

                try {

                  WorkbookSettings excelSettings =
                  new WorkbookSettings();
```

```
WritableWorkbook workbook =
Workbook.createWorkbook
(new File(path+"\\"+fileName),
excelSettings);

WritableSheet sheetTemperature =
workbook.createSheet("Temperature", 1);
WritableSheet sheetHumidity =
workbook.createSheet("Humidity", 2);
WritableSheet sheetSpeciesNumber =
workbook.createSheet("SpeciesNo", 3);
WritableSheet sheetAge = workbook
.createSheet("Age", 4);
WritableSheet sheetLifeExpectancy =
workbook.createSheet
("LifeExpactancy", 5);
WritableSheet sheetMarturity =
workbook.createSheet("Marturiy", 6);
WritableSheet sheetOptTemperature =
workbook.createSheet
("OptTemperature", 7);
WritableSheet sheetOptHumidity =
workbook.createSheet("OptHumidity", 8);
WritableSheet sheetAnalysis =
workbook.createSheet("ANALYSIS", 9);

//WritableFont font =
new WritableFont(WritableFont.ARIAL,
10, WritableFont.BOLD);
WritableCellFormat format =
new WritableCellFormat
(NumberFormats.FLOAT);
format.setWrap(true);

for(int i = 0;
i < copyOfGrid.length ; i++) {
  for(int j = 0;
  j < copyOfGrid.length; j++) {

    sheetTemperature.addCell(new Number
    (i,j, copyOfGrid[i][j].temperature,
    format));
    sheetHumidity.addCell(new Number
    (i,j, copyOfGrid[i][j].humidity,
    format));

      if(copyOfGrid[i][j].isOccupied()
      == true) {

        sheetSpeciesNumber.addCell
```

```
                    (new Number(i,j,
                    copyOfGrid[i][j]
                    .getSpeciesNumber(), format));
                    sheetAge.addCell
                    (new Number(i,j,
                    copyOfGrid[i][j]
                    .getSpecies().getAge()
                    ,format));
                    sheetLifeExpectancy.addCell
                    (new Number(i,j,
                    copyOfGrid[i][j]
                    .getSpecies()
                    .getLifeExpectancy()
                    , format));
                    sheetMarturity.addCell
                    (new Number(i,j,
                    copyOfGrid[i][j]
                    .getSpecies()
                    .getMaturityExpectancy(),
                    format));
                    sheetOptTemperature.addCell
                    (new Number(i,j,
                    copyOfGrid[i][j]
                    .getSpecies()
                    .getOptimalTemperature()
                    , format));
                    sheetOptHumidity.addCell
                    (new Number(i,j,
                    copyOfGrid[i][j]
                    .getSpecies()
                    .getOptimalHumidity()
                    , format));

                }

            }

        }

        workbook.write();
        workbook.close();

    } catch(IOException e) {

        System.err.println("Error
        (OutputCreatorDetailed)
        writing excel file.");
        e.printStackTrace();

    } catch(WriteException e) {
```

```
                    System.err.println("Error
                    (OutputCreatorDetailed)
                    writing excel file.");
                    e.printStackTrace();

                }

        }

        /*
        public void writeSheet() throws WriteException {

                WritableFont font = new WritableFont
                (WritableFont.ARIAL, 10, WritableFont
                .NO_BOLD);
                WritableCellFormat format =
                new WritableCellFormat(NumberFormats
                .FLOAT);
                format.setWrap(true);

                for(int i = 0;
                i < copyOfGrid.length ; i++) {
                  for(int j = 0;
                  j < copyOfGrid.length; j++) {

                    sheetTemperature.addCell(new Label
                    (i,j, ""+copyOfGrid[i][j]
                    .temperature));


                  }

                }
                sheet.addCell(new Label())





    // Creates Label and divides value
    // of one cell of sheet by 2.5
    l = new Label(6,0, "Divide",cf);
    s.addCell(l);
    n = new Number(6,1, 12);
    s.addCell(n);
    f = new Formula(6,2, "F1/2.5");
    s.addCell(f);
  }
```

```
    public void createDetailedOutputSting
    (Cell[][] copyOfGrid) {


            int x = copyOfGrid.length;
            int y = copyOfGrid.length;

            detailedOutput = "";

            for(int i = 0; i < x; i++) {

                    for(int j = 0; j < y; j++) {

                            if(copyOfGrid[i][j]
                            .isOccupied() == true) {

                            detailedOutput += ""
                           +copyOfGrid[i][j]
                            .temperature+" "
                           +copyOfGrid[i][j]
                            .humidity
                           +" "+copyOfGrid[i][j]
                            .getSpeciesNumber()+"
                           "+copyOfGrid[i][j]
                            .getSpecies().getAge()
                           +",";

                            } else if(copyOfGrid[i][j]
                            .isOccupied() == false) {

                            detailedOutput += ",";

                            }

                            if(j == y-1) {// line end

                              detailedOutput += "\n";

                            }

                    }

            }

            //this.saveOutput();
    }
    /*
    public void saveOutput() {
```

```
                try {

                    //Save Output to file on harddisk:
                    writer = new BufferedWriter
                    (new FileWriter(path+"\\"
                    +fileName, true));
                    writer.write(detailedOutput);
                    writer.write("\n");

                }

                  catch (IOException e) {

                    System.err.println("Error
                    (FileOperator) writing detailed
                    output file.");

                          }

                finally {
                    if (writer != null)

                        try {writer.close();}

                        catch (IOException e) { }

                }
                }

        */
}
```

Listing A.8: OutputCreatorDetailed.java

```
import java.awt.Graphics;

import java.awt.Color;
import java.awt.Image;
import java.awt.image.BufferedImage;
import javax.swing.JComponent;

import java.io.IOException;
import java.io.File;
import javax.imageio.ImageIO;


public class PictureView extends JComponent {

        private static final long serialVersionUID
        = 8188891509337052769L;

        private String fileName;

        private Image scaledImage;
        private BufferedImage bufferedMap
        , scaledBufferedMap;

        private int newSize = 102;

        public PictureView(String fileName) {

                this.fileName = fileName;
                this.createImage();

        }

        public void createImage() {

                File pictureFile = new File(fileName);

                try {

                  bufferedMap = ImageIO.read(pictureFile);

              if (bufferedMap != null) {

                scaledImage = bufferedMap
                .getScaledInstance(newSize, newSize,
                BufferedImage.SCALE_SMOOTH);

                scaledBufferedMap =
                new BufferedImage(newSize, newSize
                , BufferedImage.TYPE_INT_RGB);
```

```
            scaledBufferedMap.getGraphics()
            .drawImage(scaledImage, 0, 0, null);

            //this.repaint();

          }
        }

        catch (IOException e) { e.printStackTrace(); }

    }

    public void paintComponent(Graphics g) {

              if (scaledImage != null) {

                g.drawImage(scaledImage, 0, 0, null);

              } else {

                g.drawString("Map ("+fileName+")
                is not available", 10, 10);

              }

    }


    public int readTemperatureScaledPicture
    (int x, int y) {

      int red = 0;

         if(x+1 <= scaledBufferedMap.getWidth()
        && y+1 <= scaledBufferedMap.getHeight()) {

            Color c = new Color(scaledBufferedMap
            .getRGB(x+1, y+1));
            red = c.getRed();

                    }

             return red;
    }

    public int readHumidityScaledPicture
    (int x, int y) {

        int blue = 0;
```

```
            if(x <= scaledBufferedMap.getWidth()
            && y <= scaledBufferedMap.getHeight()) {

                Color c = new Color(scaledBufferedMap
                .getRGB(x, y));
                blue = c.getBlue();

                        }

                 return blue;
        }

        public int getScaledImageHeight() {

                return scaledBufferedMap.getHeight();

        }

        public int getScaledImageWidth() {

                return scaledBufferedMap.getHeight();

        }


        public JComponent getPicture() {

                return this;

        }
}
```

Listing A.9: PictureView.java

```
import java.awt.Graphics;
import java.awt.Canvas;
import java.awt.Dimension;
import java.awt.Color;

public class Plot extends Canvas {

        private static final long serialVersionUID
        = -2500721455446380831L;

        int sizeX = 300;
        int sizeY = 340;

         int currentScaleY = sizeY;
         int currentDivider = 1;

        int plotLength = sizeX;

        int[][] history;
        //[SpeciesNumber][amountOfSpecies]

        int[] x = new int[plotLength];
        int internalCounter;

        Species[] allSpecies;
        Counter[] speciesCounter;
        Counter[] speciesCounterOrdered;


         public Plot(Species[] allSpec) {

                this.allSpecies = allSpec;
                this.internalCounter = 0;
                history =
                new int[allSpecies.length][plotLength];

         }

         public void setPlot(int stepCounter
        , Counter[] specCounter) {

                        this.speciesCounter = specCounter;

                        for(int i = 0;
                        i < specCounter.length; i++) {

                          history[i][internalCounter]
                          = specCounter[i]
                          .getAmountOfSpecies();
```

```
                    }

                    internalCounter++;
                    //x[internalCounter]
                    // = internalCounter;

                    if(internalCounter == plotLength-1
                    || stepCounter == 0) {

                    internalCounter = 0;

                    for(int i = 0;
                    i < plotLength; i++) {

                      for(int j = 0;
                      j < speciesCounter.length ;
                      j++) {

                       history[j][i] = 0;

                      }
                    }
                    }

             for(int i = 0;
             i < allSpecies.length ; i++) {

               history[i][internalCounter]
               = specCounter[i].getAmountOfSpecies();

             }

     }

     public void paint(Graphics g) {


             g.setColor(Color.BLUE);

             g.drawString(""+currentScaleY, 0, 10);
             g.drawLine(0, 10, sizeX, 10);

             g.drawString("0", 0, sizeY);

              for(int i = 0;
              i < speciesCounter.length ; i++) {

                g.setColor(allSpecies[i]
                .getSpeciesColor());
                for(int paintPosition = 0;
                paintPosition < plotLength;
                paintPosition++) {

                    g.drawLine(paintPosition,
                    (sizeY-history[i][paintPosition]
                    /currentDivider), paintPosition,
                    (sizeY-history[i][paintPosition]
                    /currentDivider));

                    }
            }
            g.setColor(Color.WHITE);
            g.drawLine(0, sizeY, sizeX, sizeY);

            this.getNewScaleY();
    }

    public Canvas getPlot() {

             this.setMinimumSize
             (new Dimension(sizeX, sizeY));
             return this;
    }

    public void reset() {

            this.internalCounter = 0;

            for(int i = 0;
            i < allSpecies.length; i++) {
                    for(int j = 0;
                    j < plotLength; j++) {

                            history[i][j] = 0;

                    }
            }
    }

    public void getNewScaleY() {

     int largest = speciesCounter[0].getSpecies()
     .getSpeciesNumber();

       for(int i = 0;
       i < speciesCounter.length; i++) {

     if(speciesCounter[i].getAmountOfSpecies() >
            speciesCounter[largest].getAmountOfSpecies()) {

                largest = speciesCounter[i].getSpecies()
                .getSpeciesNumber();

            }
        }

            if(currentScaleY-20 <=
            speciesCounter[largest]
            .getAmountOfSpecies()) {

                currentScaleY = currentScaleY * 2;

                currentDivider = currentDivider * 2;

            }

        }

}
```

Listing A.10: Plot.java

```
import javax.swing.JFrame;
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JButton;
import javax.swing.JScrollPane;
import javax.swing.JCheckBox;
import java.awt.GridLayout;
import java.awt.Dimension;

public class Setup implements ActionListener{

        private static final long serialVersionUID
        = -4745006927865007642L;

        JFrame setupFrame = new JFrame("Species Setup");
        static JButton buttonSave =
        new JButton("Save and Exit all");
        static JButton buttonCancel =
        new JButton("Cancel and back");

        int amountOfSpiecies = 8;
        // at best is an even number;

        SetupSpeciesControl[] listOfSpecies =
        new SetupSpeciesControl[amountOfSpiecies];

        FileOperator fileOperator = new FileOperator();

        SetupEnvironmentControl environment =
        new SetupEnvironmentControl();

        JCheckBox checkBoxTemperature =
        new JCheckBox("Use this map ("+environment
        .getPathTemperature()+") for temperature.");
        JCheckBox checkBoxHumidity =
        new JCheckBox("Use this map ("+environment
        .getPathTemperature()+") for humidity.");

        public void actionPerformed(ActionEvent ae) {
                String ac = ae.getActionCommand();

                if (ac.equals("Save and Exit all")) {

                  fileOperator.save(this
                  .allSpeciesStringCreator());
```

```
            fileOperator.saveEnvironment
            (checkBoxTemperature.isSelected()
            , checkBoxHumidity.isSelected());

                  System.exit(0);

      } else if (ac.equals("Cancel and back")) {

                  setupFrame.dispose();

      }
  }

  public String allSpeciesStringCreator() {

          String all = "";

          for(int i=0;i<listOfSpecies.length;i++) {

                  if(listOfSpecies[i]
                  .createString().length() > 0) {

                     all += listOfSpecies[i]
                     .createString() + " endline2 ";

                  }
          }
          return all;
  }

  public void readInSpeciesConfiguration() {

                  String all = fileOperator
                  .readSpecies();

                  if(all != "") {

                    String[] line = all
                    .split(" endline2 ");

                    for(int i=0;i<line.length;i++) {

                      String[] column =
                      line[i].split(" ; ");

                      listOfSpecies[i]
                      .textFieldSpecName
                      .setText(column[0]);
                      listOfSpecies[i]
                      .textFieldSpecStartRatio
```

```
                                    .setText(column[1]);
                                    listOfSpecies[i]
                                    .textFieldSpecPreferred
                                    HumidityMean.setText
                                    (column[2]);
                                    listOfSpecies[i]
                                    .textFieldSpecPreferred
                                    HumidityStdDev.setText
                                    (column[3]);
                                    listOfSpecies[i]
                                    .textFieldSpecPreferred
                                    TemperatureMean.setText
                                    (column[4]);
                                    listOfSpecies[i]
                                    .textFieldSpecPreferred
                                    TemperatureStdDev.setText
                                    (column[5]);
                                    listOfSpecies[i]
                                    .textFieldSpecMaturityMean
                                    .setText(column[6]);
                                    listOfSpecies[i]
                                    .textFieldSpecMaturityStdDev
                                    .setText(column[7]);
                                    listOfSpecies[i]
                                    .textFieldSpecDieMean
                                    .setText(column[8]);
                                    listOfSpecies[i]
                                    .textFieldSpecDieStdDev
                                    .setText(column[9]);
                                    listOfSpecies[i]
                                    .textFieldSpecCompetition
                                    Strength.setText(column[10]);
                                    listOfSpecies[i]
                                    .textFieldSpecColorRed
                                    .setText(column[11]);
                                    listOfSpecies[i]
                                    .textFieldSpecColorGreen
                                    .setText(column[12]);
                                    listOfSpecies[i]
                                    .textFieldSpecColorBlue
                                    .setText(column[13]);

                                }

                            }

        }

        public void readInEnvironmentConfiguration() {
```

```
        String all = fileOperator
        .readEnvironment();

        if(all != "") {

                String[] column =
                all.split(";");

                if(column[0].charAt(0) == 't') {

                  checkBoxTemperature
                  .setSelected(true);

                } else {

                  checkBoxTemperature
                  .setSelected(false);
                }

                if(column[1].charAt(0) == 't') {

                   checkBoxHumidity
                   .setSelected(true);

                } else {

                   checkBoxHumidity
                   .setSelected(false);
                }

        }
}

public void checkPicturesSameSize() {


        if(environment.getWidthTemperatureMap()
        != environment.getWidthHumidityMap()
        || environment.getHeightTemperatureMap()
        != environment.getHeightHumidityMap()) {

          JOptionPane.showMessageDialog(null,
          "Warning: The map for temperature has
          not the same size as the map for
          humidity! (compensated by scaling)",
          "Size Problem", JOptionPane.OK_OPTION);

        }

}
```

```
    public JFrame draw() {

            JPanel panelListOfAll = new JPanel();

            panelListOfAll.setLayout(new GridLayout
            ((amountOfSpiecies/2+1), 2, 10, 10));

            for(int i = 0;
            i < listOfSpecies.length; i++) {

                    SetupSpeciesControl a =
                    new SetupSpeciesControl(i);
                    listOfSpecies[i] = a;

                    panelListOfAll
                    .add(listOfSpecies[i]);

            }

            this.readInSpeciesConfiguration();
            this.readInEnvironmentConfiguration();

            JPanel panelTemperature = new JPanel();
             panelTemperature.setLayout
             (new GridLayout(3, 1));
             PictureView picViewTemp =
             new PictureView(environment
             .getPathTemperature());
             panelTemperature.add
             (new JLabel("Map for Temperature:"));
             panelTemperature.add(picViewTemp);
             panelTemperature
             .add(checkBoxTemperature);


            JPanel panelHumidity = new JPanel();
             panelHumidity.setLayout
             (new GridLayout(3, 1));
             PictureView picViewHum =
             new PictureView(environment
             .getPathHumidity());
             panelHumidity.add
             (new JLabel("Map for Humidity:"));
             panelHumidity.add(picViewHum);
             panelHumidity.add(checkBoxHumidity);

            panelListOfAll.add(panelTemperature);
            panelListOfAll.add(panelHumidity);
```

```
                JScrollPane scrollPane =
                new JScrollPane();
                scrollPane
                .setViewportView(panelListOfAll);


                JPanel panelButton = new JPanel();
                panelButton.add(buttonSave);
                panelButton.add(buttonCancel);


                setupFrame.setDefaultCloseOperation
                (JFrame.EXIT_ON_CLOSE);
                setupFrame.setMinimumSize
                (new Dimension (1060, 800));
                setupFrame.getContentPane()
                .add(scrollPane, BorderLayout.CENTER);
                setupFrame.getContentPane()
                .add(panelButton, BorderLayout.SOUTH);

                this.checkPicturesSameSize();

                setupFrame.setVisible(true);

                buttonSave.addActionListener(this);
                buttonCancel.addActionListener(this);

                return setupFrame;

        }
}
```

Listing A.11: Setup.java

```
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.File;
import javax.imageio.ImageIO;


public class SetupEnvironmentControl {

        private static final long serialVersionUID
        = -7938562760323006747L;

        private String path
        = "C:\\Programme\\TussockSimulator";
        private String temperaturePictureFileName
        = "temperatureMap.jpg";
        private String humidityPictureFileName
        = "humidityMap.jpg";

        private BufferedImage imageTemperature
        , imageHumidity;

        private int widthAbsolute = 100; //default value
        private int heightAbsolute = 100; //default value

        private int widthTemp, heightTemp
        , widthHum, heightHum;

        public SetupEnvironmentControl() {

                try {

                  imageTemperature = ImageIO.read
                  (new File(path+"\\"
                  +temperaturePictureFileName));
                  imageHumidity = ImageIO.read
                  (new File(path+"\\"
                  +humidityPictureFileName));

                  heightTemp =
                  imageTemperature.getHeight();
                  widthTemp = imageTemperature.getWidth();
                  heightHum = imageHumidity.getHeight();
                  widthHum = imageHumidity.getWidth();

                  if(imageTemperature.getHeight()
                  > imageHumidity.getHeight()) {

                          heightAbsolute =
                          imageTemperature.getHeight();
                } else {

                        heightAbsolute =
                        imageHumidity.getHeight();

                }

                if(imageTemperature.getWidth()
                > imageHumidity.getWidth()) {

                        widthAbsolute =
                        imageTemperature.getWidth();

                } else {

                        widthAbsolute =
                        imageHumidity.getWidth();

                }

              }
              catch (IOException e) {

                        System.err.println("Error
                        (SetupEnvironmentControl)
                        reading file(s).");

                        }
        }

        public int getHeightAbsolute() {

                return heightAbsolute;

        }

        public int getWidthAbsolute() {

                return widthAbsolute;

        }

        public int getWidthTemperatureMap() {

                return widthTemp;

        }

        public int getHeightTemperatureMap() {
                return heightTemp;

        }

        public int getWidthHumidityMap() {

                return widthHum;

        }

        public int getHeightHumidityMap() {

                return heightHum;

        }

        public String getPathTemperature() {

                return path+"\\"
                +temperaturePictureFileName;

        }

        public String getPathHumidity() {

                return path+"\\"+humidityPictureFileName;

        }

        public BufferedImage getScaledTemperatureMap() {

                return imageTemperature;

        }

        public BufferedImage getScaledHumidityMap() {

                return imageHumidity;

        }

}
```

Listing A.12: SetupEnvironmentControl.java

```
import javax.swing.JPanel;
import javax.swing.JLabel;
import javax.swing.JTextField;
import javax.swing.JCheckBox;

import java.awt.Dimension;

import java.awt.GridLayout;


public class SetupSpeciesControl extends JPanel {

        private static final long serialVersionUID = 1L;

                JLabel labelSpecActive =
                new JLabel("Species Active: ");

                JCheckBox checkBoxSpecActive =
                new JCheckBox();

                JLabel labelSpecNumber =
                new JLabel("Species Number: ");

                JTextField textFieldSpecNumber =
                new JTextField();

                JLabel labelSpecName =
                new JLabel("Name: ");

                JTextField textFieldSpecName =
                new JTextField();

                JLabel labelSpecStartRatio =
                new JLabel("Setup Ratio: ");

                JTextField textFieldSpecStartRatio =
                new JTextField();

                JLabel labelSpecPreferredHumidityMean =
                new JLabel("Humidity Mean: ");

                JTextField
                textFieldSpecPreferredHumidityMean =
                new JTextField();

                JLabel labelSpecPreferredHumidityStdDev =
                new JLabel
                ("Humidity Standard Deviation: ");
```

```
JTextField
textFieldSpecPreferredHumidityStdDev =
new JTextField();

JLabel labelSpecPreferredTemperatureMean =
new JLabel("Temperature Mean: ");

JTextField
textFieldSpecPreferredTemperatureMean =
new JTextField();

JLabel labelSpecPreferredTemperatureStdDev
= new JLabel
("Temperature Standard Deviation: ");

JTextField
textFieldSpecPreferredTemperatureStdDev =
new JTextField();

JLabel labelSpecMaturityMean =
new JLabel("Maturity Mean: ");

JTextField textFieldSpecMaturityMean =
new JTextField();

JLabel labelSpecMaturityStdDev =
new JLabel
("Maturity Standard Deviation: ");

JTextField textFieldSpecMaturityStdDev
= new JTextField();

JLabel labelSpecDieMean =
new JLabel("Life Expectancy (Mean): ");

JTextField textFieldSpecDieMean =
new JTextField();

JLabel labelSpecDieStdDev =
new JLabel("Life Expectancy
(Standard Deviation): ");

JTextField textFieldSpecDieStdDev =
new JTextField();

JLabel labelSpecCompetitionStrength =
new JLabel("Competition Strength: ");

JTextField
textFieldSpecCompetitionStrength =
new JTextField();

JLabel labelSpecColor =
new JLabel("Color (RGB): ");

JTextField textFieldSpecColorRed =
new JTextField();

JTextField textFieldSpecColorGreen =
new JTextField();

JTextField textFieldSpecColorBlue =
new JTextField();

JPanel panelColor = new JPanel();

public SetupSpeciesControl
(int speciesNumber) {

        this.setLayout
        (new GridLayout(14,2));
        this.setSize(200, 400);

        this.add(labelSpecActive);
        this.add(checkBoxSpecActive);

        this.add(labelSpecNumber);
        textFieldSpecNumber.setText
        (speciesNumber+"");
        textFieldSpecNumber.setEditable
        (false);
        this.add(textFieldSpecNumber);

        this.add(labelSpecName);
        this.add(textFieldSpecName);

        this.add(labelSpecStartRatio);
        this.add(textFieldSpecStartRatio);

        this.add
        (labelSpecPreferredHumidityMean);
        this.add
        (textFieldSpecPreferred
        HumidityMean);

        this.add(labelSpecPreferred
        HumidityStdDev);
        this.add(textFieldSpecPreferred
        HumidityStdDev);
```

```
                this.add(labelSpecPreferred
                TemperatureMean);
                this.add(textFieldSpecPreferred
                TemperatureMean);

                this.add(labelSpecPreferred
                TemperatureStdDev);
                this.add(textFieldSpecPreferred
                TemperatureStdDev);

                this.add(labelSpecMaturityMean);
                this.add
                (textFieldSpecMaturityMean);

                this.add(labelSpecMaturityStdDev);
                this.add
                (textFieldSpecMaturityStdDev);

                this.add(labelSpecDieMean);
                this.add(textFieldSpecDieMean);

                this.add(labelSpecDieStdDev);
                this.add(textFieldSpecDieStdDev);

                this.add
                (labelSpecCompetitionStrength);
                this.add
                (textFieldSpec
                CompetitionStrength);

                this.add(labelSpecColor);
                textFieldSpecColorRed
                .setPreferredSize
                (new Dimension(30, 20));
                textFieldSpecColorGreen
                .setPreferredSize(new Dimension
                (30, 20));
                textFieldSpecColorBlue
                .setPreferredSize(new Dimension
                (30, 20));
                panelColor.add
                (textFieldSpecColorRed);
                panelColor.add
                (textFieldSpecColorGreen);
                panelColor.add
                (textFieldSpecColorBlue);
                this.add(panelColor);

        }
```

```
public String createString() {

        String allInformationOfSpecies
        = "";

        if(textFieldSpecName.getText()
            .length() > 0 &&
            textFieldSpecStartRatio
            .getText().length() > 0 &&
            textFieldSpecPreferredHumidity
            Mean.getText().length() > 0 &&
            textFieldSpecPreferredHumidity
            StdDev.getText().length() > 0
            && textFieldSpecPreferred
            TemperatureMean.getText()
            .length() > 0 &&
            textFieldSpecPreferred
            TemperatureStdDev.getText()
            .length() > 0 &&
            textFieldSpecMaturityMean
            .getText().length() > 0 &&
            textFieldSpecMaturityStdDev
            .getText().length() > 0 &&
            textFieldSpecDieMean
            .getText().length() > 0 &&
            textFieldSpecDieStdDev
            .getText().length() > 0 &&
            textFieldSpecCompetition
            Strength.getText().length()
            > 0)

            {


             allInformationOfSpecies
             = textFieldSpecName
             .getText() + " ; "
             + textFieldSpecStartRatio
             .getText() + " ; "
             + textFieldSpecPreferred
             HumidityMean.getText()
             + " ; "
             + textFieldSpecPreferred
             HumidityStdDev.getText()
             + " ; "
             + textFieldSpecPreferred
             TemperatureMean.getText()
             + " ; "
             + textFieldSpecPreferred
             TemperatureStdDev.getText()
```

```
                                        + " ; "
                                        + textFieldSpecMaturityMean
                                        .getText() + " ; "
                                        + textFieldSpecMaturityStdDev
                                        .getText()+ " ; "
                                        + textFieldSpecDieMean
                                        .getText() + " ; "
                                        + textFieldSpecDieStdDev
                                        .getText() + " ; "
                                        + textFieldSpecCompetition
                                        Strength.getText() + " ; "
                                        + textFieldSpecColorRed
                                        .getText() + " ; "
                                        + textFieldSpecColorGreen
                                        .getText() + " ; "
                                        + textFieldSpecColorBlue
                                        .getText();
                                }

                                return allInformationOfSpecies;

                        }

}
```

Listing A.13: SetupSpeciesControl.java

```
import java.awt.Color;
import utilities.Util;

public class Species implements Cloneable {

        String name;
        int number;

        double startRatio;

        double optimalHumidityMean, optimalHumidityStdDev;
        double optimalTemperatureMean
        , optimalTemperatureStdDev;

        private int age;

        double maturityMean, maturityStdDev;
        double dieMean, dieStdDev;

        double competitionStrength;

        Color specColor;

        private double maturityExpectancy;
        private double lifeExpectancy;

        private double optimalHumidity
        , optimalTemperature;

    public Species clone() {

        try {
            return (Species) super.clone();
        } catch (CloneNotSupportedException cnse) {
            cnse.printStackTrace();

            return null;
        }
    }

    public void initializeSpecies() {

        this.calculateMaturityExpectancy();
        this.calculateLifeExpectancy();
        this.setOptimalTemperature();
        this.setOptimalHumidity();
        this.age = 0;
```

```
}

    public int getAge() {

            return age;

    }

    public int getSpeciesNumber() {

            return number;

    }

    public Color getSpeciesColor () {

            return specColor;

    }

    public String getSpeciesName() {

            return name;

    }

    public void ageing() {

              age++;
    }

    private void setOptimalTemperature() {

            double gaussValueForTemperature =
            Util.RAND.random_gaussian
            (optimalTemperatureMean,
            optimalTemperatureStdDev);
            optimalTemperature =
            gaussValueForTemperature;

    }

    //according to Liebigs law in two independence of
    //Temperature and Humidity etc.

    private void setOptimalHumidity() {

            double gaussValueForHumidity =
            Util.RAND.random_gaussian
            (optimalHumidityMean,
                optimalHumidityStdDev);
                optimalHumidity = gaussValueForHumidity;

        }

        private void calculateMaturityExpectancy() {

                maturityExpectancy =
                Util.RAND.random_gaussian(maturityMean,
                maturityStdDev);

        }

        private void calculateLifeExpectancy() {

                lifeExpectancy =
                Util.RAND.random_gaussian(dieMean,
                dieStdDev);

        }

        public double getOptimalTemperature() {

                return optimalTemperature;

        }

        public double getOptimalHumidity() {

                return optimalHumidity;

        }

        public boolean getMaturity() {

                if (this.age >= this.maturityExpectancy) {

                        return true;

                } else {

                        return false;

                }

        }

        public double getLifeExpectancy() {

                return this.lifeExpectancy;
        }

        public double getMaturityExpectancy() {

                return this.maturityExpectancy;

        }

        public boolean getDying() {


                if (this.age >= this.lifeExpectancy) {

                        return true;

                } else {

                        return false;

                }

        }

        public double getCompetitionStrength() {

                return competitionStrength;

        }

}
```

Listing A.14: Species.java

Appendix B

Certificate of originality

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written nor material which has been accepted for the award of any other degree, examination or assignment, except where due acknowledgments has been made in the text.

Koblenz; September 30, 2008

Michael Zaggl