An Agent-Based Trade Model

Could the absence of sailable rivers have influenced growth on the African continent?

Hans Christian Møller Ekne

Master Thesis for the degree of Master of Philosophy in Economics

Department of Economics

UNIVERSITY OF OSLO

May, 2012

An Agent-Based Trade Model

Could the absence of sailable rivers have influenced growth on the African continent?

Summary

This paper develops an agent-based trade model, and then uses that model to test the hypothesis of whether the absence of sailable rivers on the African continent could have influenced its growth.

Firstly, some growth theory is presented along with a basic introduction to agent-based modeling. The main part of the paper is devoted to introducing the model and then evaluating its results. The results indicate that under certain conditions, increased availability of trade routes is beneficial, but this is not always so.

Preface

Academically, the last couple of years have been fun and exciting, especially so this last year, where my focus largely has been on agent-based modeling. This way of doing economics seems to me a very natural approach to the field. After evaluating countless first and second order marginal conditions in various economics courses along the way to my masters degree, this methodlogy brings a breath of fresh air.

This type of research seems to be a process that I rather enjoy, both in terms of the personal freedom one has in choosing a research topic and concerning the work hours. It has been a rewarding journey and I have pushed the boundries both of my modeling technique and programming skills.

There are several people that have helped me along the way. Firstly, I would like to thank my supervisor Kjell Arne Brekke. He has always given me solid advice, and this paper, along with the model contained herein, would be much poorer were it not for his sharp analyzes. I would also like to thank Reidun, and my father Øivind for proof reading and other helpful tips.

Finally, I would like to thank my girlfriend Guro for her encouragement and motivation along the way. The process has been much easier thanks to you!

Hans Christian Ekne

Østerås, May 2012

Contents

Figures and Tables

1 Introduction

In this paper I attempt to use the methodology of agent-based modeling and the toolkit that this gives us to investigate a theory concerning the growth of Africa.

My interest in agent-based modeling was triggered by reading Eric Beinhocker's excellent book The Origin of Wealth in the spring of 2009. Beinhocker's book takes the reader on a historical tour of modern economics, goes on to introduce agent-based modeling, and finally his vision for how economics can evolve. While the book certainly has its flaws, it nonetheless served as a key inspiration for my work with agent-based models. The book, of course, takes its name from one of the great masterpieces of science, a book that is also completely accessible to the layman, namely, Darwin's Origin of the Species.

The idea of, and concepts involved in agent-based modeling, apparently needed to ferment in my mind for some time, because it would be another two years before I attempted my first agent-based model. This time I was influenced by Robert Axtell and Joshua Epstein's Growing Artificial Societies, Social Science from the Bottom Up. In this book the authors introduce the reader to their Sugarscape model, an artificial world where agents populate a virtual space and act according to given micro-rules. The collective action of the individual agents result in macro patterns that not always are similar to the micro-rules used to generate them. Using Matlab, I built a simple replication of their Sugarscape world and tested some of their results. This was a very rewarding task, leading to some programming insights I would otherwise be without.

My next model, and the one which laid the foundation for the model used in this paper, was developed in January of 2012. It was initially developed as part of a project for a course aptly titled 'Agent-based Modeling', held at the University of Oslo Jan. 4.- 6. The question I investigated was basically the same as is the theme of this paper, namely, how could the existence of sailable rivers on the African continent have influenced growth? That is, given that there was a means of cheap and easy trade between the east and west of Africa, how would this in turn affect growth?

In a 1998 article by Bloom and Sachs they introduce this idea as a part of their 'geographical' explanation for the poverty of Sub-Saharan Africa, but they don't develop the concept further. Developing a method of trade and then modeling that in a simplified computer simulation is what I aim to do with this thesis. The basic model consists of agents, the economic actors, and their behavior is governed by simple behavioral rules in a virtual environment. The agents populate a space that roughly resembles the African continent, on which there exists two distinct types of resources which the agents value. To survive each round of the simulations the agents need to be able to generate a given amount of utility by consuming the resources. If they are unable to complete this, they die and are replaced. This is repeated until the simulation settles down into a somewhat repeatable pattern.

In the baseline model, the east and west of the continent have different rates of growth for the two resources, that is, the east side has a higher growth rate of resource one while the west side has a higher growth rate of resource two. Here there is almost no trade between the east and the west. When I subsequently introduce the river to the model, which effectively enables the agents to trade over a large distance, the overall welfare of the agents is shown to increase. The possibility of trade, in line with traditional Ricardian models of trade, gives rise to mutual benefit amongst the agents.

Through several iterations and development stages, the model has undergone significant change and improvement since its conception in the winter of 2012. Most notably is the change in how the agents that populate the model consume their resources. Other important changes have been the extra utility gained from implementing new functions that monitor various statistics.

The structure of this paper is rather straightforward. Chapter two is devoted to a few different theories of growth and their supporting arguments. In chapter three I delve into the agent-based computational economics methodology and discuss the pros and cons of using it. Chapter four is devoted to the Sugarscape model, while chapter five details the exact specification of the model developed for this paper. In chapter six I present the main findings of the model along with some other minor model exploration. Chapters seven and eight are devoted to a discussion and conclusion, respectively.

2 Geography and Growth

As this paper develops a spatial model of trade I found it natural to focus on some of the theories concerning geography and growth. This chapter first explores several of the ways geography can influence growth, before I conclude with a short venture into some alternative hypothesis. The theories presented herein are largely taken from ideas presented by David Bloom and Jeffrey Sachs, from their article Geography, Demography and Economic Growth in Africa; Paul Collier’s ideas as presented in his book the Bottom Billion and an article by Daron Acemoglu, Simon Johnson and James Robinson titled Reversal of Fortune: Geography and Institutions in the Making of the Modern World Income Distribution. In their article, Sachs and Bloom point to geography and demography as key explanatory factors for poverty in Sub-Saharan Africa (SSA), and this view is partly shared by Collier who promotes the idea that the poorest countries among us, overly represented in SSA, are caught in four different development traps. These traps being: the conflict trap, the natural resources trap, the trap of being landlocked with bad neighbors, and the trap of bad governance in a small country. Lastly, in the article titled Reversal of Fortune the authors promote institutions, or the lack thereof, as a key source of poverty. It should also be noted that the article by Bloom and Sachs and the book by Collier are directed more at current sources of poverty while the article by Acemoglu et al. has a different time perspective, starting its analyzes in 1500. Despite these differences, a comparison can nonetheless be insightful.

2.1 Geography and Trade

The following two arguments, the absence of sailable rivers and being landlocked, form the basis for this thesis. In short, the proponents argue that countries that for some reason have natural barriers to trade, usually experience lower growth.

2.1.1 The absence of sailable rivers

The absence of sailable rivers argument is put forth by Bloom and Sachs and reads: *“Poor agricultural and health conditions would be barriers enough to African growth, as they have been in other parts of the tropics. Yet the situation is made much worse by the continent's remarkable disadvantages in transport costs. The factors underlying these disadvantages include …the absence of rivers leading into the interior of the continent that are navigable by* *ocean-going vessels, as are the Rhine, the Mississippi, the Amazon, and the Yangtze on other continents."* (Bloom and Sachs, 1998: 236). This thesis lays the foundation for the paper, and the model developed in chapter five aims to test this hypothesis. The absence of sailable rivers argument is built on the assumption that transportation costs and access to trading partners has an impact on growth. As rivers and waterways have been, and still are, a major source of trade, countries lacking these advantages should then be expected to have had lower growth. As we will see in the model results, implementing trade in the form of a river will have profound effects on the overall average age of the agents. The absence of sailable rivers argument ties closely into the next hypothesis, namely being landlocked, in that both arguments tie growth to trade.

2.1.2 Being landlocked (with bad neighbors)

Being landlocked simply means that a country does not have access to the sea, i.e. it is bordered by other countries on all sides. Both Sachs and Bloom and Collier use this as an explanation for poverty. Many of the poorest countries in SSA find themselves in this position, and empirically, if a country finds itself in this situation it is a good chance that it is among the poorest in the world. Now, this isn't to say there are not exceptions. Switzerland for example, is a country doing very well while being landlocked. This shows that being landlocked is usually only bad if you have bad neighbors, which unfortunately is the case for many of the SSA countries. Switzerland is bordered by Germany, France, Italy, Austria and Lichtenstein, among some of the richest countries in Europe, and all are viable trading partners. On the other hand, take a country like Central African Republic, bordered by Chad, Sudan, South Sudan, Democratic Republic of the Congo, Republic of the Congo and Cameroon. CAR placed 179th on the Human Development Index for 2011. Being landlocked with bad neighbors implies that there is probably a lack of regional trade, and also the neighboring countries in many cases lack adequate infrastructure for the efficient transportation of imports and exports. According to Collier, resource rich countries do not suffer from being landlocked. In their case, the resource makes up for not having harbors.

2.2 Alternative theories

Besides the arguments that tie growth directly to access to trade routes, there are several other arguments concerning growth that merit attention. I will briefly touch upon a few of these.

Arguments concerning disease, agriculture and resources are all also tied to geography, while the right type of institutions, or lack thereof, is put forth as another contender.

2.2.1 Disease

Bloom and Sachs point to disease as a contributing factor to slow growth as well. They put special focus on the vector borne infectious disease malaria. The authors estimate that malaria takes approximately one million lives every year and that most of these deaths occur in SSA (Bloom and Sachs, 1998:232). The fact that malaria doesn't pose a risk in more temperate climates links this problem directly to geography. It is hypothesized that existence of the disease also can influence foreign business men against entering the area in fear of being infected. This again has a negative impact on growth both because it inhibits trade and prevents new business from being established.

2.2.2 Agriculture

Bloom and Sachs also single out agriculture as an important source of poverty. They claim the tropical zone has problems with low yields due to an array of different factors. Amongst them are: too low photosynthetic potential, high variability in the rainfall and diseases that can affect cattle. There are some flies which can be especially harmful towards livestock. Countries in the SSA zone are almost all to some extent affected by these problems in agricultural production (Bloom Sachs, 1998: 227).

2.2.3 Resources

According to Collier, the resource trap is one of the reasons the poorest countries have developed poorly and continue to stay poor. He argues that countries with valuable resources can easily get into problems and the economy often experiences what is known as Dutch Disease. Dutch Disease refers to the situation of increasing the value of one's currency because of the amount of exports and thereby making it difficult for local exporters to be competitive on the global market. Over time this can be destructive to the domestic economy and there is often a lack of innovation needed for growth. There are also other problems that can persist in a resource rich country. One of the arguments for the resource trap is that with a sizable income stream being generated from resources, the government is not dependent on a tax base to finance its spending. Hence it will not contribute to the growth of such a base.

Also, according to Collier the problem of the resource trap is that it makes democracies malfunction. He claims that for several resource rich countries, the politicians have economic incentives that are counterproductive to the society at large. Corruption is apparently widespread, and hard to tackle.

2.2.4 Institutions

Not everyone agree with the geographical explanation previously examined, and a thesis linking institutions to growth is put forth by Acemoglu et al. in their Reversal of Fortune paper. This theory of growth does not necessarily rule out geographical factors, but the authors do claim that there is little support for these theories. Their hypothesis is that how the colonialist chose set up their institutions is a major explanation for why some countries that in 1500 were rich are now poor, and why some countries that were poor in 1500 are now rich. The countries that eventually profited from colonialism are the so called neo-European countries, that is USA, Canada, New Zealand and Australia, while the countries or civilizations that they argue are worse off because of colonialism are for example the Incas and the Aztecs and tribes in Africa and India. The authors measure this decline or rise in poverty by using urbanization or population density as a proxy for wealth, and then measuring those variables over time. The argument they present follows the following logic: In countries with low population densities the colonial powers often created institutions that resembled their own and they often settled in these locations themselves. They created institutions that sought to upheld private property rights, something they argue created a favorable investment climate that is necessary for growth. On the other hand, the countries that had higher population densities, and were considered to be wealthier, were often exploited by the colonialists, and institutions that supported the extraction of resources were often created. These institutions would then protect the elite so they could profitably extract resources, and institutions that supported private property rights were not adopted. In turn, this created a poor climate for investment and thus led to sluggish growth (Acemoglu et al., 2002: 1262).

One of the development traps introduced by Paul Collier in his Bottom Billion book is the trap of bad governance in a small country. This relates directly to the institutions argument, although unlinke Acemoglu et al., Collier makes no attempt to explain how it came to be, he simply states that it is a statistical fact that if a country is small and has bad government, then this increases the likelihood of a self-reinforcing development trap.

2.2.5 Conflict

In the Bottom Billion, Collier also notes that conflict in itself can be a development trap. Countries that have experienced conflict are at a higher risk of having some type of conflict recurring than countries that have been at peace. A conflict like for example civil war, will in most cases have a severely damaging effect on growth prospects.

3 Agent-based computational economics

This section is meant as a short introduction to the modeling technique used in this paper. First, I attempt to highlight some of the advantageous aspects of using the methodology and also show how agent-based computational economics (ACE) fits into a broader simulation category. I include the broader field of simulation, since many of the inherent properties of simulation research also, by extension, apply to ACE. The chapter is concluded with a short discussion of some of the drawbacks to using ACE.

Before exploring the ACE methodology a minor introduction is in order. As a definition, the following suits our needs: *"ACE is the computational study of economic processes modeled as dynamic systems of interacting agents who do not necessarily posses perfect rationality and information."* (Judd, Tesfatsion, 2006: xi). Examine this definition closely and notice keywords like computational, process, interacting agent, dynamic systems, imperfect information and imperfect rationality. An example of a simple ACE model could be a group of agents in a marketplace, each with their own trading strategy. One could let these agents execute trades and evaluate which one of them was able to profit the most from trading. A next step could be to isolate the agents with the profitable trading strategies and then replicate their behavior. Using results from simulations one could then infer properties of successful trading strategies.

3.1 Advantages of ACE

One of the most striking properties of ACE models is the ease at which they capture the spatial component of reality. Of course, when studying a problem that is geographical in nature, as the thesis of this paper, this property becomes a major advantage. In ACE, space becomes a natural part of the model without making it much harder to understand or analyze, in fact, in many cases the opposite is true. For example, in this paper the spatial component of the model makes it easier to understand and observe the effects of changing parameters. An agent's location in space does not just become another variable to be assigned a memory slot, but gives us insights we might otherwise have been without.

Another advantage of ACE is that it is a ‘bottom-up’ approach. One often starts a model with individual behavioral rules and the aggregate of these then becomes the system wide macro behavior. This is advantageous in for example policy design as one can easily adjust the local individual rules and observe how, if at all, the system changes its behavior. One can thus ‘test’ the effect of certain kinds of policy in a controlled environment.

W.B Arthur sheds light on two other important advantages of ACE. The following exert captures the essence of his view: “*it [ACE] enables us to examine how the economy behaves out of equilibrium, when it is not at a steady state…it is economics done in a more general way…The static equilibrium approach suffers two characteristic indeterminacies: it cannot easily resolve among multiple equilibria; nor can it easily model individuals’ choices of expectations…and when analyzed in formation – that is, out of equilibrium – these anomalies disappear*.” (Arthur, 2006:1552). Arthur’s arguments go along these lines: In the presence of positive feedbacks (technically under non-convexity), multiple equilibria are the norm rather than the exception. He claims traditional economics are only able to identify these equilibria but not tell us how they are chosen. Arthur advocates tackling the problem “generatively” and to use ACE to *“study the probability that a particular solution emerges under a certain set of initial conditions.”* (Arthur, 2006:1558). ACE then becomes a natural vantage point for determining how these equilibria form as we are able to study histories evolving. Concerning the issue of expectations he notes that the problem is self-referential. That is, my expectations of tomorrow may influence what I do today, but what I do today changes what happens tomorrow. Arthur explains: *“…agents attempt to forecast what the outcome will be; but their actions based on their forecasts determine this outcome. So the situation is self-referential…their choices of expectations depend on their choices of expectations…this is a fundamental indeterminacy in static economics.”* (Arthur, 2006:1559). He notes that the rational expectations model is a workaround to this problem but that it becomes theoretically singular, that is, a possibility, but very unlikely. Especially so, when the agents are heterogeneous (Arthur, 2006:1560).

In “Remarks on the Foundation of Agent-Based Generative Social Science” Epstein notes that agent-based modeling’s most important contribution is to *“facilitate the generative explanation”* (Epstein, 2006:1587) as a method of explanation in the social sciences. He puts forward that the generativist aims to answer the following question: *“How could the autonomous local interactions of heterogeneous boundedly rational agents generate the given* *regularity?"* (Epstein, 2006: 1587). A more traditional approach is to use game theory to explain a social outcome. For example, given a certain state, some outcome is a Nash equilibrium, and then use that as an explanation. The generativist on the other hand is not satisfied with the Nash condition but searches along a different path, asking the question, under what plausible initial conditions would a population of agents acting under some assumptions about behavior, arrive at the observed pattern. Then, those rules, initial conditions and behavioral rules, can explain the phenomenon. In Epstein's words: *"To explain a macroeconomic regularity…is to furnish a suitable microspecification that suffices to generate it."* (Epstein, 2006:1587).

3.2 ACE in a simulation context

Having reviewed some of the advantages of using ACE, a natural next step is to view ACE as a bigger part of a simulation context. ACE is a specific type of an agent-based methodology, and again, agent-based methodologies are a part of simulation framework. In Axelrod's "Advancing the Art of Simulation in the Social Sciences" he argues that simulation is a third way of doing science contrasted with induction and deduction.

With a pure deductive approach you generally start out with a few basic building blocks you define as true (axioms) and then build a framework around these. Mathematics builds its theorems in this way and it's also the standard approach in neo-classical economics. *"Conventional economic theory, following the style of mathematics in general and real analysis in particular, begins with a set of definitions and assumptions, and proves theorems."* (Judd, 2006:885). Also in Axelrod's words: *"Good examples include the neo-classical economic models in which rational agents operating under powerful assumptions about the availability of information and the capability to optimize can achieve an efficient reallocation among themselves through costless trading."* (Axelrod, 2005:5). And then there is induction, which he defines to be discovery of patterns in empirical data (Axelrod, 2005:5).

Simulation on the other hand *"is a third way of doing science. Like deduction, it starts with a set of explicit assumptions. But unlike deduction, it does not prove theorems. Instead, a simulation generates data than can be analyzed inductively. Unlike typical induction, however, the simulated data comes from a rigorously specified set of rules rather than direct measurement of the real world."* (Axelrod, 2005:5). Simulation can then give the researcher opportunity to investigate what type of explicit assumptions gives rise to varying emergent macro properties of a system. A researcher can then use a computer as a virtual laboratory and run experiments.

When programming a simulation model Axelrod identifies three goals the model should achieve: *"validity, usability, and extendibility."* (Axelrod, 2005:7). He argues that *"the goal of validity is for the program to correctly implement the model."* (Axelrod, 2005:7). This is of course critical, if the program implements something other than what the modeler had in mind the program can turn out to be misleading and wrong conclusions can be drawn. There is also usability, a property that is to allow the researcher and others to be able to use the program and understand it. Extendibility is also argued as a goal of the model. A good model should be able to be modified for other uses as well. In Java this can for example be done by adhering to principles of object oriented programming, where reuse and extendibility are central goals.

3.3 Current research objectives

Current ACE research can roughly be divided into four areas, differentiated by their objectives. These objectives are (Tesfatsion, 2006:838): Empirical understanding, normative understanding, quantitative insight and theory generation and methodological advancement. By empirical understanding we mean to answers questions of the type: Why has capitalism evolved the way it has? Why and how have the systems that are in place today evolved, and why are they continuing? Normative understanding is also an objective of ACE and the goal is to understand *"how can agent-based models be used as laboratories for the discovery of good economic design?"* (Tesfatsion, 2006:839). Researchers going down this road typically try to figure out where some policy will take us and what kind of systemic properties arise from different kinds of policies. Thirdly, another broad objective of ACE research is to obtain qualitative insight and theory generation. This approach aims to better understand the dynamics of economies under varying initial conditions. In this way we might be able to understand why some paths take us somewhere and why other paths have not been observed, i.e. why some alternative histories have not materialized. This is related to the study of contra-factual histories. It is well suited to give us an understanding of what is important for some outcomes to materialize, and in some cases this type of study can give perspectives that are useful for predicting future events. Lastly, we can classify a goal of ACE as that of methodological advancement. That is, how can we best provide researches with the tools needed to understand economies through the use of computational experiments? The current state of affairs within ACE methodology is mentioned by several authors in the “Handbook of Computational Economics, Vol. 2”.

This paper then, is an addition to the area of empirical understanding. We seek to understand why Africa has grown slowly and use an ACE model to test that hypothesis.

3.4 Drawbacks of ACE

Although this paper so far has shed light on mostly positive aspects of agent-based modeling, it should be noted that there are drawbacks to using this technique vs more traditional approaches. Concerning the disadvantages of agent-based modeling relative to more traditional approaches, there are several points that merit further examination. Firstly, when constructing an ACE model, that model should be dynamically complete. By dynamically complete is meant that given a specified set of initial conditions, the model should be able to evolve on its own without further input or intervention from the modeler. This is a disadvantage, as the completeness requires the modeler to specify methods and detailed specifications of initial conditions. Small changes in initial conditions can change the outcome of a model to a significant degree. Thus for solid and robust predictions to be achieved testing over wide range of these initial conditions is needed. This is an important point, and it should also be noted, one that has not been fully pursued in this paper. Due to time limitations and perhaps poor modeling technique, not all parameters that could affect the outcome have been rigorously tested. Tesfatsion also notes that it is not clear how ACE models will scale when we dramatically increase the number of agents and initial specifications.

Secondly, another drawback of ACE is the difficulty involved in empirically validating the models. ACE models typically predict outcome distributions for theoretical economic outcomes, but in the real world these outcomes (if at all predicted) are always realized as one specific event and these events might be in a thin tail of the predicted distributions. (Tesfatsion, 2006:845). This problem means it can be difficult to validate a model, at least if the scenario is not repeating. Also, computational programs can pose a bigger challenge when it comes to checking their correctness, compared to, for example, a mathematical proof. More on this in the discussion.

4 Overview of the Sugarscape model

The key inspiration for building a model such as the one I have made in this paper, stems from ideas picked up in “Growing Artificial Societies: Social Science from the bottom Up” by Joshua M. Epstein and Robert Axtell. In this fascinating book, the authors create a rich artificial world populated by virtual agents. By using simple local rules governing the evolution of the virtual landscape and the agents’ behavior the scientists are for example able to generate a distinct wealth distribution and intricate trading patterns. This chapter firstly serves as an example of an agent-based model, and I will attempt to give a short overview of the aspects of the book that have relevance for this paper. I will also give in-dept analyzes of some of the algorithms that are used, as a few of these algorithms also become an integral part of my model.

4.1 Model basics

The basic model the authors introduce the reader to, is a model that consists of an environment and agents acting in that environment. First they describe an environment in which the agents can interact. In their Sugarscape model this environment consists of a 50 by 50 lattice of individual sites. These sites can have different properties such as sugar levels and rates of growth of sugar. The agents are the ‘people’ of the model, and they populate this Sugarscape. The agents in turn have behavioral rules and attributes. The behavioral rules of the agents can for example be how they move, and an agent attribute can for example be the agents’ vision or age. In addition to these to objects, the sites and the agents, the Sugarscape is also classified by rules. These rules can either be environment-environment rules, that is, for example a rule that tells one site to grow sugar fast if a site next to it also has sugar. Rules can also be agent-environment, and an example of such a rule is for example how the agents move around in the environment, what the authors refer to as a movement rule. There can also be agent-agent rules, and these are rules that govern how the agents interact with each other. An example of this is for example a trading rule. Rules will be defined more precisely in the subsequent sections.

The authors also describe the system heuristically. They invite the reader to think of the artificial society as a discrete time dynamical system in which a vector $\boldsymbol{\alpha}$ describes all the agents' internal states and the vector **β** describes all environmental states and that these two vectors interact as a high-dimensional discrete time dynamical system on the general form:

$$\boldsymbol{\alpha}^{t+1} = \boldsymbol{f}(\boldsymbol{\alpha}^{t}, \boldsymbol{\beta}^{t})$$

$$\boldsymbol{\beta}^{t+1} = \boldsymbol{g}(\boldsymbol{\alpha}^{t}, \boldsymbol{\beta}^{t})$$

where the vector functions **f(*)** and **g(*)** map the space of all sites at time t to the space at time t+1. In this way it is possible to describe the system rather compactly.

4.2 Life and death on Sugarscape

Before the authors start the simulations they give precise definitions of the rules that govern system behavior. They begin by establishing all rules that govern the environment as part of the set **E**, and all rules that govern agent behavior as part of the set **A**. The ordered pair (**E**,**A**) now describe the complete set of rules for the system. For example, the growback rule $\mathbf{G}_{\alpha}$, which was mentioned previously, belongs to the set **E**, and the subscript indicates the rate at which sugar resources grow back on a site. The authors define the growback rule in the following way:

Sugarscape growback rule $\mathbf{G}_{\alpha}$: At each lattice position, sugar grows back at a rate of α units per time interval up to the capacity at that position (Epstein and Axtell, 1996:23).

When the subscript is ∞ this indicates resources fully replenish up to the capacity of that site.

The movement rule **M** belongs to the set **A** and describes how the agents move. It works in the following way: Agents can move to the sites that are in their 'extended' von Neumann neighborhood. That is, the sites directly to the north, south, east and west of the agent. The distance the agents can move is regulated by their vision attribute. For example, an agent with a vision attribute of two has a choice of eight possible sites he can move to. The two sites directly north, the two directly west and so on. In this way we give the agent something resembling a bounded rationality, as he might have a golden egg laying on a site northwest of his position but is unable 'see' it. So, this means that when it's the agents' turn to move it will evaluate where there is the most sugar and move there. The agent will then take all the sugar in that site and add it to his existing sugar value. Stating the movement rule more compactly:

Agent movement rule **M**:

- Look as far as vision permits in the four principal lattice directions and identify the unoccupied site(s) having the most sugar;
- If the greatest sugar value appears on multiple sites then select the nearest one;
- Move to this site;
- Collect all the sugar at this new position. (Epstein and Axtell, 1996:25)

The replacement rule $\mathbf{R}_{[a,b]}$ is introduced and it also belongs to the set **A**. Here the interval [a,b] is meant to indicate the interval which the maximum ages of the agents are uniformly drawn from. Formally stated:

Agent replacement rule $\mathbf{R}_{[a,b]}$: When an agent dies he is replaced by and agent of age 0 having random genetic attributes, random position on the sugarscape, random initial endowment, and maximum age randomly selected from the range [a,b] (Epstein and Axtell, 1996:33).

To illustrate the notation, if a simulation is described as $(\{\mathbf{G}_2\},\{\mathbf{M},\mathbf{R}_{[20,30]}\})$ this means that resources grow back at according to the growback rule $\mathbf{G}_2$, that is they replenish 2 sugar for each turn until capacity is reached. Also, agents move according to movement rule **M** and the agents' maximum age is between 20 and 30 turns. This classification is rather useful and will be used extensively in describing my model simulations as well.

With the rules in place, the authors start describing their simulations. In the most basic simulations one sugar is placed on every site but in the upper right quadrant and in the lower left quadrant of the lattice they place extra amounts of sugar so that 'sugar tops' are formed there. These levels are then the sugar capacities of the different sites, and the sites have growback equal to their capacity. They then distribute the agents randomly on the sites, and the agents then start moving according to their movement rule M.

After the agent has moved he will consume a given amount of sugar according to his metabolism. This metabolism can for example be a number between 1 and 4. The program then evaluates whether he has reached his maximum age or whether he didn't have enough resources to satisfy his metabolism, in which case he will die and is replaced. When the agent is replaced, another agent is spawned at a random available site in the lattice. An important aspect to note is also that the 50 by 50 lattice of sites has the shape of a torus. This implies that if an agent finds himself on the left most point of the lattice and is considering the movement rule, the agent will consider sites that are on the far right side of the lattice as well, and move there if that maximizes his sugar wealth.

After all the agents have moved and consumed, one turn of the system has elapsed. A new turn is then initiated and the system calls the growback rule. Sugar is replaced according to this rule and the agents move again. This continues as long as the simulation lasts.

With this simple system now in place, the authors report several interesting findings. One of which is how sugar wealth is distributed amongst the agents. Under the Rules ({G1},{M1,R[60,100]}) the authors find that wealth distribution becomes severely skewed in the population. A few of the agents have a lot of sugar wealth, while most agents are very poor. They classify this skewed wealth distribution as an emergent structure which is statistical in nature. The term ‘emergent structure’, is here meant to be a stable macroscopic pattern arising from the local interaction of agents (Epstein and Axtell, 1996:35).

The authors now go on to develop several different kinds of rules both for the environment and the agents, but these do not have any direct connection to the model used in this paper, so they are omitted. Instead, I move to the chapter which covers trading amongst the agents, the central feature of my model.

4.3 Sugar and Spice in Sugarscape

Epstein and Axtell now enrich the model with another resource, spice. This is done to facilitate trade amongst the agents. As well as using the other rules previously mentioned, the authors now also introduce a new trading rule and a few new concepts. These concepts include a welfare function and the marginal rate of substitution of spice for sugar, MRS.

Firstly, the agents need a way to evaluate how they value sugar and spice. Earlier, in the simple one resource model, this was easy, as the agents could just choose to maximize their sugar wealth. Now, with two resources, the problem because a little more challenging. The authors solve this by introducing the following welfare function[1]:

[1] The authors refer to this as a welfare function and this serves the same purpose as my utility function.

$$W(w_1, w_2) := w_1^{m_1/m_T} * w_2^{m_2/m_T}$$

Sugar is given the subscript 1 while spice is given 2. W is here given to be the overall welfare of the agent and w_1 and w_2 is taken to be the wealth levels of spice and sugar, respectively. M1 and m2 is taken to be the agents' metabolism of sugar and spice respectively and m_T is simply their sum, that is, $m_T = m_1+m_2$. The agents now a have a consistent means of evaluating alternative sugar/spice combinations, and then choosing the alternative which maximizes this welfare function. The authors now introduce the multicommodity movement rule:

<u>Multicommodity agent movement rule **M**</u> :

- Look out as far as vision permits in each of the four lattice directions, north, south, east and west;
- Considering only unoccupied lattice positions, find the nearest position producing maximum welfare;
- Move to the new position;
- Collect all the resources at that location (Epstein and Axtell, 1996:99).

Having introduced the new movement rule the authors go on to develop the concept of MRS. This is important because it governs how the trading rule functions. The MRS is a classic tool used in economics to determine relative valuations of resources and a short digression is in order. From Cowell's Microeconomics we have the following definition: *" The marginal rate of substitution of good i for good j (written MRS_{ij}) is $U_j \ (\boldsymbol{X})/U_i \ (\boldsymbol{X})$."* (Cowell, 2006:79). Here, U is a utility function, serving the same purpose as Epstein and Axtell's welfare function. $U_i \ (\boldsymbol{X})$ is taken to be $\partial U/\partial X_i$, i.e. the partial derivative of that utility function with respect to the good i, and **X** is a vector of the goods. He further comments : *"The marginal rate of substitution has an attractive intuition: MRS_{ij} is the person's marginal willingness to pay for commodity j, measured in terms of commodity i-"* (Cowell, 2006:79).

Now, back to Sugarscape, consider a situation where agent one and agent two are evaluating whether to trade. Letting resource one be sugar, resource two be spice, and the postscript denotes the resources, MRS_{21} becomes the agent's marginal rate of substitution of spice for sugar. Letting W_1 and W_2 equal the partial derivative of the welfare function with respect to sugar and spice respectively, some algebra yields:

$$MRS_{21} = \frac{W_1\ (r)}{W_2\ (r)} = \frac{\left(\partial W \big/ \partial w_1\right)}{\left(\partial U \big/ \partial w_2\right)} = \frac{\left(\frac{m_1}{m_T} w_1^{(m_1 - m_T)/m_T} * w_2^{m_2/m_T}\right)}{\left(\frac{m_2}{m_T} w_1^{m_1/m_T} * w_2^{(m_2 - m_T)/m_T}\right)} = \frac{m_1 w_2}{m_2 w_1}$$

Letting the superscript indicate the number of the agent, going back to the intuition behind the numbers, we have that since MRS_{21}^{2} denotes agent two's willingness to pay for sugar in units of spice and MRS_{21}^{1} is agent one's willingness to pay for sugar in units of spice, if MRS_{21}^{2} is lower than MRS_{21}^{1}, agent one has a higher willingness to pay for sugar than agent two which implies that in a potential trade the flow of goods must be spice from agent one to agent two and sugar from agent two to agent one. Of course if the situation is reversed the flow of trade will go in the opposite direction. Concerning the exchange price of the resources we use *"the geometric mean of the two endpoints of the feasible price range."* (Axtell, Epstein, 1996:104). According to the authors the result of this rule for computing the price is to lessen the effect of two agents having big differences in their MRSs. So the price then becomes:

$$p(MRS_{21}^{1}, MRS_{21}^{2}) = \sqrt[2]{MRS_{21}^{1}\ \times\ MRS_{21}^{2}}.$$

Having figured out the direction of trade and the potential price, Epstein and Axtell now formulate their trading rule:

<u>Agent trade rule **T**</u>:

- Agent and neighbor compute their MRSs; if these are equal then end, else continue;
- The direction of exchange is as follows: spice flows from the agent with the higher MRS to the agent with the lower MRS while sugar goes in the opposite direction;
- The geometric mean of the two MRSs is calculated- this will serve as the price,p;
- The quantities to be exchanged are as follows: if p > 1 then p units of spice for 1 unit of sugar; if p < 1 then 1/p units of sugar for 1 unit of spice;

- If this trade will (a) make both agents better off (increase the welfare of both agents), and (b) not cause the agents' MRSs to cross over one another, then the trade is made and return to start, else end (Epstein and Axtell, 1996:105).

With all this framework finally in place, the authors go on to describe the results of their simulations. The first simulation they report from is a classical neoclassic setting with infinitely lived agents with behavioral rules **M** and **T**. The agents' vision is uniformly drawn from the integer interval [1,5] and so is their metabolisms. Initial endowments for sugar and spice are randomly distributed between 25 and 50 for both sugar and spice. They then report that this configuration under rule system $(\{\mathbf{G}_1\},\{\mathbf{M},\mathbf{T}\})$ attains something resembling a market equilibrium price after the system settles down. This is then an economic equilibrium that has emerged from the bottom up (Epstein and Axtell, 1996:109).

The authors report simulations for several different rule system configurations, among them is the systems rule $(\{\mathbf{G}_1\},\{\mathbf{M},\mathbf{R}_{[60,100]},\mathbf{T}\})$. This rule is almost the same as before, except now, instead of having infinitely lived agents, they are using replacement rule $\mathbf{R}_{[60,100]}$. This gives rise to much larger variation in average trade price than in the earlier example. In the words of the authors: *"...this straightforward departure from the neoclassical agent produces market performance at considerable variance with the Walrasian general equilibrium."* (Epstein and Axtell, 1996: 120). Instead of convergence towards an equilibrium, the price never settles down.

Besides these examples the authors test the implications of variations in the welfare function, introducing credit rules, a sexual reproduction rule and a cultural transmission rule. These all contribute to variations in the previous results. Introducing credit rules for example, leads to intricate credit networks, where some of the agents only act as middlemen, and several layers of borrowers and lenders emerge.

This concludes the overview of the Sugarscape model. Epstein and Axtell's virtual world serves as a great example of how to start thinking about agent-based models. The next chapter will focus on my implementation, and how the rules and algorithms have been adapted to suit slightly different modeling needs.

5 Afriscape[2]

This chapter deals with the specifics of the model that has been built for this project. In tribute to the Sugarscape model, and because this model resembles the African continent, it will be referred to as Afriscape. Many of the previously developed concepts will be applicable here as well. On a few issues the model diverges slightly, and these will need to be studied in more detail.

5.1 Model Setup

The model is constructed similarly to the way Sugarscape is setup. Instead of a 50 by 50 lattice, I here start with a lattice of 70 by 70 unique sites. The collection of these sites will be referred to as the grid. This grid is then modeled to resemble that of the African continent. An illustration is given in figure 5-1. It is a very rough approximation, and all the results could probably have been obtained by creating a simpler structure, but it removes some of the abstraction from the model, making it at least resemble a concrete geographical location.

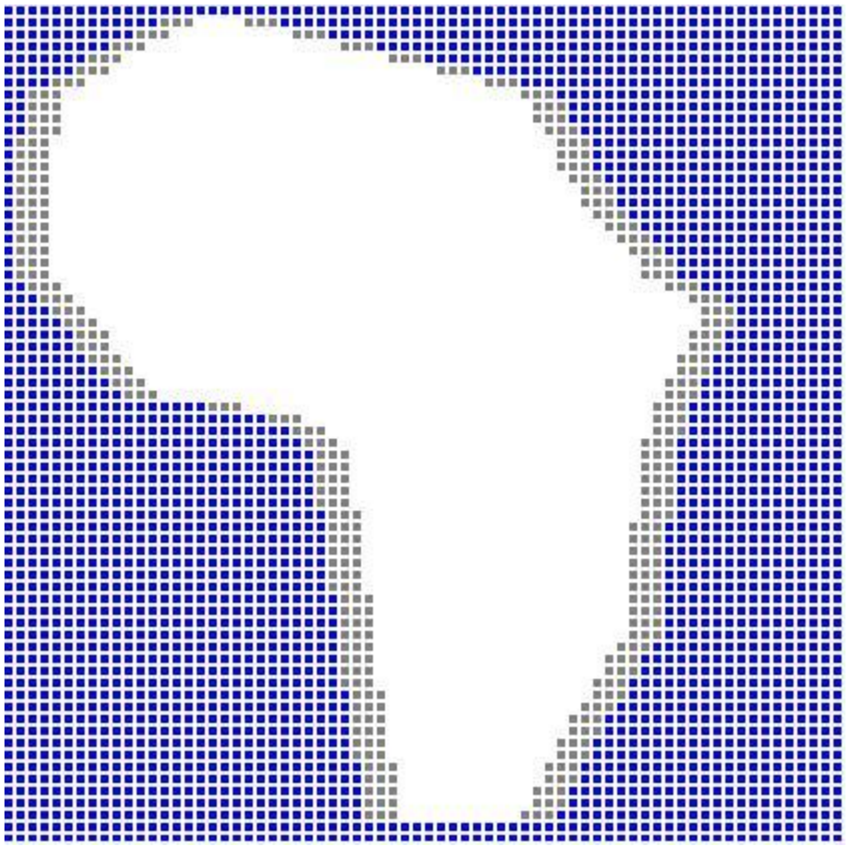
Figure 5-1

All sites in the grid belonging to the African continent are labeled land, and all other sites in the grid are labeled water. Thus, we have a continent surrounded by water. The left and right hand side of the model will be referred to as

[2] Although maybe not the most elegant name, it does make it easier to distinguish between my model and Sugarscape.

the west and east side respectively. After the continent is created, some of the land sites are endowed with resource capacity levels.

This model is a two resource model and all sites can be endowed with both of the resources. They will be referred to as resource one and resource two. The resources are initially distributed in the following matter: All sites that are at most three sites removed from the water in a horizontal direction on the west side, are endowed with a capacity of 24 units of resource one and two units of resource two. Conversely, on the east side, the sites that are at most three sites removed from the water in a horizontal direction are endowed with a capacity of 24 units of resource two and two units of resource one. In figure 5-1, all sites with high values of either resource one or two are colored grey. Each grid site is also endowed with a movement cost. In most experiments this is set to 1.

The next step is then to create the agents that populate the space. They are created with the following attributes: vision, maximum age and resource endowments of each of the resources. Along with these attributes the agents are also given individual weights as to how they evaluate their utility functions. In this respect, the model diverges from the Sugarscape model. Instead of creating a Cobb-Douglas utility function, in Africscape the agents base their evaluations on a logarithmic utility function on the form:

$$U = a * \ln(Resource\ 1) + \ b * \ln(Resource\ 2)$$

The a and b are taken to be the agent’s individual weighting of each of the resources. For example, an agent with a > b will value resource one higher than resource two. For practical purposes a+b =1, and in most simulations a=b=0.5, although this restriction is relaxed in a few experiments. This changing of the utility function of course also means that the MRSs of the agents change. With this setup, MRS_{12} now becomes:

$$MRS_{12} = \frac{U_2\ (r)}{U_1\ (r)} = \frac{\left({\partial U}/{\partial R_2} \right)}{\left({\partial U}/{\partial R_1} \right)} = \frac{\left(b \ /{R_2} \right)}{\left(a \ /{R_1} \right)} = \left(\frac{R_1}{R_2}\right) \times \left(\frac{b}{a}\right)$$

U_2 is then taken to be the partial derivative of the utility function with respect to resource two, and R_1 and R_2 is taken to be resource one and two respectively. With a=b the agents MRS_{12} is then just the fraction of the amount of resource one over resource two.

5.2 System Rules

Most of the rules developed in the last chapter have to be slightly modified to fit into the Afriscape setting. I will here first cover the system rules, and then the agent rules.

The first system rule introduced is the growback rule, it is almost identical to the one used Sugarscape, but with a few slight modifications. It is given as:

Afriscape growback rule $\mathbf{AG}_{[wr1,wr2,er1,er2)]}$: On the west side, resource one grows back at a rate wr1 units per time interval up to the capacity of that position, and unit two grows back a rate wr2 units per time interval up to the capacity of that position. On the east side, resource one grows back at a rate er1 units per time interval up to the capacity of that position, and unit two grows back a rate er2 units per time interval up to the capacity of that position. In the baseline case, wr1 and er2 are both set to 12, while wr2 and er2 are set to one.

So this rule enables the east and west side to have different growback rates for each of the two resources, and as we shall see in the next chapter, these differences will strongly affect the average population age when we open up for trade.

The next rule that we introduce is unique to Afriscape and is an environment rule, named the river rule. It's defined as:

Afriscape river rule $\mathbf{ARR}_{\alpha}$: All lattice sites with y-coordinate[3] equal to α that were formerly labeled land are now labeled water.

This rule creates the river that runs across Afriscape, and this river will be of importance because it greatly facilitates trade between the east and west side.

[3] The grid is what we refer to as a two dimensional array in java. These arrays start at 0 and then move down the integer line. Also one starts with the upper left corner and counts downwards and rightwards, i.e. site[y][x] then refers to a site that is y removed from the upper left corner down, and x sites to the right of the upper left corner.(This system also switches the x and y coordinates compared to the usual carthesian system).

Concerning the rules governing the agents, when the algorithm is running the first rule that the agent is confronted with is the trading rule. The Afriscape trading rule is defined as:

Afriscape trading rule **AT**:

- The agent constructs a five by five matrix of the sites around him, with himself at the center.
- If there are no other agents here, skip this step, else randomly initiate trade with the other agent(s) in this matrix.
- The direction of exchange is as follows: resource one flows from the agent with the higher MRS to the agent with the lower MRS while resource two goes in the opposite direction.
- The geometric mean of the two MRSs is calculated- this will serve as the price, p.
- The quantities to be exchanged are as follows: if $p > 1$ then p units of resource one for one unit of resource two; if $p<1$ then $1/p$ units of resource two for one unit of resource one.
- If this trade will (a) make both agents better off (increase the utility of both agents), and (b) not cause the agents' MRSs to cross over one another, then the trade is made.
- Provided the agent is close to the river, construct an array of all sites that have y-coordinates that are a most two sites above and tw sites below the y-coordinate of the river.
- Follow steps 2-6 of this rule.
- If there were no trades made then end, else repeat the whole process from step 1.

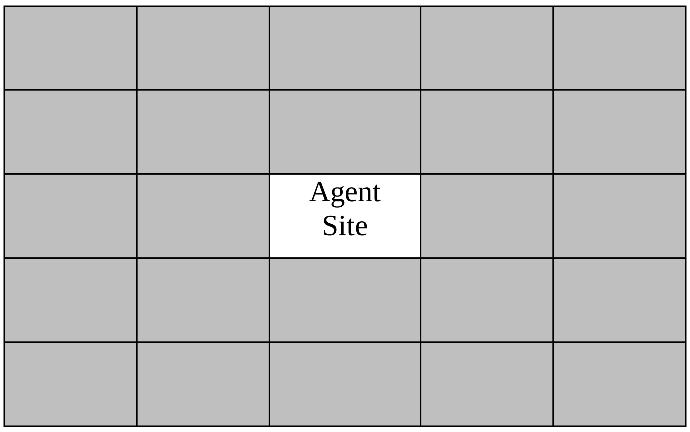

Figure 5-2

Figure5-2 illustrates how the agent's position is related to all the sites that are considered for trade. The river sites come in addition to these.

As we see, this rule is dependent on the river rule, **ARR**$_\alpha$, that was introduced earlier. The rule is rather long and tedious, but the point is that the agent should be able to realize all possible gains from trading. When the river rule is turned on, any agent occupying a site close[4] to the river can trade with any of the other agents close to the river. In effect, the river enables agents that might not otherwise be able to trade, to do so, and in just the same fashion as if the potential trading partners were in each other's immediate surroundings.

The movement rule, observing that it is almost identical to the Sugarscape multicommodity movement rule, is defined as:

<u>Afriscape multicommodity movement rule **AM**</u>:

- Look out as far as vision permits in each of the four lattice directions, north, south, east and west;
- Considering only unoccupied lattice positions, find the position that yields maximum utility after movement costs have been deducted, if several spots will yield the same utility, randomize over these;
- Move to the new position;
- Collect all the resources at that location.

[4] If an agent is one or two sites removed from the river, either north or south, he can trade with any of the other agents that are also in such a position.

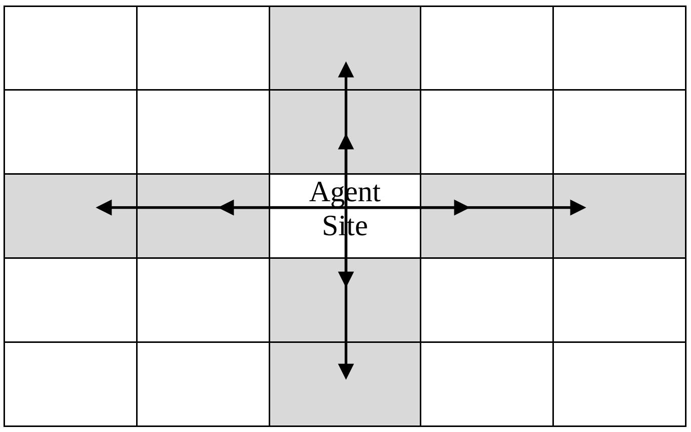

Figure 5-3

Figure 5-3 illustrates the possible sites an agent needs to evaluate when he is considering whether to move. In the example above it is assumed the agent has a vision parameter of two.

So, as is apparent, this rule is very similar to the Sugarscape rule except in part two. In the Sugarscape algorithm, it moves the agent to the closest site, while in the Afriscape version the agent randomizes over sites that give equal utility. Movement costs are also factored in here, something that is not included in Sugarscape.

After the agent has attempted to make a profitable move and also attempted to make profitable trades, he consumes resources. An easy way to compute this consumption could be done by having the agent consume a constant given portion of each of the resources he owns. The problem with this way of consuming is that the agent's utility is not linear in the inputs and theoretically the agent could then make a move that would give him a higher utility in one round and then kill him in the next. This problem was solved by having the agents always consume a given percentage of their resources so that they always achieve a utility of two. That is, every time the agent consumes resources he solves the following equation for m:

$$2 = a * \ln(m * Resource\ 1) + \ b * \ln(m * Resource\ 2)$$

which implies:

$$m = \frac{e^2}{(Resource\ 1)^a \ * (Resource\ 2)^b}$$

This way of consuming resources ensures that the agent always consumes in a fashion that maximizes his remaining life given that he does not gain any further resources. Also, this enables the average agent population age to be used as a proxy for measuring system welfare.

A more realistic approach might be to have some goods consumed proportionally, and other in given amounts. We can imagine a situation in real life where for example water might be a resource that is consumed in fixed amounts each time period, while champagne might be more of a luxury beverage, something that is consumed proportionally, according to how much wealth the person has.

5.3 Model initialization

After the world has been created and the agents initialized and placed on the map, the following is a summary of a turn in the simulation:

1. The move order of agents is randomized and the first agent to move decides if he wants to trade with other agents and then moves to another site. This is then repeated for all the agents.

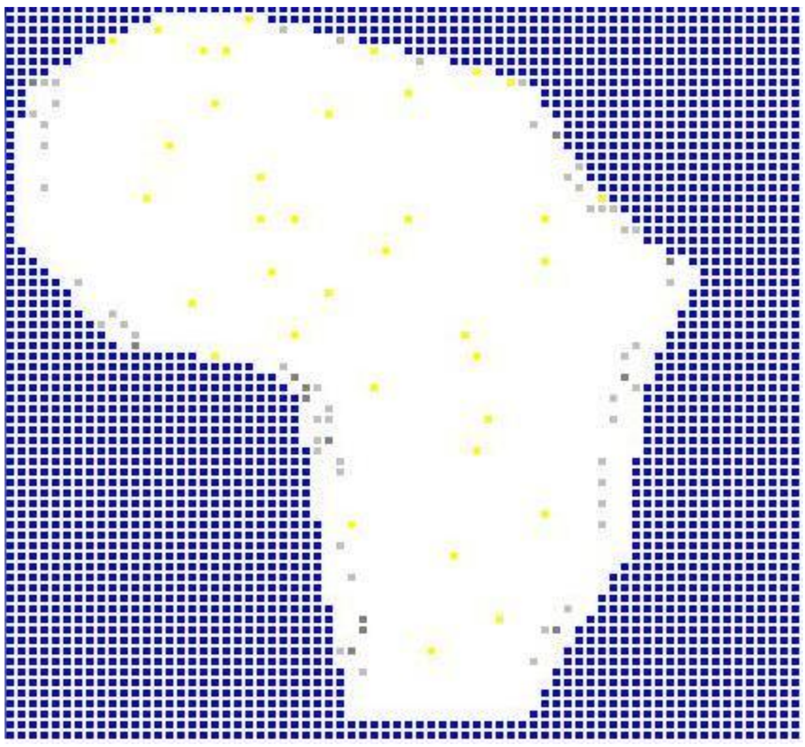
Figure 5-4

2. All the agents consume resources according to their metabolism.

3. Agents that have reached their maximum age or are out of resources, die and are replaced by new agents, randomly placed on available land sites and with initial starting values.

4. The process starts anew.

A picture of the model is given in figure 5-4 without a river, and the full code for the model is listed in the appendix. This picture is taken after the model has run for 2500 turns and we can clearly see a convergence towards the coastal sites, where the resources are placed. Agents are here colored according to how many trades they have completed, with the yellow agents the fewest and the dark gray the most trades.

5.3.1 Note on resource allocation

It should be noted that the placement of the resources on the map is not at all realistic, or according to any kind of empirics. Since where the resources are placed basically determines the location of the agents, this of course determines to some extent how the model behaves. Studies of SSA show that only 21%[5] of the population lives less than 100 km from the sea (Bloom and Sachs, 1998: 239). That means most of the population lives far from the sea. We are thus seeing the effects of uniting those that live on one side and those that live on another, without concern for the middle of the continent. The model lets us observe the effect of uniting two closed economies so to speak and monitoring the process of interaction.

[5] More precisely: " around 19 percent, lives within 100 km of the coast. If one also includes populations close to rivers navigable by ocean-going vessels, this figure only rises to 21 percent".

6 Results

This chapter presents the results of the various model experiments[6]. Results for several configurations are reported along with how many simulations were run. A typical experiment was conducted by first setting the appropriate parameter values, specifying the amount of simulations that were to be run and then letting the program produce the simulations. The results were then imported into an Excel spreadsheet and analyzed there. One experiment usually involved ten simulations that each lasted for 2500 turns. This was done in order to make it possible to observe asymptotical properties of the simulations and after around 2500 the simulations seemed to have settled into patterns, as will become apparent from viewing the evolution of different statistics. To obtain estimates for values like the average age, the following procedure was used: The last 100 values of each simulation in an experiment were averaged, this averaged value of agent age was then averaged again with the nine other simulations of that experiment. This ensures that the values reported are normally asymptotic. (Again, assuming the simulations of the same experiment resemble draws from the same probability distribution.)

6.1 The baseline case

We start the baseline case by setting up the model as in section 5.1. and then apply the following rule system : $(\{\mathbf{G}_{[12,1,1,12]}\},\{\mathbf{AT},\mathbf{AM},\mathbf{R}_{[60,100]}\})$. Breaking down this compact notation, we have that the environment is changing according to rule $\mathbf{G}_{[12,1,1,12]}$, which means that resource one is growing back at a rate of 12 per turn on the west side and resource two is growing back at a rate of one per turn on the west side. The opposite holds for the east side. The agents trade according to **AT** and move according to **AM**, both of which are detailed in section 5.2. Lastly, the agents are replaced according to rule $\mathbf{R}_{[60,100]}$, which was the compact notation for expressing that when agents die, they are replaced by another agent at a random spot in Afriscape, with a maximum age between 60 and 100. This model setup is then without the river, since the **ARR** rule is lacking. In addition, the population is set to 100, number of turns per simulation is 2500, movement cost is set to 1, initial resource levels for each

[6] To be precise, each experiment consists of several simulations and they consist of running the program for several turns. The values reported for each experiment are then averages found by averaging over all the simulations. Each simulation of an experiment is assumed to belong to the same probability distribution as the other simulations of that same experiment.

resource is set to 20, vision is set to 4 and finally the utility weights, a and b, are both equal to 0.5. This experiment consists of ten simulations with this setup.

The first parameter to be investigated is the average population age. Since the agents are making local choices to maximize their own lifetime this is a good proxy of total welfare. From this experiment the average age turns out to be 9.54(.11)[7] turns. So this seems to be rather consistent over all the 10 simulations. For a typical simulation, the average age develops something like in the table below:

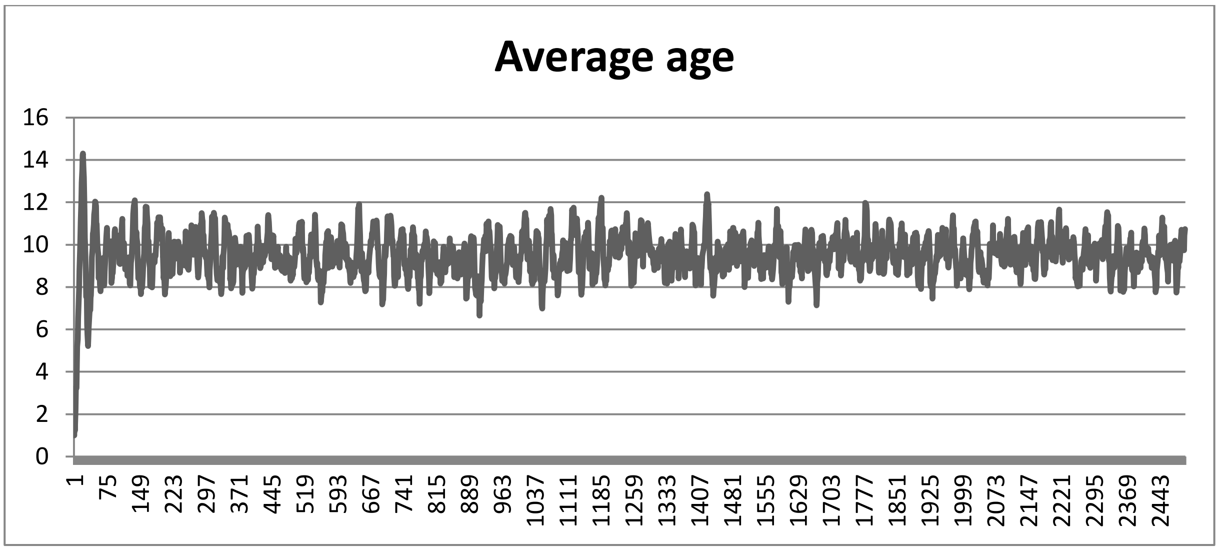

Table 6-1

As we see there is a lot of variability in this parameter, it resembles a mean reverting stochastic process. The agents have an expected maximum age of 80, yet the total expected age is just below 10. This indicates that most of the agents are replaced way before they reach maximum age and thus what is causing most deaths here is a failure to achieve the needed utility for survival.

Another variable of interest is the average traded price. The average traded price, is simply the average of all local prices being used by agents in bilateral trades. Here, the average trading price is plotted against what will be referred to as the west/east ratio. The west/east ratio is simply the number of agents that populate the western sites of the model divided by the agents that populated the eastern sites of the model. Afriscape is not symmetric and in most cases there is found to be a higher density of agents on the east side than the west. Since the number

[7] The standard deviation is given in the parenthesis.

of agents on each side of the model can influence the price, this ratio is important for understanding price dynamics.

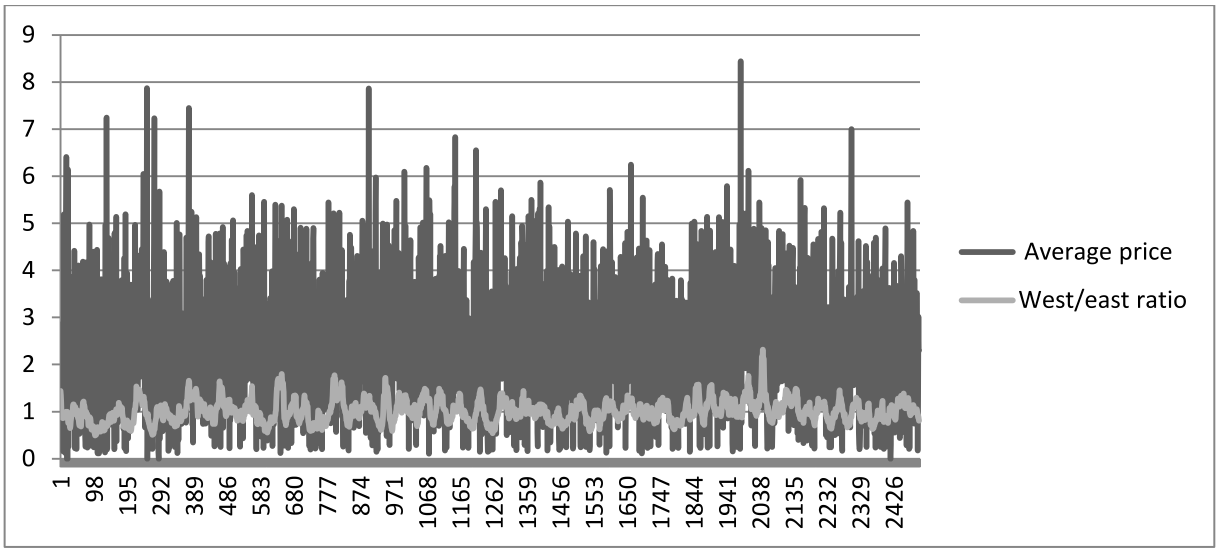

Table 6-2

As should be noted from table 6-2, the price fluctuates wildly and does not settle down to anything resembling an equilibrium. One explanation for this lies partly in the replacement rule used, and will be become apparent in the later experiments. Although there is no river in this experiment, the agents still have a reasonably high trading volume. This trading comes about mainly from the fact that when new agents are entered into the model they are given an equal amount of each resource, but they frequently meet agents who have a disproportionately high amount of one the resources. In many cases this triggers trade, since the agents will have vastly different MRSs.

6.2 Adding a river

Now everything is kept constant except we change to the system rule: ({$\mathbf{G}_{[12,1,1,12]}$, $\mathbf{ARR}_{31}$ },{**AT**,**AM**,$\mathbf{R}_{[60,100]}$}). The only difference between this experiment and the last one is that now we add the river rule. To reiterate, the river rule $\mathbf{ARR}_{31}$ simply means that at y-coordinate 31 there is a river, and all agents occupying sites that are at most two sites removed from this river can trade with any other agents who also satisfy this condition. The results for the average age are rather conclusive in favor of the river. The average age is now 23.5(1.53). This is a huge increase from the previous experiment. It bears repeating that except the inclusion of the river, the simulations in this experiment has the same initial conditions as in the baseline case. The same amount of resources are being grown, but the agents manage to profit from the east/west trade route that the river represents. From a typical run we have the following development of average age:

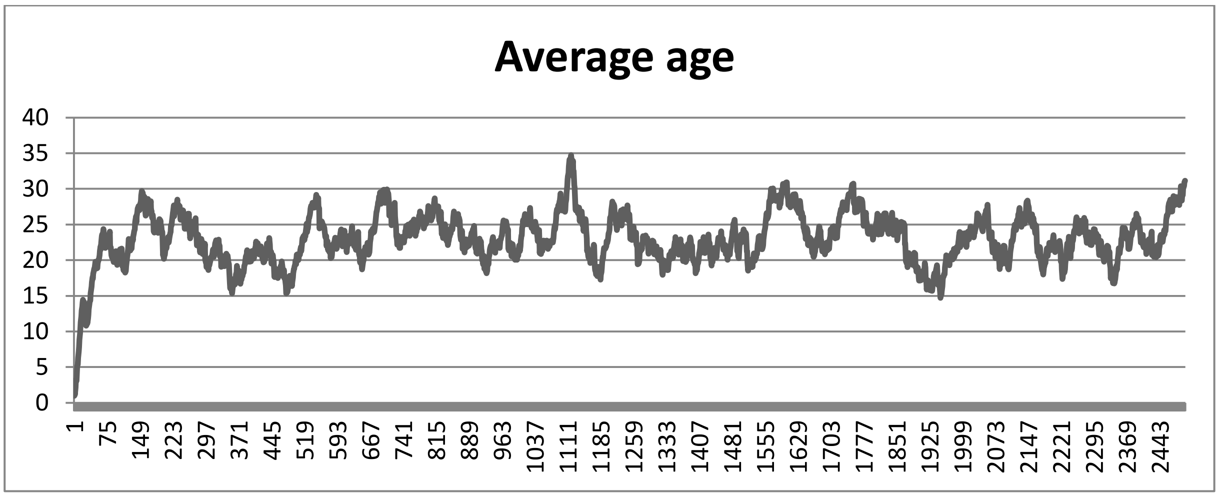

Table 6-3

As in the baseline case, the average age does not settle into an equilibrium, instead it fluctuates rather wildly between tops of 35 and lows of 15. The variation aside, it seems the agents are able to utilize the opportunity that the trade gives, and thus enabling them to more than double their average age.

Again, examining the average traded price and the west/east ratio, a typical simulation yields the following results:

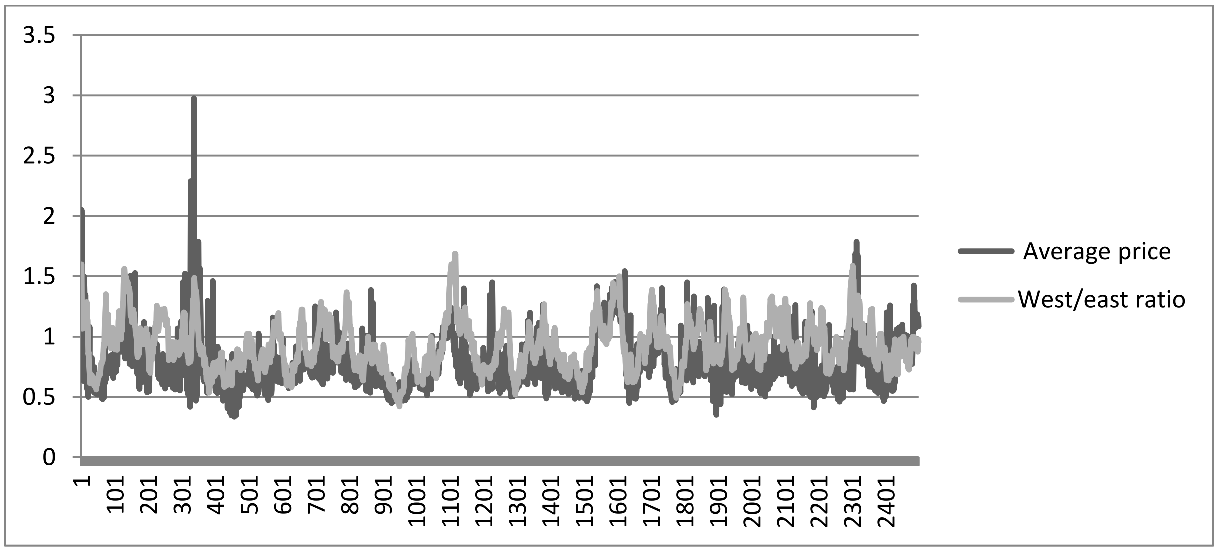

Table 6-4

The picture seen in table 6-4 is a significant departure from the erratic baseline case. Although there is still much variability in the traded price, we now see a clear correlation between the average price and the west/east ratio. This stems from the fact that asymmetry in Afriscape causes the agents to be unequally spaced on the west and east side, which again causes an imbalance in the price. This happens because the price is simply the geometric mean between the agents' MRSs, and the agents' MRSs is the product of the ratio of their respective utility weights, multiplied with the ratio of their amounts of resource one and two. With the utility weights both being equal to one half, their ratio becomes one and thus the MRS reduces to the ratio of resource one divided by resource two. Consequently, if there is a higher abundance of resource two, this causes the price to be on average below one, and this is exactly what happens when more agents are occupying the east sites than the west. If more agents are gathering resources at a place where there is a higher abundance of one of the resources, then the most abundant resource will fall in relative price. This logic holds the other way as well and is contributing to the high correlation between average traded price and the west/east ratio.

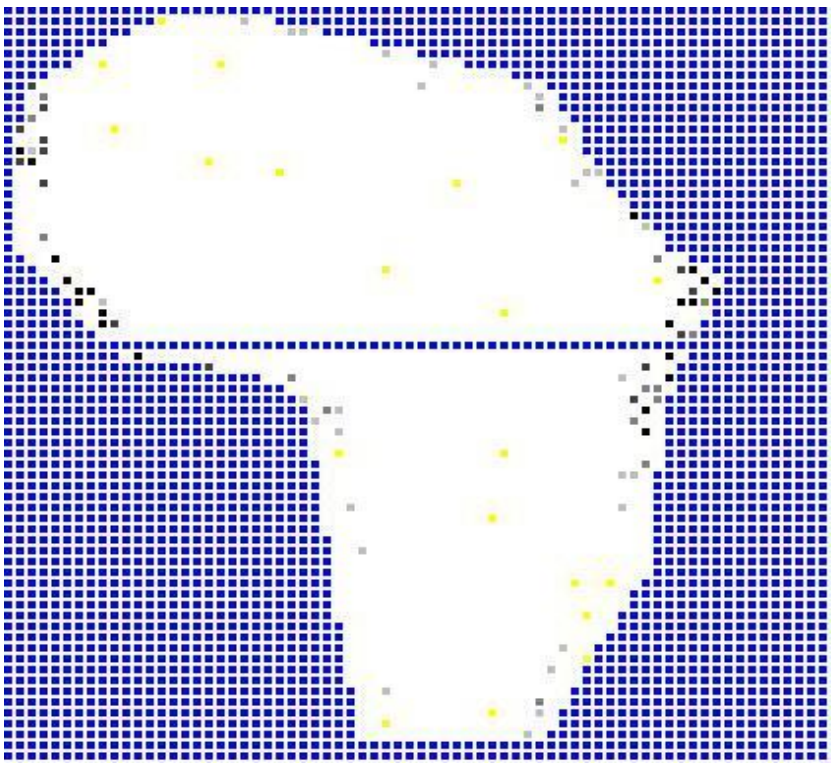
Figure 6-1

6.3 Special cases

6.3.1 Changing the growback rates

In a third experiment, I changed the growback rates so that they were equal on the west and east side of the model. This was done to investigate the effect of growback rates on the results of the two former experiments. The two experiments had the same initial parameter values as the baseline case and the baseline case with a river, except that now the growback rule changed and the resource cap on each side for both resources was set to 14. The following rules where thus used: $(\{\mathbf{G}_{[7,7,7,7]}\},\{\mathbf{AT},\mathbf{AM},\mathbf{R}_{[60,100]}\})$ and $(\{\mathbf{G}_{[7,7,7,7],}\ \mathbf{ARR}_{31}\},\{\mathbf{AT},\mathbf{AM},\mathbf{R}_{[60,100]}\})$. The growback rule $\mathbf{G}_{[7,7,7,7]}$ simply means that both on the west and east side of the model, both resources replenish at a rate of seven per turn.

Without the river the agents now had an average age of 38.5(0.84) and with the river their average age was 38.7(0.73). These results indicate that given equal resources the option for trade that the river gives does not increase average age. Another aspect of these two experiments, that at first sight seemed rather strange, was the total lack of trade. No agents made any trades in either of the experiments.

This lack of trade result could of course have been obtained analytically. Consider that each agent starts with equal amounts of each resource and the only thing making him potentially different from the other agents is his maximum age. When the agent then consumes resources he will consume the same amount of each resource and his ratio of resources will be one. Whenever the agent collects any resources he will always gather equal amounts of each resource and his MRS will still be one. Since this holds for all the agents there will never be any incentive for trade, the agents will always have equal MRSs.

6.3.2 Changing growback rates with variation in population

The former experiments spurred another set of experiments: What happens when the conditions are identical to those of section 6.3.1, except with variation in vision parameters and utility weights? Now, the agents' vision, instead of being set to 4, was an integer uniformly drawn on the set [1,4]. Also, the agents' utility weight a, was uniformly drawn on the interval [0.1,0.9] while b was set equal to 1-a.

For the following rule, ({$\mathbf{G}_{[7,7,7,7]}$},{**AT**,**AM**,$\mathbf{R}_{[60,100]}$}), i.e. without a river, an average age of 38(0.73) was obtained. A typical run had the following evolution of average age:

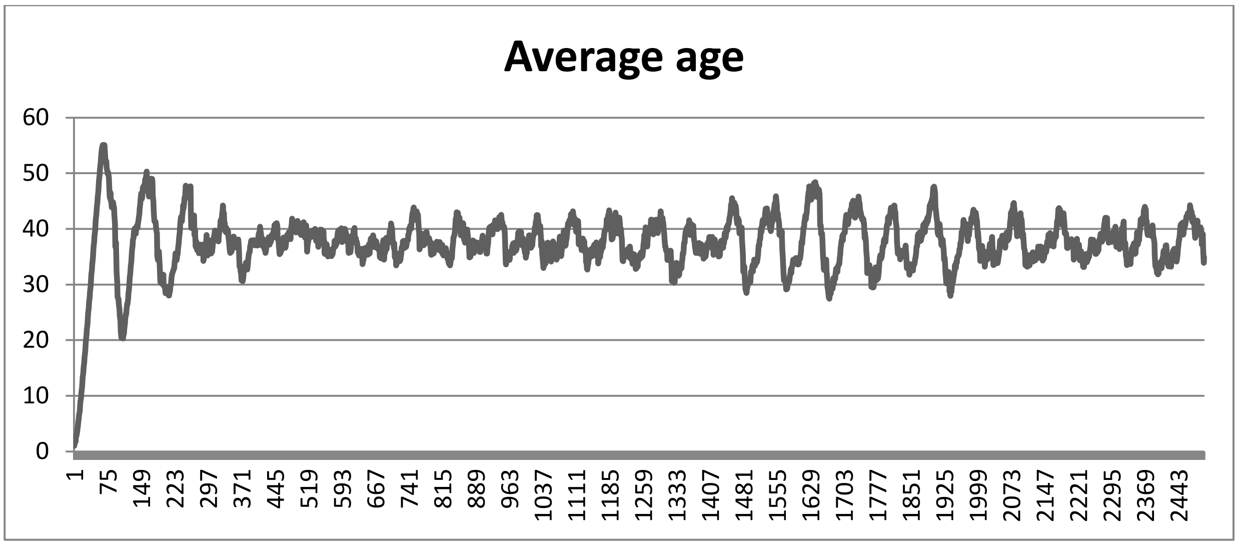

Table 6-5

Also, the trade price plotted against the west/east ratio yielded the following graph for a typical run:

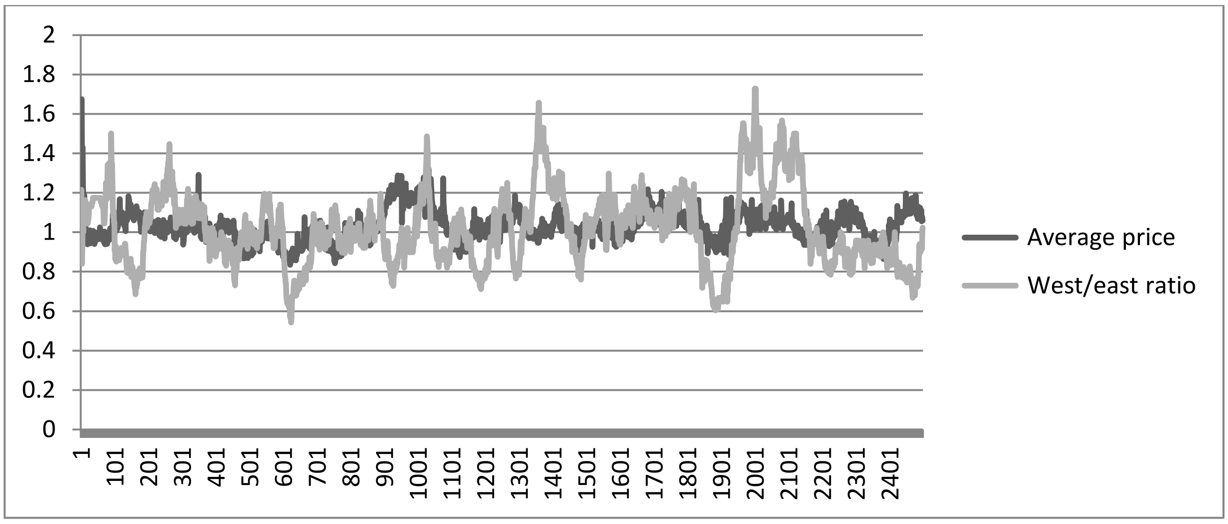

Table 6-6

We clearly observe that the price tends to one, and now there is almost no correlation between the average traded price and the west/east ratio. Of course, this happens since the resources are growing back at the same rate everywhere, and thus the location of the agents has less impact on the price. The trade that is observed in this experiment is simply generated by differences in preferences amongst the agents, implemented by the variation in their utility weights, a and b.

With the system rules ({$\mathbf{G}_{[7,7,7,7]}$, $\mathbf{ARR}_{31}$ },{**AT**,**AM**,$\mathbf{R}_{[60,100]}$}) we observe an average age of 38.2(0.7), an insignificant change compared to the case without a river. These results then show that the model needs some sort of differences in the amount of resources on each side of the river for trade to be of any value. Without differences in for example growback rates, the results clearly show the river is useless.

6.3.3 What happens when we let them live forever?

In this experiment the replacement rule $\mathbf{R}_{[60,100]}$ is exchanged for $\mathbf{R}_{[\infty]}$, i.e. the agents don't have a maximum age and are only replaced if they are not able to meet their utility needs. The initial conditions are set equal to those as in the baseline case with river except for the change in the replacement rule. Stated in the system rule form: ({$\mathbf{G}_{[12,1,1,12]}$, $\mathbf{ARR}_{31}$ },{**AT**,**AM**,$\mathbf{R}_{[\infty]}$}). These rules are closer to the neoclassical approach of often using infinitely lived agents, and the results are rather striking

Observing that average age is no longer of interest, we turn to the average traded price and the west east ratio. From a typical simulation we have the following development:

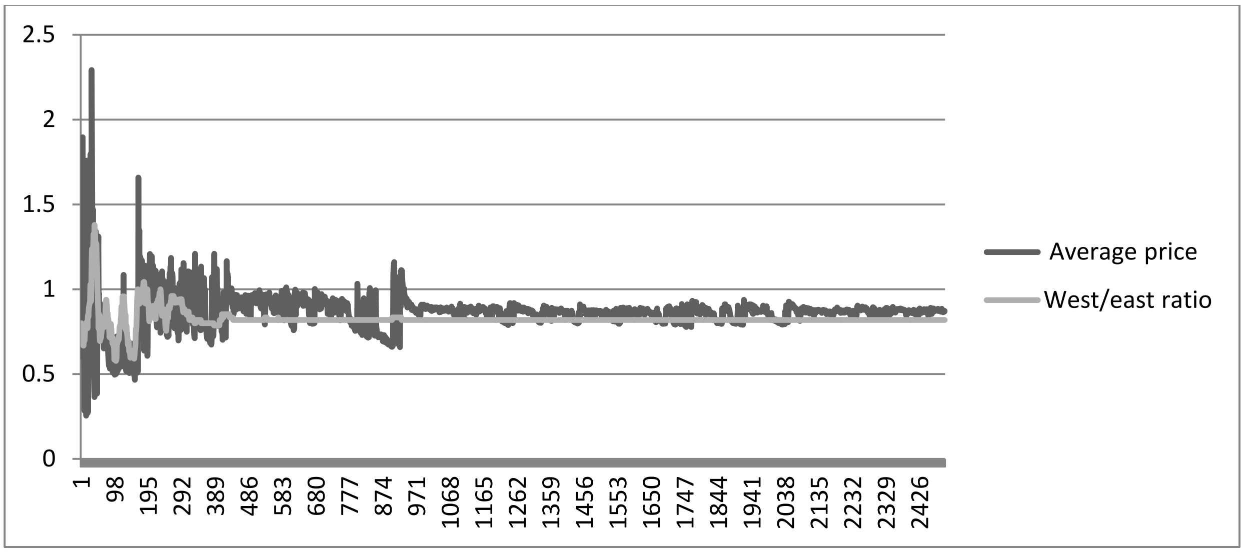

Table 6-7

This is a totally different picture than what was observed with the different replacement rule. Now, the price quickly stabilizes towards an equilibrium and the west/east ratio helps explain why the equilibrium is below one. This result is the same as was observed by Epstein and Axtell in their Sugarscape model, namely that changing the replacement rule had large consequences for the development of an equilibrium, and shows how vulnerable some of the models we construct are, concerning the initial assumptions.

6.3.4 What does the world look like when it's at max population?

Lastly, I investigate the Afriscape model when the population is multiplied by five. In this case the initial parameters are as in the baseline case with the river, except now we increase the population parameter from 100 to 500. The following systems rule is in effect: $(\{\mathbf{G}_{[12,1,1,12]}, \mathbf{ARR}_{31}\},\{\mathbf{AT},\mathbf{AM},\mathbf{R}_{[60,100]}\})$. Below is the evolution of the average age in a typical simulation.

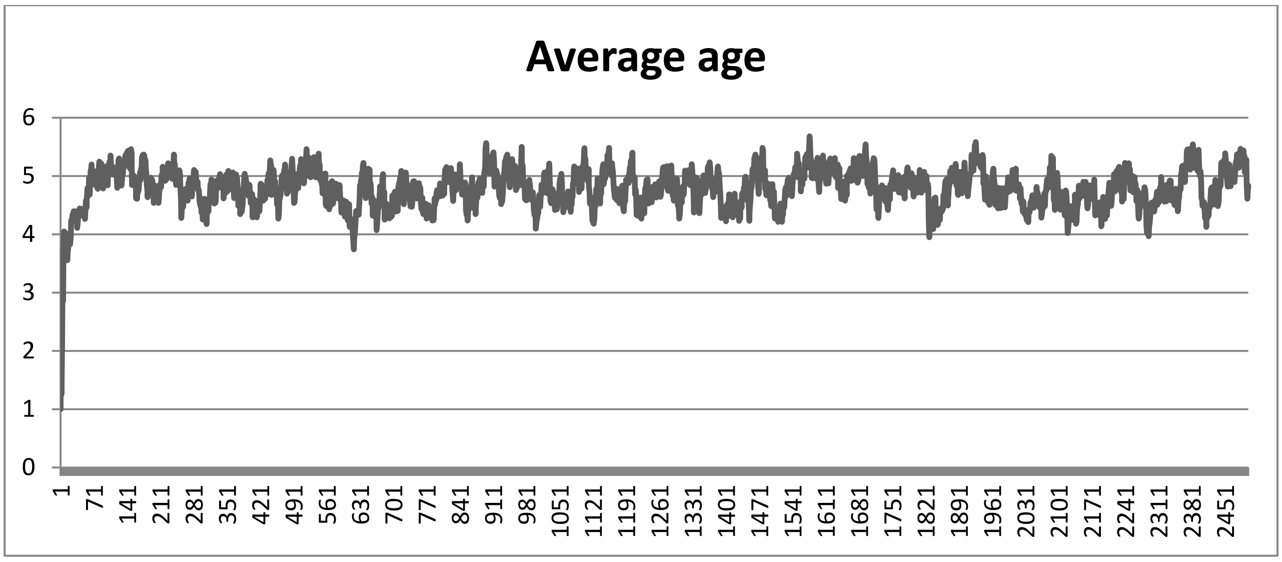

Table 6-8

The experiment yielded an average age of 4.74(0.18) which was rather low. This indicates that the system cannot sustain a large population and that most agents are replaced quickly, most never reaching their maximum age.

7 Discussion

Anyone familiar with the Sugarscape model, will quickly observe the close resemblance to Afriscape. These similarities, both in modeling technique and results add rigor to both models. That many of Afriscape's results resemble those of Sugarscape's imply that the differences used in modeling technique did not much affect the main results. This adds credibility to the models.

The main differences between the Sugarscape world and Afriscape are of course the geography and how the resources are spread, and also the utility function used by the agents to maximize their welfare. The results obtained on how the replacement rule and the evolution of the traded price effect each other, implies that the differences in the models don't matter for these results. Of these results, the strongest is the tendency towards an equilibrium in the 'neoclassical' experiments, i.e. with infinitely lived agents, but the lack of convergence with a different replacement rule. This implies care should be taken when assuming that economic agents live forever or that they have some sort of dynastical world view.

There can also be made a case for doing wider parameter sweeps. Models like this of course benefit from as much rigor as possible, and doing more experiments over a larger array of initial conditions would be a part of future work with this model.

Importance of the west/east ratio on the average price was an unforeseen consequence of distributing the resources the way I had done it. It could be interesting to observe how the results would change if the model was symmetric. Without risking too much, I believe it is a safe bet to assume the price, in the neoclassical experiment at least, would asymptotically tend to one.

There is also the case of the distribution of resources, which could have been done differently. Given more time, an ambitious project might be to plot observed resources into a map that much closer resembles that of Africa, perhaps using more advanced GIS[8] techniques. In the Afriscape model there are no resources in the middle of the continent, this is in stark contrast to reality. Again, an assumption made for simplicity. Also, simulations could be run with different types of agents, not randomly populated, but according to demographic data.

[8] GIS refers to geographical information system.

There could also have been used a replacement rule more closely resembling that of sexual reproduction. One could for example have agents produce offspring only if they satisfied certain requirements, and then have the new agent introduced into the model in the vicinity of his parents. This might lead to a convergence of agents to certain spots in the model and thereby affect the relative price and average age of agents.

One could also argue that the results presented herein could have been produced with a much simpler model, and that instead of resembling Africa the model could for example resemble ball. While this has some truth to it, once the model was developed, continuing to use it was easier than constructing another model. Although, varying the space upon which the agents move could add increased credibility to the results and thus be a venue worth exploring.

Of course, another objection to the model is the fact that rivers don’t run straight through a continent. A river is likely to start somewhere inland and then run its course to the seas. Making a river that runs in the fashion as in Afriscape is a gross simplification.

Also, coastal trading between agents could be tested. What would happen to the model if agents all along the coastal stretches could trade with each other? How might that influence the average age and price? Implementing this would probably lead to a rather large increase in the time each simulation takes. At present, the most time consuming simulations take approximately one minute, putting an experiment at roughly ten. Letting all agents on the coast trade with each other would increase the number of calculations needed in an experiment substantially.

Although some outcomes may be obvious, simulation was still fruitful, as in example 6.3.1. The analytical result became obvious once the simulation showed that zero trading had taken place. In this case there could only be one explanation. Thus, although analyzes can be faster and much easier to perform, simulation can help guide the way as to where the analyzes should be performed.

It should also be noted that the assumptions concerning the distributions of the simulations may not hold in all cases. For example, in the case where the agents are infinitely lived, the initial placement of the agents seems to stick with the population throughout the simulation. More testing would be needed to confirm this, but the results indicate that the geography of the west side causes the agents to either populate above the river, or below it, but not so often on each side of it as on the east side. This only seems to be the case when the agents don't have a maximum age.

The implication is that we are not actually drawing from the same probability distribution concerning the average ages. Once the population locks into an either north/west plus east configuration, or a south/west plus east configuration, the results from each of these cases will result in two different distributions. This means the total probability distribution will have two tops, so to speak, instead of the bell shape that is assumed.

Also, computational programs can pose a bigger challenge when it comes to checking their correctness, compared to, for example, a mathematical proof. As is apparent from the long appendix, the computer code does take a considerable time to proof read. There is a large amount of logic that needs to be implemented correctly, and one small mistake can invalidate the results.

8 Conclusion

The main aim of this paper has been to develop a model of trade and use that to test a hypothesis of whether the African continent would have been better off had there been a river running from west to east, that could be used as a trade route. The model is highly stylized and the results should therefore be interpreted with care.

The paper can roughly be divided into two parts. In the first part, theories concerning geography and growth were investigated, as well as the agent-based methodology. The Sugarscape model was an example that was studied in detail. The second part of the paper was devoted to the Afriscape model developed therein and its results. As the discussion showed, there are several issues that still remain and wider parameter sweeps could have been made.

The overall conclusion of the paper is that trade seems to be an important part of growth given that certain conditions are met. Comparing the results for experiments of baseline Afriscape with and without a river, it was clearly observed that trade between the east and the west greatly improved average age of the agent population. But, on the other hand, when the agents were given the same amount of resources and there was variation in the population, the river did not make a significant contribution to the average population age. This suggests that for trade routes to have been a factor in explaining growth, each side in a bilateral trading arrangement should have a resource the other side needs, and also an advantage in producing that resource, otherwise there would not be noticeable effects of the trade route.

The road ahead is bound to bring new insights concerning agent-based modeling within economics. ACE as a field is rather young compared to the existing methodologies and there seems to be several venues worth exploring. Concerning this model there are certainly improvements that could be made and alternative hypothesis that could be tested with it.

Concluding, it is probably wise to accept that all models are just that, models. They are not reality nor will they ever become that. All models have shortcomings and in the end it comes down to choosing which model gives the greatest insights at the least cost.

Bibliography

Acemoglu, Daron, Johnson, Simon, Robinson, James A. (2002): *Reversal of Fortune: Geography and Institutions in the Making of the Modern World Income Distribution.* The Quarterly Journal of Economics, 117 (4):1231-1294

Arthur, W. Brian (2006): *Out-of-Equilibrium Economics and Agent-based Modeling.* Handbook of Computational Economics, Vol. 2: Agent-Based Computational Economics, K. Judd and L. Tesfatsion, eds, Elsevier/North-Holland.

Axelrod, Robert. *Advancing the Art of Simulation in the Social Sciences.* Forthcoming in Handbook of Research on Nature Inspired Computing for Economy and Management, Jean-Philippe Rennard, ed., Hersey,PA: Idea Group.

Axelrold, Robert., Hammond, Ross A. (2006): *The Evolution of Ethnocentrism.* Journal of Conflict Resolution, vol. 50, No. 6, 926-936

Axtell, Robert. Epstein, Joshua M. (1996): *Growing Artificial Societies, Social Science from the Bottom Up*. MIT Press.

Bloom, David E. Sachs, Jeffrey D. Collier, Paul. Udry, Christopher (1998): *Geography, Demography, and Economic Growth in Africa.* Brookings Papers on Economic Activity, vol. 1998, No. 2: 207-295

Collier, Paul (2008): *The Bottom Billion, Why the Poorest Countries Are Failing and What Can Be Done About It.* Oxford University Press.

Cowell, Frank (2006): *Microeconomics, Principles and Analysis.* Oxford University Press Inc., New York.

Epstein, Joshua M. (2006): *Remarks on the Foundation of Agent-Based Generative Social Science.* Handbook of Computational Economics, Vol. 2: Agent-Based Computational Economics, K. Judd and L. Tesfatsion, eds, Elsevier/North-Holland.

Judd, Kenneth. Tesfatsion, L. (2006): *Handbook of Computational Economics, Vol. 2: Agent-Based Computational Economic.* Elsevier/North Holland.

Krugman, Paul (2000): *Geography and Trade*. MIT Press.

Krugman, Paul (2009): *How Did Economists Get it so Wrong?* The New York Times, Sep. 6, 2009

Miller, John H. and Page, Scott E. (2007): *Complex Adaptive Systems, an Introduction to Computational Models of Social Life*. Princeton University Press.

Schelling, Thomas C. (1978): *Micromotives and Macrobehavior.* New York: W. W. Norton.

Tesfatsion, Leigh (2003): *Agent-based computational economics: modeling economies as complex adaptive systems*. Information Sciences, 149: 263-269.

Appendix

Modeling Language

I chose to write the model in the object oriented language Java, and there are several reasons for this choice. The first reason is portability, as Java is very portable and can be run on any machine that has a Java Virtual Machine. This ensures that when the code is made publicly available anyone can run, test and modify the code. Secondly Java is a full-fledged programming language which opens up many options in terms of further development. The only restrictions to the model, thus becomes the progammer’s limited programming skills and imagination. Thirdly, when creating a relatively complex program, writing the code yourself ensures that you understand the logic behind it. Lastly, Java runs reasonably fast compared to the alternatives, which for me was either NetLogo or Matlab. Faster execution time is of course always positive when doing large and complex simulations, especially when there are several variables which need to be tested. NetLogo is a simulation toolkit that has most of the features needed to make rather complex simulations. Matlab was also an option because I had written a similar program in that language earlier. The main drawback with Matlab is that you need an installation of the program to run the code which increases the chance that someone who wants to review the code will not be able to.

Programming

Concerning the programming, there are in retrospect several things could have been written differently. First of all it could have been more user friendly. At the moment it takes some time to read it through to understand most of it. More variables could have been initialized from the main sim2 file that runs the program. Also the output should have been organized to make it easier to import. At the moment each simulation produces its own output file which needs to be imported into excel. This was fine when I was initially experimenting with the program and testing out the various parameters, but once I was running bigger experiments importing each simulation into excel became tedious.

Code

```
/*
 * This is the grid class. When we initialize the grid objects, they become
the space
 * in which the agents move. All grid objects have values of each resource.
 *
 *
 */

public class Grid {

      // the state variables of the grid object are declared here
      private boolean land;
      private double rate1;
      private double rate2;
      private int cap1;
      private int cap2;
      private double current1;
      private double current2;
      private boolean occupied;
      private int x_coord;
      private int y_coord;
      private double movementCost;
      private int agent_number;

      // this constructor creates an object of the grid class without any
args
      public Grid() {
            this.land = false;
            this.rate1 = 1;
            this.rate2 = 1;
            this.x_coord = 0;
            this.y_coord = 0;
            this.cap1 = 0;
            this.cap2 = 0;
            this.current1 = cap1;
            this.current2 = cap2;
            this.occupied = false;
            this.movementCost = 1;
            this.agent_number = 99999;
      }
      // this method prints info about the grid object
      public void displayInfo() {
            System.out.println("The x coordinate of this grid object is: "
+ x_coord);
            System.out.println("The y coordinate of this grid object is: "
+ y_coord);
            System.out.println("The growback rate of resourse 1: " +
rate1);
            System.out.println("The growback rate of resourse 2: " +
rate2);
            System.out.println("The resource cap of resourse 1: " + cap1);
            System.out.println("The resource cap of resourse 2: " + cap2);
            System.out.println("The current value of resourse 1: " +
current1);
            System.out.println("The current value of resourse 2: " +
current2);
```

```
            System.out.println("The current occupied status of the grid: "
+ occupied);
            if (land == false ) {
                  System.out.println("This is a water tile");
            } //end if
            else {
                  System.out.println("The is a land tile");
            } // end else

      } // end displayInfo()
      // this method sets the x coordinate of the grid objects
      private void set_x_coordinate(int j) {
            x_coord = j;
      }
      // this method sets the x coordinate of the grid objects
      private void set_y_coordinate(int i) {
            y_coord = i;
      }
      //this method sets the value of the resource
      public void set_resource1(double x) {
            current1 = x;
      }
      public void set_agent_number(int x) {
            agent_number = x;
      }
      //this method sets the value of the resource
      public void set_resource2(double x) {
            current2 = x;
      }
      // this method returns the value of resource 1
      public double get_resource1() {
            return current1;
      }
      // this method returns the value of resource 2
      public double get_resource2() {
            return current2;
      }
      // this method returns the movementCost of the grid object
      public double get_movementCost() {
            return movementCost;
      }
      // this method returns the x_coordinate of the grid object
      public int get_x_coordinate() {
            return x_coord;
      }
      // this method returns the y_coordinate of the grid object
      public int get_y_coordinate() {
            return y_coord;
      }
      // this method returns the value of the land status
      public boolean get_land_status() {
            return land;
      }
      // this method returns the value of the occupied status
      public boolean get_occupied_status() {
            return occupied;
      }
      //this method returns the agent_nubmer
      public int get_agent_number() {
            return agent_number;
      }
      // this method sets the grid object occupied
      public void set_occupied() {
            occupied = true;
      }
      public void set_vacant() {
            occupied = false;
      }
      // this method changes the occupied status of the grid object
      public void change_occupied() {
            if (occupied == true) {
                  occupied = false;
            } //end if
            else {
                  occupied = true;
            } //end else

      } //end change_occupied
      private void set_rate1(double r1) {
            rate1 = r1;
      }
      private void set_rate2(double r2) {
            rate2 = r2;
      }
      public static Grid[][] startupG(int arraysize, int river_latitude,
boolean river) {
            // this line creates the space
            Grid[][] space = new Grid [arraysize][arraysize];
            // this loop populates the space
            int i = 0;
            int j = 0;
            while (i<arraysize) {
                  while (j<arraysize) {
                        space[i][j] = new Grid();
                        j++;
                  } // end while j loop
                  j = 0;
                  i++;
            } // end while i loop
            // this method call gives all the grid objects coordinates
            Grid.initialize_coordinates(arraysize, space);
            // this method call assigns all the proper land values
            Grid.mapping_Africa(space,river_latitude, river);
            // this method reassigns all the cap values of the resources
            Grid.mapping_resources(space);

            return space;
      }
      // this creates a vector with all the tiles next to the river, agents
can trade across it and all along it
      public static Grid[] mapping_next_to_river(Grid[][] space1, int
river_latitude) {
            // all land tiles above the river are to be added
            int i =0;
            int j =0;
            for (i =0; i<70;i++) {
                  if (space1[river_latitude+1][i].get_land_status() ==
true) {
                        j++;
                  }
                  if (space1[river_latitude-1][i].get_land_status() ==
true) {
```

```
                              j++;
                        }
                        // the following if loops should only be included when
there is enhanced trade on the river
                        if (space1[river_latitude+2][i].get_land_status() ==
true) {
                              j++;
                        }
                        if (space1[river_latitude-2][i].get_land_status() ==
true) {
                              j++;
                        }
                  }
                  Grid[] next_to_river = new Grid[j];
                  j=0;
                  for (i =0; i<70;i++) {
                        if (space1[river_latitude+1][i].get_land_status() ==
true) {
                              next_to_river[j]= space1[river_latitude+1][i];
                              j++;
                        }
                        if (space1[river_latitude-1][i].get_land_status() ==
true) {
                              next_to_river[j]= space1[river_latitude+1][i];
                              j++;
                        }
                        if (space1[river_latitude+2][i].get_land_status() ==
true) {
                              next_to_river[j]= space1[river_latitude+1][i];
                              j++;
                        }
                        if (space1[river_latitude-2][i].get_land_status() ==
true) {
                              next_to_river[j]= space1[river_latitude+1][i];
                              j++;
                        }
                  }

                  // all land tiles below the river need to added...
                  return next_to_river;
            }

            // this is grow_back1 its called once in every game loop and makes
calls on growback2 when needed
            public static void grow_back1(int arraysize, Grid[][] space) {
                  int counter4 = 0;
                  int counter5 = 0;
                  while (counter4<arraysize) {
                        while (counter5<arraysize) {
                              if (space[counter4][counter5].get_land_status() ==
true) {
                                    space[counter4][counter5].grow_back2();
                              }
                              counter5++;
                        } // end while j loop
                        counter5 = 0;
                        counter4++;
                  } // end while i loop

            }
```

```
      // this method is called to grow back the resources at a given rate
if they are less than cap
      private void grow_back2() {
            current1 = Math.min(current1 + rate1,cap1);
            current2 = Math.min(current2 + rate2,cap2);
      } // end grow_back
      // this method changes the cap of resource 1
      public void changeCap1(int cap1) {
            this.cap1 = cap1;
      }
      // this method changes the cap of resource 2
      public void changeCap2(int cap2) {
            this.cap2 = cap2;
      }
      //this method changes the value of the land variable to true
      private void set_land() {
            land = true;
      }
      // this method changes the value of the land variable to false
      private void set_water() {
            land = false;
      }
      // this method assigns each grid object its x and y coordinates
      public static final void initialize_coordinates(int arraysize, Grid
space[][]) {
            int i = 0;
            int j = 0;
            while (i<arraysize) {
                  while (j<arraysize) {
                        space[i][j].set_x_coordinate(j);
                        space[i][j].set_y_coordinate(i);
                        j++;
                  } // end while j loop
                  j = 0;
                  i++;
            } // end while i loop

      } // end initialize_coordinates
      // mapping the resources, first west side then east side
      public static void mapping_resources(Grid space[][]) {
            int i = 1;
            int j = 1;

            // the following while loops are used in the baseline case
            while (i<70) {
                  while (j<70) {
                        if (space[i][(j-1)].get_land_status() == false &&
space[i][j].get_land_status() == true) {
                              space[i][j].changeCap1(24); //random integer
in interval [1,6]
                              space[i][j].changeCap2(2); //random integer in
interval [1,2]
                              space[i][j+1].changeCap1(24);
                              space[i][j+1].changeCap2(2);
                              space[i][j+2].changeCap1(24);
                              space[i][j+2].changeCap2(2);
                              space[i][j].current1 = space[i][j].cap1;
      // set the current equal to cap
                              space[i][j].current2 = space[i][j].cap2; //
set the current equal to cap
```

```
                              space[i][j+1].current1 = space[i][j+1].cap1;
      // set the current equal to cap
                              space[i][j+1].current2 = space[i][j+1].cap2;
      // set the current equal to cap
                              space[i][j+2].current1 = space[i][j+2].cap1;
      // set the current equal to cap
                              space[i][j+2].current2 = space[i][j+2].cap2;
      // set the current equal to cap
                              space[i][j].set_rate1(12);
                              space[i][j+1].set_rate1(12);
                              space[i][j+2].set_rate1(12);
                        } // this ends the west side
                        if (space[i][j-1].get_land_status() == true &&
space[i][j].get_land_status() == false) {
                              space[i][j-1].changeCap1(2);
                              space[i][j-1].changeCap2(24);
                              space[i][j-2].changeCap1(2);
                              space[i][j-2].changeCap2(24);
                              space[i][j-3].changeCap1(2);
                              space[i][j-3].changeCap2(24);
                              space[i][j-1].current1 = space[i][j].cap1;
      // set the current equal to cap
                              space[i][j-1].current2 = space[i][j].cap2;
      // set the current equal to cap
                              space[i][j-2].current1 = space[i][j-2].cap1;
      // set the current equal to cap
                              space[i][j-2].current2 = space[i][j-2].cap2;
      // set the current equal to cap
                              space[i][j-3].current1 = space[i][j-3].cap1;
      // set the current equal to cap
                              space[i][j-3].current2 = space[i][j-3].cap2;
      // set the current equal to cap
                              space[i][j-1].set_rate2(12);
                              space[i][j-2].set_rate2(12);
                              space[i][j-3].set_rate2(12);
                        } // this ends the east side
                        if (space[i-1][j-1].get_land_status() == false ) {
                              space[i-1][j-1].changeCap1(0);
                              space[i-1][j-1].changeCap2(0);
                              space[i-1][j-1].current1 = space[i-1][j-
1].cap1;    // set the current equal to cap
                              space[i-1][j-1].current2 = space[i-1][j-
1].cap2;    // set the current equal to cap
                        } // ends setting all water tiles to 0

                        j++;
                  } // end while j loop
                  j = 1;
                  i++;
            } // end while i loop

            /*
            // this while loop is the non-baseline. Here we have equal
growth rates on each side
            while (i<70) {
                  while (j<70) {
                        if (space[i][(j-1)].get_land_status() == false &&
space[i][j].get_land_status() == true) {
                              space[i][j].changeCap1(14);
                              space[i][j].changeCap2(14);
                              space[i][j+1].changeCap1(14);
                              space[i][j+1].changeCap2(14);
                              space[i][j+2].changeCap1(14);
                              space[i][j+2].changeCap2(14);
                              space[i][j].current1 = space[i][j].cap1;
      // set the current equal to cap
                              space[i][j].current2 = space[i][j].cap2; //
set the current equal to cap
                              space[i][j+1].current1 = space[i][j+1].cap1;
      // set the current equal to cap
                              space[i][j+1].current2 = space[i][j+1].cap2;
      // set the current equal to cap
                              space[i][j+2].current1 = space[i][j+2].cap1;
      // set the current equal to cap
                              space[i][j+2].current2 = space[i][j+2].cap2;
      // set the current equal to cap
                              space[i][j].set_rate1(7);
                              space[i][j+1].set_rate1(7);
                              space[i][j+2].set_rate1(7);
                              space[i][j].set_rate2(7);
                              space[i][j+1].set_rate2(7);
                              space[i][j+2].set_rate2(7);
                        } // this ends the west side
                        if (space[i][j-1].get_land_status() == true &&
space[i][j].get_land_status() == false) {
                              space[i][j-1].changeCap1(14);
                              space[i][j-1].changeCap2(14);
                              space[i][j-2].changeCap1(14);
                              space[i][j-2].changeCap2(14);
                              space[i][j-3].changeCap1(14);
                              space[i][j-3].changeCap2(14);
                              space[i][j-1].current1 = space[i][j].cap1;
      // set the current equal to cap
                              space[i][j-1].current2 = space[i][j].cap2;
      // set the current equal to cap
                              space[i][j-2].current1 = space[i][j-2].cap1;
      // set the current equal to cap
                              space[i][j-2].current2 = space[i][j-2].cap2;
      // set the current equal to cap
                              space[i][j-3].current1 = space[i][j-3].cap1;
      // set the current equal to cap
                              space[i][j-3].current2 = space[i][j-3].cap2;
      // set the current equal to cap
                              space[i][j-1].set_rate2(7);
                              space[i][j-2].set_rate2(7);
                              space[i][j-3].set_rate2(7);
                              space[i][j-1].set_rate1(7);
                              space[i][j-2].set_rate1(7);
                              space[i][j-3].set_rate1(7);
                        } // this ends the east side
                        if (space[i-1][j-1].get_land_status() == false ) {
                              space[i-1][j-1].changeCap1(0);
                              space[i-1][j-1].changeCap2(0);
                              space[i-1][j-1].current1 = space[i-1][j-
1].cap1;    // set the current equal to cap
                              space[i-1][j-1].current2 = space[i-1][j-
1].cap2;    // set the current equal to cap
                        } // ends setting all water tiles to 0

                        j++;
                  } // end while j loop
                  j = 1;
```

```
                        i++;
                } // end while i loop

                 */
                // this loop is for the comparative advantage case.. not used
                /*
                while (i<70) {
                        while (j<70) {
                                if (space[i][(j-1)].get_land_status() == false &&
space[i][j].get_land_status() == true) {
                                        space[i][j].changeCap1(24);
                                        space[i][j].changeCap2(10);
                                        space[i][j+1].changeCap1(24);
                                        space[i][j+1].changeCap2(10);
                                        space[i][j+2].changeCap1(24);
                                        space[i][j+2].changeCap2(10);
                                        space[i][j].current1 = space[i][j].cap1;
        // set the current equal to cap
                                        space[i][j].current2 = space[i][j].cap2; //
set the current equal to cap
                                        space[i][j+1].current1 = space[i][j+1].cap1;
        // set the current equal to cap
                                        space[i][j+1].current2 = space[i][j+1].cap2;
        // set the current equal to cap
                                        space[i][j+2].current1 = space[i][j+2].cap1;
        // set the current equal to cap
                                        space[i][j+2].current2 = space[i][j+2].cap2;
        // set the current equal to cap
                                        space[i][j].set_rate1(12);
                                        space[i][j+1].set_rate1(12);
                                        space[i][j+2].set_rate1(12);
                                        space[i][j].set_rate2(5);
                                        space[i][j+1].set_rate2(5);
                                        space[i][j+2].set_rate2(5);
                                } // this ends the west side
                                if (space[i][j-1].get_land_status() == true &&
space[i][j].get_land_status() == false) {
                                        space[i][j-1].changeCap1(12);
                                        space[i][j-1].changeCap2(8);
                                        space[i][j-2].changeCap1(12);
                                        space[i][j-2].changeCap2(8);
                                        space[i][j-3].changeCap1(12);
                                        space[i][j-3].changeCap2(8);
                                        space[i][j-1].current1 = space[i][j].cap1;
        // set the current equal to cap
                                        space[i][j-1].current2 = space[i][j].cap2;
        // set the current equal to cap
                                        space[i][j-2].current1 = space[i][j-2].cap1;
        // set the current equal to cap
                                        space[i][j-2].current2 = space[i][j-2].cap2;
        // set the current equal to cap
                                        space[i][j-3].current1 = space[i][j-3].cap1;
        // set the current equal to cap
                                        space[i][j-3].current2 = space[i][j-3].cap2;
        // set the current equal to cap
                                        space[i][j-1].set_rate1(6);
                                        space[i][j-2].set_rate1(6);
                                        space[i][j-3].set_rate1(6);
                                        space[i][j-1].set_rate2(4);
                                        space[i][j-2].set_rate2(4);
                                        space[i][j-3].set_rate2(4);
                        } // this ends the east side
                        if (space[i-1][j-1].get_land_status() == false ) {
                              space[i-1][j-1].changeCap1(0);
                              space[i-1][j-1].changeCap2(0);
                              space[i-1][j-1].current1 = space[i-1][j-
1].cap1;    // set the current equal to cap
                              space[i-1][j-1].current2 = space[i-1][j-
1].cap2;    // set the current equal to cap
                        } // ends setting all water tiles to 0

                        j++;
                  } // end while j loop
                  j = 1;
                  i++;
            } // end while i loop
             */

            for (j = 0; j<70; j++){
                  space[69][j].changeCap1(0);
                  space[69][j].changeCap2(0);
                  space[69][j].current1 = space[69][j].cap1;     // set the
current equal to cap
                  space[69][j].current2 = space[69][j].cap2;     // set the
current equal to cap

            }
            for (j = 0; j<70; j++){
                  space[j][69].changeCap1(0);
                  space[j][69].changeCap2(0);
                  space[j][69].current1 = space[j][69].cap1;     // set the
current equal to cap
                  space[j][69].current2 = space[j][69].cap2;     // set the
current equal to cap

            }
      }
      /** this method sets the values of
       * the land attribute such that the grid will resemble
       * the African continent... unfortunately for me, I did not bother to
find
       * an elegant way to solve this problem...as you see from the rather
cumbersome code.
       */
      public static void mapping_Africa(Grid[][] space, int river_latitude,
boolean river) {
            int i = 1;
            for (int j = 13; j<23; j++) {
                  space[i][j].set_land();
            } // end i = 1
            i = 2;
            for (int j = 11; j<27; j++) {
                  space[i][j].set_land();
            } // end i = 2
            i = 3;
            for (int j = 9; j<31; j++) {
                  space[i][j].set_land();
            } // end i = 3
            i = 4;
            for (int j = 7; j<35; j++) {
                  space[i][j].set_land();
            } // end i = 4
```

```
i = 5;
for (int j = 6; j<39; j++) {
      space[i][j].set_land();
} // end i = 5
i = 6;
for (int j = 4; j<43; j++) {
      space[i][j].set_land();
} // end i = 6
i = 7;
for (int j = 2; j<46; j++) {
      space[i][j].set_land();
} // end i = 7
i = 8;
for (int j = 2; j<47; j++) {
      space[i][j].set_land();
} // end i = 8
i = 9;
for (int j = 2; j<47; j++) {
      space[i][j].set_land();
} // end i = 9
i = 10;
for (int j = 2; j<48; j++) {
      space[i][j].set_land();
} // end i = 10
i = 11;
for (int j = 1; j<49; j++) {
      space[i][j].set_land();
} // end i = 11
i = 12;
for (int j = 1; j<49; j++) {
      space[i][j].set_land();
} // end i = 12
i = 13;
for (int j = 1; j<49; j++) {
      space[i][j].set_land();
} // end i = 13
i = 14;
for (int j = 1; j<50; j++) {
      space[i][j].set_land();
} // end i = 14
i = 15;
for (int j = 1; j<51; j++) {
      space[i][j].set_land();
} // end i = 15
i = 16;
for (int j = 1; j<51; j++) {
      space[i][j].set_land();
} // end i = 16
i = 17;
for (int j = 1; j<52; j++) {
      space[i][j].set_land();
} // end i = 17
i = 18;
for (int j = 1; j<53; j++) {
      space[i][j].set_land();
} // end i = 18
i = 19;
for (int j = 1; j<54; j++) {
      space[i][j].set_land();
} // end i = 19
i = 20;
```

```
for (int j = 1; j<55; j++) {
      space[i][j].set_land();
} // end i = 20
i = 21;
for (int j = 1; j<56; j++) {
      space[i][j].set_land();
} // end i = 21
i = 22;
for (int j = 1; j<56; j++) {
      space[i][j].set_land();
} // end i = 22
i = 23;
for (int j = 2; j<58; j++) {
      space[i][j].set_land();
} // end i = 23
i = 24;
for (int j = 3; j<60; j++) {
      space[i][j].set_land();
} // end i = 24
i = 25;
for (int j = 4; j<61; j++) {
      space[i][j].set_land();
} // end i = 25
i = 26;
for (int j = 5; j<61; j++) {
      space[i][j].set_land();
} // end i = 26
i = 27;
for (int j = 6; j<60; j++) {
      space[i][j].set_land();
} // end i = 27
i = 28;
for (int j = 6; j<60; j++) {
      space[i][j].set_land();
} // end i = 28
i = 29;
for (int j = 7; j<59; j++) {
      space[i][j].set_land();
} // end i = 29
i = 30;
for (int j = 8; j<59; j++) {
      space[i][j].set_land();
} // end i = 30
i = 31;
for (int j = 9; j<58; j++) {
      space[i][j].set_land();
} // end i = 31
i = 32;
for (int j = 10; j<58; j++) {
      space[i][j].set_land();
} // end i = 32
i = 33;
for (int j = 17; j<57; j++) {
      space[i][j].set_land();
} // end i = 33
i = 34;
for (int j = 22; j<57; j++) {
      space[i][j].set_land();
} // end i = 34
i = 35;
for (int j = 24; j<57; j++) {
        space[i][j].set_land();
} // end i = 35
i = 36;
for (int j = 25; j<56; j++) {
        space[i][j].set_land();
} // end i = 36
i = 37;
for (int j = 26; j<56; j++) {
        space[i][j].set_land();
} // end i = 37
i = 38;
for (int j = 26; j<56; j++) {
        space[i][j].set_land();
} // end i = 38
i = 39;
for (int j = 26; j<56; j++) {
        space[i][j].set_land();
} // end i = 39
i = 40;
for (int j = 26; j<56; j++) {
        space[i][j].set_land();
} // end i = 40
i = 41;
for (int j = 26; j<56; j++) {
        space[i][j].set_land();
} // end i = 41
i = 42;
for (int j = 27; j<56; j++) {
        space[i][j].set_land();
} // end i = 42
i = 43;
for (int j = 27; j<55; j++) {
        space[i][j].set_land();
} // end i = 43
i = 44;
for (int j = 27; j<55; j++) {
        space[i][j].set_land();
} // end i = 44
i = 45;
for (int j = 27; j<55; j++) {
        space[i][j].set_land();
} // end i = 45
i = 46;
for (int j = 27; j<55; j++) {
        space[i][j].set_land();
} // end i = 46
i = 47;
for (int j = 27; j<55; j++) {
        space[i][j].set_land();
} // end i = 47
i = 48;
for (int j = 27; j<55; j++) {
        space[i][j].set_land();
} // end i = 48
i = 49;
for (int j = 28; j<55; j++) {
        space[i][j].set_land();
} // end i = 49
i = 50;
for (int j = 28; j<55; j++) {
        space[i][j].set_land();
```

```
} // end i = 50
i = 51;
for (int j = 28; j<55; j++) {
      space[i][j].set_land();
} // end i = 51
i = 52;
for (int j = 28; j<55; j++) {
      space[i][j].set_land();
} // end i = 52
i = 53;
for (int j = 28; j<54; j++) {
      space[i][j].set_land();
} // end i = 53
i = 54;
for (int j = 28; j<53; j++) {
      space[i][j].set_land();
} // end i = 54
i = 55;
for (int j = 28; j<53; j++) {
      space[i][j].set_land();
} // end i = 55
i = 56;
for (int j = 28; j<52; j++) {
      space[i][j].set_land();
} // end i = 56
i = 57;
for (int j = 29; j<52; j++) {
      space[i][j].set_land();
} // end i = 57
i = 58;
for (int j = 29; j<51; j++) {
      space[i][j].set_land();
} // end i = 58
i = 59;
for (int j = 29; j<50; j++) {
      space[i][j].set_land();
} // end i = 59
i = 60;
for (int j = 29; j<50; j++) {
      space[i][j].set_land();
} // end i = 60
i = 61;
for (int j = 29; j<49; j++) {
      space[i][j].set_land();
} // end i = 61
i = 62;
for (int j = 29; j<49; j++) {
      space[i][j].set_land();
} // end i = 62
i = 63;
for (int j = 30; j<48; j++) {
      space[i][j].set_land();
} // end i = 63
i = 64;
for (int j = 30; j<48; j++) {
      space[i][j].set_land();
} // end i = 64
i = 65;
for (int j = 30; j<47; j++) {
      space[i][j].set_land();
} // end i = 65
```

```
            i = 66;
            for (int j = 30; j<47; j++) {
                  space[i][j].set_land();
            } // end i = 66
            i = 67;
            for (int j = 30; j<46; j++) {
                  space[i][j].set_land();
            } // end i = 67
            if (river == true) {
                  i = river_latitude;
                  for (int j = 0; j<70; j++) {
                        space[i][j].set_water();
                  }

            }
      } // end mapping_Africa()

} // end class Grid
```

```
 import java.util.*;


public class Agent {
      // the state variables for the agent class
      private int x_coord;
      private int y_coord;
      private int color;
      private int age;
      private int max_age;
      //private double metabolism1;
      //private double metabolism2;
      private double metabolism;
      private double resource1;
      private double resource2;
      private double a;
      private double b;
      private int vision;
      private int trades;
      private double total_util_this_round;
      private double util_needs;
      private double util_gained_from_eating;  // test variable.. should be
equal to util_needs

      Random gen2 = new Random();

      // the constructor method for the agents, without args
      public Agent() {
            x_coord= 35;
            y_coord= 35;
            color = 10;
            age = 0;
            max_age = (int) (60 + Math.random()* 41);
            //metabolism1 = .2; //this dictates the amount of resource the
agent consumes.. .1 = 10%..
```

```
            //metabolism2 = .2;
            resource1 = 20;
            resource2 = 20;
            a = .1+Math.random()*.8;  // .5;
            b = 1-a ; // .5;
            vision = (int) (1 + Math.random() *4); // 4;
            trades =0;
            total_util_this_round =0;
            util_needs = 2;
            util_gained_from_eating=0;
      }
      // this method displays info about the agent
      public void displayInfo() {

            System.out.print("X : " + x_coord);
            System.out.print(" Y: " + y_coord);
            System.out.print(" R1 : " + resource1);
            System.out.print(" R2 : " + resource2);
            System.out.print(" Max age: " + max_age);
            System.out.println(" Age: " + age);
      }

      //the utility function of the agents.. it represents how much utility
the agents get from consuming parts of their total wealth
      // when considering whether or not to move this is the function the
agents view against each other
      // it is of course also what the agents view when they consider
whether or not to make any given trade.. will this trade make me ha a
higher utility from consumption this period
      // if utility from completing a trade is higher than not then do it.
      //NEW note.. the agent no longer consumes in this way.. he now only
needs to achieve one specific utility level to survive and he does
      //so by consuming an equal percentage of each of his resources.. This
percentage which we call metabolism is computed when the util level u is
know.
      public double util(double resource1, double resource2) {
            double util = (a * Math.log(resource1)) + (b *
Math.log(resource2));
            return util;
      }
      public double get_util() {
            double util = (this.a * Math.log(resource1)) + (this.b *
Math.log(resource2));
            return util;
      }
      public double get_metabolism(double util,double resource1,double
resource2) {
            metabolism = (Math.exp(util_needs))/(Math.pow(resource1,
a)*Math.pow(resource2,b));
            return metabolism;
      }
      public double get_metabolism() {
            metabolism = (Math.exp(util_needs))/(Math.pow(resource1,
a)*Math.pow(resource2,b));
            return metabolism;
      }
      public double util_gained_from_eating_this_round() {
            metabolism = (Math.exp(util_needs))/(Math.pow(resource1,
a)*Math.pow(resource2,b));
            util_gained_from_eating = a * Math.log(metabolism*resource1) +
b * Math.log(metabolism*resource2);
```

```
		return util_gained_from_eating;
	}
	public void set_util_gained_from_eating(double input) {
		util_gained_from_eating = input;
	}
	public double get_util_gained_from_eating() {
		return util_gained_from_eating;
	}

	// this method returns the agents MRS12, their marginal rate of 
substitution of good 1 for good 2.
	public double get_MRS12() {
		double MRS12 = (this.resource1/this.resource2) * 
(this.b/this.a);
		return MRS12;
	}
	// this method returns the agents MRS12 but takes arguments
	public double get_MRS12(double res1,double res2) {
		double MRS12 = (res1/res2) * (this.b/this.a);
		return MRS12;
	}
	// this method gets the total_util_this_round
	public double get_total_util_this_round() {
		return total_util_this_round;
	}
	// this method updates the total_util_this round
	public void add_to_total_util_gained_this_round(double inc) {
		total_util_this_round = total_util_this_round+ inc;
	}
	// this method sets the total_util_this round
	public void set_total_util_gained_this_round(double inc) {
		total_util_this_round = inc;
	}
	//this shows the number of trades
	public int get_trades() {
		return trades;
	}
	// when called, this method adds one to the trade attribute
	public void trade() {
		trades++;
	}
	// this method sets the amount of resource
	public void set_resource1(double res1) {
		resource1 = res1;
	}
	// this method sets the amount of resource
	public void set_resource2(double res2) {
		resource2 = res2;
	}
	// this method returns the amount of the resource the agent object 
has
	public double get_resource1() {
		return resource1;
	}
	// this method returns the amount of the resource the agent object 
has
	public double get_resource2() {
		return resource2;
	}
	// this method returns the x_coordinate of the grid object
	public int get_x_coordinate() {
```

```
            return x_coord;
        }
        // this method returns the y_coordinate of the grid object
        public int get_y_coordinate() {
            return y_coord;
        }
        // this method returns the agents vision
        private int get_vision() {
            return vision;
        }
        // this method returns the age of the agent object
        public int get_age() {
            return age;
        }
        // this method returns the max age of the agent object
        public int get_max_age() {
            return max_age;
        }
        // this method returns the color of the agent object
        public int get_color() {
            return color;
        }
        // this method ages the agent by 1
        public void get_older() {
            age++;
        }
        // this method changes the coordinates of the agent object
        public void moveAgent(int x_coord, int y_coord) {
            this.set_x_coordinate(x_coord);
            this.set_y_coordinate(y_coord);
        }
        // this method finds the agents vacant spots in the grid and places
them there
        public void get_spot(Grid[][] space,int ag) {
            int r = (int) ( Math.random()* 70);  // random integer in
interval [0,69]
            int v = (int) ( Math.random()* 70);
            while ((space[r][v].get_occupied_status() == true ||
space[r][v].get_land_status() == false)) {
                r = (int) ( Math.random()* 70);
                v = (int) ( Math.random()* 70);
            } // end while loop searching for good spot
            // move the agent to the spot
            this.moveAgent( v, r);
            //set the spot to occupied
            space[r][v].set_occupied();
            // set the agent number
            space[r][v].set_agent_number(ag);
        }
        // this method sets the x coordinate of the agent objects
        private void set_x_coordinate(int x_coord) {
            this.x_coord = x_coord;
        }
        // this method sets the y coordinate of the agent objects
        private void set_y_coordinate(int y_coord) {
            this.y_coord = y_coord;
        }
        public static Agent[] startupA(Grid[][] space, int population_size) {
            int i = 0;
            // this line creates and an array of agents
            Agent[] agent = new Agent[population_size];
            // this loop populates the space
            while (i<population_size) {
                  agent[i] = new Agent();
                  i++;
            } // end while i loop
            // this loop gives all the agents a spot on one of the
unoccupied land tiles and sets it equal to occupied
            for (i = 0; i<population_size ;i++) {
                  agent[i].get_spot(space,i);
            } // end for loop giving agents spots
            // array with agents and
            return agent;
      } // startup method end

      // this is called when the agent consume resources
      public void eat() {
            metabolism = (Math.exp(util_needs))/(Math.pow(resource1,
a)*Math.pow(resource2,b));
            resource1 = resource1*(1 - metabolism);
            resource2 = resource2*(1 - metabolism);


      }
      // this method takes as arguments the coordinates of the agent and
the whole space
      //and then generates a 16*9 array called moore_array
      //this should of course been generate_von_neumann neighborhood.. too
lazy to correct that now...
      public double[][] generate_moore_neighborhood(int i, Agent[] agent,
Grid[][] space) {
            //going north
            double[][] moore_array = new double[agent[i].get_vision() *
4][9];
            int k = agent[i].get_y_coordinate();
            int r = 0;
            int v = 0;
            while (k > (agent[i].get_y_coordinate() -
agent[i].get_vision()) && k > 0) {
                  moore_array[0+r][0] =  space[k-
1][agent[i].get_x_coordinate()].get_x_coordinate();
                  moore_array[0+r][1] =  space[k-
1][agent[i].get_x_coordinate()].get_y_coordinate();
                  if (space[k-
1][agent[i].get_x_coordinate()].get_land_status() == true) {
                        moore_array[0+r][2] = 1;
                  }
                  moore_array[0+r][3] =  space[k-
1][agent[i].get_x_coordinate()].get_movementCost();
                  moore_array[0+r][4] =  moore_array[0+r][3];
                                                //total movement cost
column
                  moore_array[0+r][5] =  space[k-
1][agent[i].get_x_coordinate()].get_resource1();          // column 6 res1
                  moore_array[0+r][6] =  space[k-
1][agent[i].get_x_coordinate()].get_resource2();     //column 7 res2
                  moore_array[0+r][7] =
agent[i].util(moore_array[0+r][5]+agent[i].resource1-
(moore_array[0+r][4]/2),moore_array[0+r][6]+agent[i].resource2-
(moore_array[0+r][4]/2));          //column 8 utility gained at this spot
```

```
                    if (space[k-
1][agent[i].get_x_coordinate()].get_occupied_status() == true) {
                            moore_array[0+r][8] = 1;
                    }
                    r++;
                    k = k-1;
            }
            // this loop adds the previous total movement cost to the
current and sets it to total
            while (v+1 < r) {
                    moore_array[0+v+1][4] = moore_array[0+v][3] +
moore_array[0+v][4];
                    moore_array[0+v+1][7] =
agent[i].util(moore_array[0+v+1][5]+agent[i].resource1-
(moore_array[0+v+1][4]/2),moore_array[0+v+1][6]+agent[i].resource2-
(moore_array[0+v+1][4]/2));        //column 8 utility gained at this spot

                    v++;
            }
            //going south
            k = agent[i].get_y_coordinate();
            r = agent[i].get_vision();
            v = r;
            while (k < (agent[i].get_y_coordinate() +
agent[i].get_vision()) && k < 69 ) {
                    moore_array[0+r][0] =
space[k+1][agent[i].get_x_coordinate()].get_x_coordinate();
                    moore_array[0+r][1] =
space[k+1][agent[i].get_x_coordinate()].get_y_coordinate();
                    if
(space[k+1][agent[i].get_x_coordinate()].get_land_status() == true) {
                            moore_array[0+r][2] = 1;
                    }
                    moore_array[0+r][3] =
space[k+1][agent[i].get_x_coordinate()].get_movementCost();
                    moore_array[0+r][4] =  moore_array[0+r][3];
                                                     //total movement cost
column
                    moore_array[0+r][5] =
space[k+1][agent[i].get_x_coordinate()].get_resource1();         // column
6 res1
                    moore_array[0+r][6] =
space[k+1][agent[i].get_x_coordinate()].get_resource2();   //column 7 res2
                    moore_array[0+r][7] =
agent[i].util(moore_array[0+r][5]+agent[i].resource1-
(moore_array[0+r][4]/2),moore_array[0+r][6]+agent[i].resource2-
(moore_array[0+r][4]/2));          //column 8 utility gained at this spot

                    if
(space[k+1][agent[i].get_x_coordinate()].get_occupied_status() == true) {
                            moore_array[0+r][8] = 1;
                    }
                    r++;
                    k++;
            }
            // this loop adds the previous total movement cost to the
current and sets it to total
            while (v+1 < r) {
                    moore_array[0+v+1][4] = moore_array[0+v][3] +
moore_array[0+v][4];
```

```
                  moore_array[0+v+1][7] =
agent[i].util(moore_array[0+v+1][5]+agent[i].resource1-
(moore_array[0+v+1][4]/2),moore_array[0+v+1][6]+agent[i].resource2-
(moore_array[0+v+1][4]/2));
                  v++;
            }
            //going west
            k = agent[i].get_x_coordinate();
            r = agent[i].get_vision() * 2;
            v = r;
            while (k > (agent[i].get_x_coordinate() -
agent[i].get_vision()) && k > 0) {
                  moore_array[0+r][0] =
space[agent[i].get_y_coordinate()][k-1].get_x_coordinate();
                  moore_array[0+r][1] =
space[agent[i].get_y_coordinate()][k-1].get_y_coordinate();
                  if (space[agent[i].get_y_coordinate()][k-
1].get_land_status() == true) {
                        moore_array[0+r][2] = 1;
                  }
                  moore_array[0+r][3] =
space[agent[i].get_y_coordinate()][k-1].get_movementCost();
                  moore_array[0+r][4] =  moore_array[0+r][3];
                                                     //total movement cost
column
                  moore_array[0+r][5] =
space[agent[i].get_y_coordinate()][k-1].get_resource1();         // column
6 res1
                  moore_array[0+r][6] =
space[agent[i].get_y_coordinate()][k-1].get_resource2();   //column 7 res2
                  moore_array[0+r][7] =
agent[i].util(moore_array[0+r][5]+agent[i].resource1-
(moore_array[0+r][4]/2),moore_array[0+r][6]+agent[i].resource2-
(moore_array[0+r][4]/2));          //column 8 utility gained at this spot

                  if (space[agent[i].get_y_coordinate()][k-
1].get_occupied_status() == true) {
                        moore_array[0+r][8] = 1;
                  }
                  r++;
                  k = k-1;
            }
            // this loop adds the previous total movement cost to the
current and sets it to total
            while (v+1 < r) {
                  moore_array[0+v+1][4] = moore_array[0+v][3] +
moore_array[0+v][4];
                  moore_array[0+v+1][7] =
agent[i].util(moore_array[0+v+1][5]+agent[i].resource1-
(moore_array[0+v+1][4]/2),moore_array[0+v+1][6]+agent[i].resource2-
(moore_array[0+v+1][4]/2));
                  v++;
            }
            //going east
            k = agent[i].get_x_coordinate();
            r = agent[i].get_vision() * 3;
            v = r;
            while (k < (agent[i].get_x_coordinate() +
agent[i].get_vision()) && k < 69) {
                  moore_array[0+r][0] =
space[agent[i].get_y_coordinate()][k+1].get_x_coordinate();
```

```
                              moore_array[0+r][1] =
space[agent[i].get_y_coordinate()][k+1].get_y_coordinate();
                              if
(space[agent[i].get_y_coordinate()][k+1].get_land_status() == true) {
                                        moore_array[0+r][2] = 1;
                              }
                              moore_array[0+r][3] =
space[agent[i].get_y_coordinate()][k+1].get_movementCost();
                              moore_array[0+r][4] =  moore_array[0+r][3];
                                                            //total movement cost
column
                              moore_array[0+r][5] =
space[agent[i].get_y_coordinate()][k+1].get_resource1();          // column
6 res1
                              moore_array[0+r][6] =
space[agent[i].get_y_coordinate()][k+1].get_resource2();   //column 7 res2
                              moore_array[0+r][7] =
agent[i].util(moore_array[0+r][5]+agent[i].resource1-
(moore_array[0+r][4]/2),moore_array[0+r][6]+agent[i].resource2-
(moore_array[0+r][4]/2));           //column 8 utility gained at this spot

                              if
(space[agent[i].get_y_coordinate()][k+1].get_occupied_status() == true) {
                                        moore_array[0+r][8] = 1;
                              }
                              r++;
                              k++;
                    }
                    // this loop adds the previous total movement cost to the
current and sets it to total

                    while (v+1 < r) {
                              moore_array[0+v+1][4] = moore_array[0+v][3] +
moore_array[0+v][4];
                              moore_array[0+v+1][7] =
agent[i].util(moore_array[0+v+1][5]+agent[i].resource1-
(moore_array[0+v+1][4]/2),moore_array[0+v+1][6]+agent[i].resource2-
(moore_array[0+v+1][4]/2));
                              v++;
                    }
                    return moore_array;

          } // end generate moore_neigborhood method
          // this method takes the moore_array as args and returns the optimal
tile for the agent to move to
          //this loop runs through all the utilities and finds the largest
value
          public double[] optimal_tile(int w,double[][] moore_array, Agent[]
agent) {
                    int k = agent[w].get_vision() * 4;
                    int r = 0;
                    double highest_for_now = 0;
                    // this loop finds the highest of possible values
                    while (r < k) {
                              if ((highest_for_now <= moore_array[r][7]) &&
(moore_array[r][8] != 1)) {
                                        highest_for_now = moore_array[r][7];
                              }
                              r++;
                    }
                    r = 0;
```

```
            int counter1 = 0;
            // this loop counts how many times the largest value occurs
            while (r < k) {
                  if ((highest_for_now == moore_array[r][7]) &&
(moore_array[r][8] != 1)) {
                        counter1++;
                  }
                  r++;
            }
            double[][] high = new double[counter1][4];
            double[] optimal = new double[4];
            r = 0;
            int j = 0;
            // this loop generates the values and spots in the new array
            while (r < k) {
                  if ((highest_for_now == moore_array[r][7]) &&
(moore_array[r][8] != 1)) {
                        high[j][0] = moore_array[r][0] ;
                        high[j][1] = moore_array[r][1] ;
                        high[j][2] = highest_for_now ;            // this is
the highest for now value
                        high[j][3] = moore_array[r][4];     // this is the
total movementcost
                        j++;
                  }
                  r++;
            }
            // i need to randomize over the possible values this was one
way to do it

            /*
            try {
                  j = gen2.nextInt(counter1);
                  optimal[0] = high[j][0];
                  optimal[1] = high[j][1];
                  optimal[2] = high[j][2];
                  optimal[3] = high[j][3];
                  }
            catch (IllegalArgumentException e) {
                  optimal[0] = agent[w].get_x_coordinate();
                  optimal[1] = agent[w].get_y_coordinate();
                  optimal[2] = agent[w].util(agent[w].get_resource1(),
agent[w].get_resource2());
                  optimal[3] = 0;

            }
             */
            j = (int) Math.random()*counter1;
            if (counter1>0){
                  optimal[0] = high[j][0];
                  optimal[1] = high[j][1];
                  optimal[2] = high[j][2];
                  optimal[3] = high[j][3];
            }
            else {
                  optimal[0] = agent[w].get_x_coordinate();
                  optimal[1] = agent[w].get_y_coordinate();
                  optimal[2] = agent[w].util(agent[w].get_resource1(),
agent[w].get_resource2());
                  optimal[3] = 0;
            }
            return optimal;
      }

      // this method generates a shuffled array with length k

      public static int[] shuffle(int k) {
            // TODO Auto-generated method stub
            // this is the testrandom method class

            // Random rgen = new Random();  // Random number generator
            int[] index = new int[k];

            //--- Initialize the array to the ints 0-51
            for (int i=1; i < index.length; i++) {
                  index[i] = i ;
            }

            //--- Shuffle by exchanging each element randomly
            for (int i=0; i < index.length; i++) {
                  int randomPosition = (int) (Math.random()
*(index.length));
                  int temp = index[i];
                  index[i] = index[randomPosition];
                  index[randomPosition] = temp;
            } // end for loop
            return index;

      } // end shuffle
      public void trading_rule(Grid[] trade_move,Agent[] agent, int[]
shuffledlist,int n,int j,statistics stat1) {
            int [] rlist = Agent.shuffle(trade_move.length);
            double max_util_gain =0;
            for (n =0; n<trade_move.length;n++) {
                  if (trade_move[rlist[n]].get_occupied_status() == true) {
                        // compute own and others mrs
                        double own_MRS12 =
agent[shuffledlist[j]].get_MRS12();
                        double other_MRS12 =
agent[trade_move[rlist[n]].get_agent_number()].get_MRS12();
                        double price = Math.sqrt(own_MRS12*other_MRS12);
                        if (own_MRS12 > other_MRS12) {
                              // now direction of trade must be resource 1
from own_agent and he then receives resource 2
                              // i.e. if owns willingness to pay for
resource 2 is higher than others willingness to pay for 2
                              if (price >= 1) {
                                    // p units of resource 1 is exchanged
for 1 unit of resource 2
                                    double new_own_resource1 =
agent[shuffledlist[j]].get_resource1()-price;
                                    double new_own_resource2 =
agent[shuffledlist[j]].get_resource2()+1;
                                    double new_other_resource1 =
agent[trade_move[rlist[n]].get_agent_number()].get_resource1()+price ;
                                    double new_other_resource2 =
agent[trade_move[rlist[n]].get_agent_number()].get_resource2()-1;
                                    double new_own_util =
agent[shuffledlist[j]].util(new_own_resource1, new_own_resource2);
```

```
                                        double new_other_util =
agent[trade_move[rlist[n]].get_agent_number()].util(new_other_resource1,
new_other_resource2);
                                        double new_own_MRS12 =
agent[shuffledlist[j]].get_MRS12(new_own_resource1,new_own_resource2);
                                        double new_other_MRS12 =
agent[trade_move[rlist[n]].get_agent_number()].get_MRS12(new_other_resource
1,new_other_resource2);
                                        if ((new_own_MRS12 > new_other_MRS12) &&
((new_own_util > agent[shuffledlist[j]].get_util()) && (new_other_util >
agent[trade_move[rlist[n]].get_agent_number()].get_util()))) {
                                              if (max_util_gain < new_own_util -
agent[shuffledlist[j]].get_util()) {
                                                    max_util_gain = new_own_util
- agent[shuffledlist[j]].get_util();
                                              }

      agent[shuffledlist[j]].set_resource1(new_own_resource1);

      agent[shuffledlist[j]].set_resource2(new_own_resource2);

      agent[trade_move[rlist[n]].get_agent_number()].set_resource1(new_othe
r_resource1);

      agent[trade_move[rlist[n]].get_agent_number()].set_resource2(new_othe
r_resource2);
                                              agent[shuffledlist[j]].trade();

      agent[trade_move[rlist[n]].get_agent_number()].trade();

      stat1.set_per_round_trade_volume(stat1.get_per_round_trade_volume()+p
rice);

      stat1.set_turn_trades(1+stat1.get_turn_trades());

      stat1.set_per_round_total_price(stat1.get_total_turn_price() +price);
                                        }
                                  }
                                  else {
                                        //  1 unit of resource 1 is exchanged
for 1/p units of resource 2
                                        double new_own_resource1 =
agent[shuffledlist[j]].get_resource1()-1;
                                        double new_own_resource2 =
agent[shuffledlist[j]].get_resource2()+1/price;
                                        double new_other_resource1 =
agent[trade_move[rlist[n]].get_agent_number()].get_resource1()+1 ;
                                        double new_other_resource2 =
agent[trade_move[rlist[n]].get_agent_number()].get_resource2()-1/price;
                                        double new_own_util =
agent[shuffledlist[j]].util(new_own_resource1, new_own_resource2);
                                        double new_other_util =
agent[trade_move[rlist[n]].get_agent_number()].util(new_other_resource1,
new_other_resource2);
                                        double new_own_MRS12 =
agent[shuffledlist[j]].get_MRS12(new_own_resource1,new_own_resource2);
                                        double new_other_MRS12 =
agent[trade_move[rlist[n]].get_agent_number()].get_MRS12(new_other_resource
1,new_other_resource2);
```

```
                                    if ((new_own_MRS12 > new_other_MRS12) &&
((new_own_util > agent[shuffledlist[j]].get_util()) && (new_other_util >
agent[trade_move[rlist[n]].get_agent_number()].get_util()))) {
                                          if (max_util_gain < new_own_util -
agent[shuffledlist[j]].get_util()) {
                                                max_util_gain = new_own_util
- agent[shuffledlist[j]].get_util();
                                          }

      agent[shuffledlist[j]].set_resource1(new_own_resource1);

      agent[shuffledlist[j]].set_resource2(new_own_resource2);

      agent[trade_move[rlist[n]].get_agent_number()].set_resource1(new_othe
r_resource1);

      agent[trade_move[rlist[n]].get_agent_number()].set_resource2(new_othe
r_resource2);
                                          agent[shuffledlist[j]].trade();

      agent[trade_move[rlist[n]].get_agent_number()].trade();

      stat1.set_per_round_trade_volume(stat1.get_per_round_trade_volume()+1
);

      stat1.set_turn_trades(1+stat1.get_turn_trades());

      stat1.set_per_round_total_price(stat1.get_total_turn_price() +price);
                                    }
                              }

                        }
                        else if (own_MRS12 < other_MRS12) {
                              // now direction of trade must be resource 2
from own_agent and he then receives resource 1
                              // i.e. if owns willingness to pay for
resource 1 is higher than others willingness to pay for 1
                              if (price >= 1) {
                                    // p units of resource 1 is exchanged
for 1 unit of resource 2
                                    double new_own_resource1 =
agent[shuffledlist[j]].get_resource1()+price;
                                    double new_own_resource2 =
agent[shuffledlist[j]].get_resource2()-1;
                                    double new_other_resource1 =
agent[trade_move[rlist[n]].get_agent_number()].get_resource1()-price ;
                                    double new_other_resource2 =
agent[trade_move[rlist[n]].get_agent_number()].get_resource2()+1;
                                    double new_own_util =
agent[shuffledlist[j]].util(new_own_resource1, new_own_resource2);
                                    double new_other_util =
agent[trade_move[rlist[n]].get_agent_number()].util(new_other_resource1,
new_other_resource2);
                                    double new_own_MRS12 =
agent[shuffledlist[j]].get_MRS12(new_own_resource1,new_own_resource2);
                                    double new_other_MRS12 =
agent[trade_move[rlist[n]].get_agent_number()].get_MRS12(new_other_resource
1,new_other_resource2);
                                    if ((new_own_MRS12 < new_other_MRS12) &&
((new_own_util > agent[shuffledlist[j]].get_util()) && (new_other_util >
agent[trade_move[rlist[n]].get_agent_number()].get_util()))) {
```

```
                                                    if (max_util_gain < new_own_util -
agent[shuffledlist[j]].get_util()) {
                                                          max_util_gain = new_own_util
- agent[shuffledlist[j]].get_util();
                                                    }

      agent[shuffledlist[j]].set_resource1(new_own_resource1);

      agent[shuffledlist[j]].set_resource2(new_own_resource2);

      agent[trade_move[rlist[n]].get_agent_number()].set_resource1(new_othe
r_resource1);

      agent[trade_move[rlist[n]].get_agent_number()].set_resource2(new_othe
r_resource2);
                                                    agent[shuffledlist[j]].trade();

      agent[trade_move[rlist[n]].get_agent_number()].trade();

      stat1.set_per_round_trade_volume(stat1.get_per_round_trade_volume()+p
rice);

      stat1.set_turn_trades(1+stat1.get_turn_trades());

      stat1.set_per_round_total_price(stat1.get_total_turn_price() +price);
                                              }
                                        }
                                        else {
                                              //  1 unit of resource 1 is exchanged
for 1/p units of resource 2
                                              double new_own_resource1 =
agent[shuffledlist[j]].get_resource1()+1;
                                              double new_own_resource2 =
agent[shuffledlist[j]].get_resource2()-1/price;
                                              double new_other_resource1 =
agent[trade_move[rlist[n]].get_agent_number()].get_resource1()-1 ;
                                              double new_other_resource2 =
agent[trade_move[rlist[n]].get_agent_number()].get_resource2()+1/price;
                                              double new_own_util =
agent[shuffledlist[j]].util(new_own_resource1, new_own_resource2);
                                              double new_other_util =
agent[trade_move[rlist[n]].get_agent_number()].util(new_other_resource1,
new_other_resource2);
                                              double new_own_MRS12 =
agent[shuffledlist[j]].get_MRS12(new_own_resource1,new_own_resource2);
                                              double new_other_MRS12 =
agent[trade_move[rlist[n]].get_agent_number()].get_MRS12(new_other_resource
1,new_other_resource2);
                                              if ((new_own_MRS12 < new_other_MRS12) &&
((new_own_util > agent[shuffledlist[j]].get_util()) && (new_other_util >
agent[trade_move[rlist[n]].get_agent_number()].get_util()))) {
                                                    if (max_util_gain < new_own_util -
agent[shuffledlist[j]].get_util()) {
                                                          max_util_gain = new_own_util
- agent[shuffledlist[j]].get_util();
                                                    }

      agent[shuffledlist[j]].set_resource1(new_own_resource1);

      agent[shuffledlist[j]].set_resource2(new_own_resource2);
      agent[trade_move[rlist[n]].get_agent_number()].set_resource1(new_othe
r_resource1);

      agent[trade_move[rlist[n]].get_agent_number()].set_resource2(new_othe
r_resource2);
                                          agent[shuffledlist[j]].trade();

      agent[trade_move[rlist[n]].get_agent_number()].trade();

      stat1.set_per_round_trade_volume(stat1.get_per_round_trade_volume()+1
);

      stat1.set_turn_trades(1+stat1.get_turn_trades());

      stat1.set_per_round_total_price(stat1.get_total_turn_price() +price);
                                                }
                                          }

                                    }
                                    else {
                                          // then own_MRS12 = other_MRS12 and do nothing
                                    }
                              }
                        }
      } // end trading rule
} //end Agent class




public class Sim1 {

      public Sim1(int population_size,Grid[][] space, Agent[] agent, int
arraysize, int river_latitude, boolean river) {
            int i = 0;
            int j = 0;
            while (i<arraysize) {
                  while (j<arraysize) {
                        space[i][j] = new Grid();
                        j++;
                  } // end while j loop
                  j = 0;
                  i++;
            } // end while i loop
            // this method call gives all the grid objects coordinates
            Grid.initialize_coordinates(arraysize, space);
            // this method call assigns all the proper land values
            Grid.mapping_Africa(space, river_latitude, river);
            // this method reassigns all the cap values of the resources
            Grid.mapping_resources(space);
            // this line creates and an array of agents
            // Agent[] agent = new Agent[population_size];
            // this loop populates the space
            i = 0;
            while (i<population_size) {
                  agent[i] = new Agent();
                  i++;
            } // end while i loop
            // this loop gives all the agents a spot on one of the
unoccupied land tiles and sets it equal to occupied
            for (i = 0; i<population_size ;i++) {
                  agent[i].get_spot(space,i);
            } // end for loop giving agents spots
      }// end of constructor
      // start of the game loop  make this its own method
      // create a combination object of both the agent array and the grid
array.. this will then be the return type of the game loop
      public static void gameLoop(Agent[] agent, Grid[][] space, int
population_size, int arraysize, boolean river, int river_latitude, Grid[]
next_to_river, statistics stat1, boolean trading, boolean repeat_trade) {

            //1. we run the grow_back rule
            Grid.grow_back1(arraysize, space);
            //2. the movement and TRADE rule.. we need to do it for all
agents in random order
            // shuffled list is a randomized list we can use to initialize
the agents
            int[] shuffledlist = Agent.shuffle(population_size);
            int j;
            int k;
            int l;
            stat1.set_per_round_trade_volume(0);
            stat1.set_turn_trades(0);
            stat1.set_trades_on_river(0);
            stat1.set_river_trade_vol(0);
            stat1.set_per_round_total_price(0);
            for (j = 0; j < population_size; j++) {
                  // the agent sets total util gained this round equal to
zero

      agent[shuffledlist[j]].set_total_util_gained_this_round(0);
                  //the agent scans all the positions around him in random
order and whenever he finds another agent he tries to init trade
                  int temp_trades = agent[shuffledlist[j]].get_trades();
                  if (trading == true) {
                        while (temp_trades ==
agent[shuffledlist[j]].get_trades()) {
                              int index_trades = temp_trades;
                              int n=0;
                              int m=0;
                              Grid [] trade_move;
                              try {
                                    trade_move = new Grid[25];
                                    for (k = 0; k < 5;k++) {
                                          for (l=0; l<5;l++) {
                                                trade_move[m] = space[k +
agent[shuffledlist[j]].get_y_coordinate()-2][l +
agent[shuffledlist[j]].get_x_coordinate()-2];
                                                m++;
                                          }
                                    }

      agent[shuffledlist[j]].trading_rule(trade_move, agent, shuffledlist,
n, j,stat1);
                              } // end try
                              // this catch loop will execute if the agent
is close to the boundary
                              catch (ArrayIndexOutOfBoundsException e) {
```

```
                              if
(agent[shuffledlist[j]].get_x_coordinate() == 1) {
                                    trade_move = new Grid[20];
                                    m=0;
                                    for (k = 0; k < 5;k++) {
                                          for (l=0; l<4;l++) {
                                                trade_move[m] =
space[k + agent[shuffledlist[j]].get_y_coordinate()-2][l +
agent[shuffledlist[j]].get_x_coordinate()-1];
                                                m++;
                                          }
                                    }

      agent[shuffledlist[j]].trading_rule(trade_move, agent, shuffledlist,
n, j,stat1);
                              }
                              if
(agent[shuffledlist[j]].get_y_coordinate() == 1) {
                                    trade_move = new Grid[20];
                                    m=0;
                                    for (k = 0; k < 4;k++) {
                                          for (l=0; l<5;l++) {
                                                trade_move[m] =
space[k + agent[shuffledlist[j]].get_y_coordinate()-1][l +
agent[shuffledlist[j]].get_x_coordinate()-2];
                                                m++;
                                          }
                                    }

      agent[shuffledlist[j]].trading_rule(trade_move, agent, shuffledlist,
n, j,stat1);
                              }
                        } // end catch
                        n= 0;
                        m=0;
                        int c6=0;
                        if (river == true) {
                              if
(agent[shuffledlist[j]].get_y_coordinate() == river_latitude+1 ||
agent[shuffledlist[j]].get_y_coordinate() == river_latitude-1 ||
agent[shuffledlist[j]].get_y_coordinate() == river_latitude+2 ||
agent[shuffledlist[j]].get_y_coordinate() == river_latitude-2) {
                                    // refresh the next_to_river array
                                    for (int c5 =0; c5<70;c5++) {
                                          if
(space[river_latitude+1][c5].get_land_status() == true) {
                                                next_to_river[c6]=
space[river_latitude+1][c5];
                                                c6++;
                                          }
                                          if (space[river_latitude-
1][c5].get_land_status() == true) {
                                                next_to_river[c6]=
space[river_latitude+1][c5];
                                                c6++;
                                          }
                                          // when these two if loops
are included the agent can trade on two tiles above and below the river
                                          if
(space[river_latitude+2][c5].get_land_status() == true) {
```

```
                                                                next_to_river[c6]=
space[river_latitude+1][c5];
                                                                c6++;
                                                          }
                                                          if (space[river_latitude-
2][c5].get_land_status() == true) {
                                                                next_to_river[c6]=
space[river_latitude+1][c5];
                                                                c6++;
                                                          }
                                                    }
                                                    // these lines add the number of
river trades to its statistic and volume of trades
                                                    double temp_river_trade_vol =
stat1.get_per_round_trade_volume();
                                                    int temp_total_trades =
stat1.get_turn_trades();


      agent[shuffledlist[j]].trading_rule(next_to_river, agent,
shuffledlist, n, j,stat1);

                                                    if (temp_total_trades <
stat1.get_turn_trades()) {

      stat1.set_trades_on_river(stat1.get_turn_trades_on_river()+(stat1.get
_turn_trades()-temp_total_trades));

      stat1.set_river_trade_vol(stat1.get_river_trade_vol()+(stat1.get_per_
round_trade_volume()-temp_river_trade_vol));
                                                    }
                                              }
                                        } // end river loop
                                        if (index_trades <
agent[shuffledlist[j]].get_trades() && repeat_trade == true) {
                                              temp_trades =
agent[shuffledlist[j]].get_trades();
                                        }
                                        else {
                                              temp_trades =
1+agent[shuffledlist[j]].get_trades();
                                        }
                                  } // end while trade loop
                            } // end of outer trade loop

                            // start of movement rule
                            double[][] moore_neigborhood =
agent[shuffledlist[j]].generate_moore_neighborhood(shuffledlist[j], agent,
space);
                            double[] highest =
agent[shuffledlist[j]].optimal_tile(shuffledlist[j], moore_neigborhood,
agent);
                            // if the agents current utility is less that what he can
get at the new highest tile after movement costs have been deducted, move
the agent to the spot
                            if
(agent[shuffledlist[j]].util(agent[shuffledlist[j]].get_resource1(),
agent[shuffledlist[j]].get_resource2()) < highest[2] ) {
                                  //the old spot needs to be set vacant and the old
spot needs to have its agent number changed back to 99999
```

```
      space[agent[shuffledlist[j]].get_y_coordinate()][agent[shuffledlist[j
]].get_x_coordinate()].set_vacant();  // the old spot is now set to vacant

      space[agent[shuffledlist[j]].get_y_coordinate()][agent[shuffledlist[j
]].get_x_coordinate()].set_agent_number(99999);
                        // the agent moves to the new spot

      agent[shuffledlist[j]].moveAgent((int)highest[0],(int)highest[1]);
                        // take current1 and current2 and add the new
resources and also deduct movement costs

      agent[shuffledlist[j]].set_resource1(agent[shuffledlist[j]].get_resou
rce1()+space[(int)highest[1]][(int)highest[0]].get_resource1()-
(highest[3]/2));

      agent[shuffledlist[j]].set_resource2(agent[shuffledlist[j]].get_resou
rce2()+space[(int)highest[1]][(int)highest[0]].get_resource2()-
(highest[3]/2));
                        // set the current on the spot equal to zero

      space[(int)highest[1]][(int)highest[0]].set_resource1(0);  //the new
spot is harvested

      space[(int)highest[1]][(int)highest[0]].set_resource2(0);

      space[(int)highest[1]][(int)highest[0]].set_occupied();  // the new
spot is now set to occupied

      space[(int)highest[1]][(int)highest[0]].set_agent_number(shuffledlist
[j]);  // the new spot is given the new agent number
                  }
            } // end movement and trade rule
            //3. the death rule
            for (k = 0;k < population_size; k++) {
                  // before the agent consumes we would like to know
exactly how much util he gains by eating this round.. this is irrelevant in
this version
                  //
agent[k].set_total_util_gained_this_round(agent[k].get_util());

      agent[k].set_util_gained_from_eating(agent[k].util_gained_from_eating
_this_round());
                  // agents consume according to metabolism
                  agent[k].eat();
                  // agents get older, ie use the age method
                  agent[k].get_older();
                  //if old enough or low util, agent dies
                  if( agent[k].get_resource1() < 0 ||
agent[k].get_resource2() < 0 || agent[k].get_age() >
agent[k].get_max_age()) {

      space[agent[k].get_y_coordinate()][agent[k].get_x_coordinate()].set_v
acant(); // old spot is now vacant

      space[agent[k].get_y_coordinate()][agent[k].get_x_coordinate()].set_a
gent_number(99999); // the agent number of the old spot is set to 99999
                        agent[k] = new Agent(); // new values are
initialized
                        agent[k].get_spot(space,k);  // agent is assigned a
new spot and the method sets it occupied
```

```
                        }
                } // end the death rule
                // compute some statistics
                stat1.compute_average_util_gained(agent);
                stat1.compute_average_age(agent);
                stat1.age_distribution(agent);
                stat1.compute_average_turn_price();
                stat1.compute_east_west_counter(agent, space);
                stat1.compute_west_east_ratio();
        } // end method gameLoop
}       // end class Sim1


import javax.swing.*;
import java.awt.geom.*;
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.net.*;
import java.io.*;


public class Sim2 extends JApplet implements Runnable {

        static int population_size = 100;
        static int number_of_turns = 2500;
        static int deathcounter = 0;
        static int movecounter = 0;
        static int number_of_exps = 10;
        static final int arraysize = 70;
        static boolean newround = false;
        static boolean river = true;
        static boolean trading = true;
        static boolean repeat_trade = true;
        static int river_latitude = 31;
        static final String newline = System.getProperty("line.separator");
        BufferedWriter out;

        int turn=0;
        int experiment =1;
        Thread runner;
        Grid[][] space1;
        Agent[] agent1;
        Grid[] next_to_river;
        statistics stat1;



        public void init() {
                // this sets the size of the applet
                setSize(700, 800);
                space1 = Grid.startupG(arraysize, river_latitude,river);
                agent1 = Agent.startupA(space1, population_size);
                next_to_river =
Grid.mapping_next_to_river(space1,river_latitude);
                stat1 = new statistics();
                try {
                        out = new BufferedWriter(new
FileWriter("run_nr_"+experiment+"_river_"+river+"_trade_"+trading+"_populat
ion_size"+population_size+".txt"));
```

```
                        out.write("Run number: "+experiment+",turn,average
utility, average age, total age, trade vol, river vol, turn trades, river
trades, average price, w_e_ratio, west count, east count, age dis" );
                        out.close();
                } catch (IOException e) {
                }
        }
        public void paint(Graphics g) {
                Graphics2D g2 = (Graphics2D)g;
                g2.drawString("Afriscape", 500, 20);

        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,RenderingHints.VA
LUE_ANTIALIAS_ON);
                // g2.setBackground(Color.RED);
                for (int i=0;i<arraysize;i++) {
                        for (int j=0;j<arraysize;j++) {
                                if (space1[i][j].get_land_status() == false) {
                                        g2.setPaint(Color.blue);
                                        g2.fill(new Rectangle2D.Double(6 *
space1[i][j].get_x_coordinate(), 6 * space1[i][j].get_y_coordinate(), 4,
4));
                                }
                                else {
                                        g2.setPaint(Color.white);
                                        g2.fillRect(6 *
space1[i][j].get_x_coordinate(), 6 * space1[i][j].get_y_coordinate(), 4,
4);
                                }
                        }
                }
                /*
            // this loop colors the agents according to how old they are
                for (int i=0; i<population_size;i++) {
                        if (agent1[i].get_age() <= 1) {
                                g2.setPaint(Color.CYAN);
                                g2.fillRect(6 * agent1[i].get_x_coordinate(), 6 *
agent1[i].get_y_coordinate(), 4, 4);
                        }
                        else if (agent1[i].get_age() > 1 && agent1[i].get_age()
<= 5) {
                                g2.setPaint(Color.MAGENTA);
                                g2.fillRect(6 * agent1[i].get_x_coordinate(), 6 *
agent1[i].get_y_coordinate(), 4, 4);
                        }
                        else if (agent1[i].get_age() > 5 && agent1[i].get_age()
<= 15) {
                                g2.setPaint(Color(255,0,255));
                                g2.fillRect(6 * agent1[i].get_x_coordinate(), 6 *
agent1[i].get_y_coordinate(), 4, 4);
                        }
                        else if (agent1[i].get_age() > 15 && agent1[i].get_age()
<= 30) {
                                g2.setPaint(Color.GRAY);
                                g2.fillRect(6 * agent1[i].get_x_coordinate(), 6 *
agent1[i].get_y_coordinate(), 4, 4);
                        }
                        else {
                                g2.setPaint(Color.BLACK);
                                g2.fillRect(6 * agent1[i].get_x_coordinate(), 6 *
agent1[i].get_y_coordinate(), 4, 4);
                        }
            }
             */
            // this loop colors the agents according to how many trades 
they make
            for (int i=0; i<population_size;i++) {
                  if (agent1[i].get_trades() <= 1) {
                        g2.setPaint(Color.yellow);
                        g2.fillRect(6 * agent1[i].get_x_coordinate(), 6 * 
agent1[i].get_y_coordinate(), 4, 4);
                  }
                  else if (agent1[i].get_trades() > 1 && 
agent1[i].get_trades() <= 20) {
                        g2.setPaint(Color.lightGray);
                        g2.fillRect(6 * agent1[i].get_x_coordinate(), 6 * 
agent1[i].get_y_coordinate(), 4, 4);
                  }
                  else if (agent1[i].get_trades() > 20 && 
agent1[i].get_trades() <= 100) {
                        g2.setPaint(Color.GRAY);
                        g2.fillRect(6 * agent1[i].get_x_coordinate(), 6 * 
agent1[i].get_y_coordinate(), 4, 4);
                  }
                  else if (agent1[i].get_trades() > 100 && 
agent1[i].get_trades() <= 500) {
                        g2.setPaint(Color.DARK_GRAY);
                        g2.fillRect(6 * agent1[i].get_x_coordinate(), 6 * 
agent1[i].get_y_coordinate(), 4, 4);
                  }
                  else {
                        g2.setPaint(Color.BLACK);
                        g2.fillRect(6 * agent1[i].get_x_coordinate(), 6 * 
agent1[i].get_y_coordinate(), 4, 4);
                  }

            }
            if (experiment == 10) {
                  g2.drawString("You are at exp. 10!!", 500, 60);
            }
            if (experiment == 20) {
                  g2.drawString("You are at exp. 20 !!", 500, 80);
            }
            if (experiment == 40) {
                  g2.drawString("You are at exp. 40!!", 500, 100);
            }
            if (experiment == 100) {
                  g2.drawString("You are at exp.!!", 500, 120);
            }
      }
      private Paint Color(int i, int j, int k) {
            // TODO Auto-generated method stub
            return null;
      }
      public void start() {
            if (runner == null) {
                  runner = new Thread(this);
                  runner.start();
            }
      }
      public void run() {
            Thread thisThread = Thread.currentThread();
            while (runner == thisThread && turn<number_of_turns &&
experiment<number_of_exps+1) {
                  turn++;
                  Sim1.gameLoop(agent1, space1,
population_size,arraysize,river,river_latitude,next_to_river,stat1,trading,
repeat_trade);
                  try {
                        out = new BufferedWriter(new
FileWriter("run_nr_"+experiment+"_river_"+river+"_trade_"+trading+"_populat
ion_size"+population_size+".txt", true));

      out.write(newline+","+turn+","+stat1.get_average_util_gained()+","+st
at1.get_average_age()+","+stat1.get_total_age()+","+stat1.get_per_round_tra
de_volume()+","+stat1.get_river_trade_vol()+","+stat1.get_turn_trades()+","
+stat1.get_turn_trades_on_river()+","+stat1.get_average_turn_price()+","+st
at1.get_west_east_ratio()+","+stat1.get_west_east_counters()+","+stat1.get_
age_distribution()+","+agent1[4].get_trades()+","+agent1[4].get_metabolism(
)+","+agent1[4].get_resource1()+","+agent1[4].get_resource2()+","+agent1[4]
.get_age()+","+agent1[4].get_util_gained_from_eating());

                        out.close();
                  }
                  catch (IOException e) {
                  }

                  /*
                  repaint();
            try {
                Thread.sleep(700);
            } catch (InterruptedException e) {
                // do nothing
            }
                   */

            } // end while
            // System.out.println("Completed run nr.: " +experiment);
            turn=0;
            repaint();
            experiment++;
            if (experiment <number_of_exps+1) {

                  try {
                        out = new BufferedWriter(new
FileWriter("run_nr_"+experiment+"_river_"+river+"_trade_"+trading+"_populat
ion_size"+population_size+".txt"));
                        out.write("Run number: "+experiment+",turn,average
utility, average age, total age, trade vol, river vol, turn trades, river
trades,average price, w_e_ratio, west count, east count, age dis" );
                        out.close();
                  }
                  catch (IOException e) {
                  }

                  space1 = null;
                  agent1 = null;
                  next_to_river =null;
                  runner = null;
                  space1 = Grid.startupG(arraysize, river_latitude,river);
                  agent1 = Agent.startupA(space1, population_size);
                  next_to_river =
Grid.mapping_next_to_river(space1,river_latitude);
```

```
				stat1 = new statistics();
				runner = new Thread(this);
				runner.start();
				try {
					repaint();
				}
				catch (NullPointerException e) {
					// do nothing
				}
			} // end if
		}
		public void stop() {
			if (runner != null) {
				runner = null;
			}
		}
		public void actionPerformed(ActionEvent event) {
			if (runner != null) {
				runner = null;
			}
		}
}// end class




// the sole purpose of this class is to create an object that keeps track
of different statistics about the population


public class statistics {

		private double sum_total_age;
		private double average_age;
		private double average_util_gained;
		private double per_round_trade_volume;
		private int turn_trades_on_river;
		private int turn_trades;
		private double total_turn_price;
		private double average_turn_price;
		private double river_trade_vol;
		private int i0;
		private int i5;
		private int i10;
		private int i15;
		private int i20;
		private int i25;
		private int i30;
		private int i35;
		private int i40;
		private int i45;
		private int i50;
		private int i55;
		private double east_counter;
		private double west_counter;
		private double west_east_ratio;

		// this is the constructor class(
		public statistics() {
			sum_total_age=0;
```

```
            average_age=0;
            average_util_gained=0;
            per_round_trade_volume=0;  // in terms of good
            turn_trades_on_river=0;
            turn_trades =0;
            river_trade_vol=0;
            average_turn_price=0;
      }

      public void compute_average_age(Agent[] agent) {
            double total_age = 0;
            for (int i =0;i<agent.length;i++) {
                  total_age = total_age+ agent[i].get_age();
            }
            sum_total_age = total_age;
            average_age = total_age/agent.length;
      }
      public void compute_average_util_gained(Agent[] agent) {
            double total_util_gained = 0;
            for (int i =0;i<agent.length;i++) {
                  total_util_gained = total_util_gained +
agent[i].get_total_util_this_round();
            }
            average_util_gained = total_util_gained/agent.length;
      }
      public void compute_east_west_counter(Agent[] agent, Grid[][] space)
{
            east_counter=0;
            west_counter=0;
            for(int i =0;i<agent.length;i++) {
                  if
(space[agent[i].get_y_coordinate()][agent[i].get_x_coordinate()+1].get_land
_status() == false) {
                        east_counter++;
                  }
                  else if
(space[agent[i].get_y_coordinate()][agent[i].get_x_coordinate()+2].get_land
_status() == false) {
                        east_counter++;
                  }
                  else if
(space[agent[i].get_y_coordinate()][agent[i].get_x_coordinate()+3].get_land
_status() == false) {
                        east_counter++;
                  }
                  else if
(space[agent[i].get_y_coordinate()][agent[i].get_x_coordinate()-
1].get_land_status() == false) {
                        west_counter++;
                  }
                  else if
(space[agent[i].get_y_coordinate()][agent[i].get_x_coordinate()-
2].get_land_status() == false) {
                        west_counter++;
                  }
                  else if
(space[agent[i].get_y_coordinate()][agent[i].get_x_coordinate()-
3].get_land_status() == false) {
                        west_counter++;
                  }
            }
```

```
}
public void compute_west_east_ratio() {
      west_east_ratio = west_counter/east_counter;
}
public double get_west_east_ratio() {
      return west_east_ratio;
}

public void set_per_round_trade_volume(double input) {
      per_round_trade_volume= input;
}
public void set_trades_on_river(int input) {
      turn_trades_on_river = input;
}
public int get_turn_trades_on_river() {
      return turn_trades_on_river;
}
public int get_turn_trades() {
      return turn_trades;
}
public void set_turn_trades(int input) {
      turn_trades = input;
}
public void set_river_trade_vol(double input) {
      river_trade_vol = input;
}
public double get_river_trade_vol() {
      return river_trade_vol;
}
public double get_per_round_trade_volume() {
      return per_round_trade_volume;
}
public double get_average_age() {
      return average_age;
}
public double get_average_turn_price() {
      return average_turn_price;
}
public double get_average_util_gained() {
      return average_util_gained;
}
public double get_total_age(){
      return sum_total_age;
}
public double get_total_turn_price() {
      return total_turn_price;
}
public void set_per_round_total_price(double input) {
      total_turn_price = input;
}

public void compute_average_turn_price() {
      average_turn_price = total_turn_price / turn_trades ;
}
public void set_average_age(double inc_age) {
      average_age = inc_age;
}
public void set_average_util_gained(double inc_util) {
      average_util_gained = inc_util;
}
```

```
	public void age_distribution(Agent[] agent) {
		i0 =0;
		i5 =0;
		i10=0;
		i15=0;
		i20=0;
		i25=0;
		i30=0;
		i35=0;
		i40=0;
		i45=0;
		i50=0;
		i55=0;
		for (int i =0;i<agent.length;i++) {
			if (agent[i].get_age() <= 4) {
				i0++;
			}
			else if (agent[i].get_age() > 4 && agent[i].get_age() <=
9) {
				i5++;
			}
			else if (agent[i].get_age() > 9 && agent[i].get_age() <=
14) {
				i10++;
			}
			else if (agent[i].get_age() > 14 && agent[i].get_age() <=
19) {
				i15++;
			}
			else if (agent[i].get_age() > 19 && agent[i].get_age() <=
24) {
				i20++;
			}
			else if (agent[i].get_age() > 24 && agent[i].get_age() <=
29) {
				i25++;
			}
			else if (agent[i].get_age() > 29 && agent[i].get_age() <=
34) {
				i30++;
			}
			else if (agent[i].get_age() > 34 && agent[i].get_age() <=
39) {
				i35++;
			}
			else if (agent[i].get_age() > 39 && agent[i].get_age() <=
44) {
				i40++;
			}
			else if (agent[i].get_age() > 44 && agent[i].get_age() <=
49) {
				i45++;
			}
			else if (agent[i].get_age() > 49 && agent[i].get_age() <=
54) {
				i50++;
			}
			else {
				i55++;
			}
```

```
            }
        }
        public String get_age_distribution() {
            String age_distribute;
            age_distribute =
i0+","+i5+","+i10+","+i15+","+i20+","+i25+","+i30+","+i35+","+i40+","+i45+"
,"+i50+","+i55;
            return age_distribute;
        }
        public String get_west_east_counters() {
            String counters = west_counter+","+east_counter;
            return counters;
        }

}
```