AUA Working Paper Series No. 2015-3
October 2015

Dealing with farm heterogeneity on modeling agricultural policy: An Agent Based Modeling Approach

Dimitris Kremmydas[†]
Department of Agricultural Economics and Rural Development, Agricultural University of Athens
kremmydas@aua.gr

Stelios Rozakis
Department of Agricultural Economics and Rural Development, Agricultural University of Athens
rozakis@aua.gr

Ioannis Athanasiadis
Electrical and Computer Engineering Dept. of Democritus University of Thrace, in Xanthi, Greece
ioannis@athanasiadis.info

[†] Corresponding Author

A preliminary version of this paper was presented in the 20th Annual Workshop on the Economic Science with Heterogeneous Interacting Agents (WEHIA), in Sophia Antipolis, in the Côte d'Azur region of France, on May 2015

Agricultural University of Athens ·
Department of Agricultural Economics
& Rural Development · http://www.aoa.aua.gr

Dealing with farm heterogeneity on modeling agricultural policy: An Agent Based Modeling Approach

Dimitris Kremmydas[1†], Stelios Rozakis[1], Ioannis Athanasiadis[2]

[1] Department of Agricultural Economics and Rural Development, Agricultural University of Athens
[2] Electrical and Computer Engineering Dept, Democritus University of Thrace
[†] Corresponding author

Abstract

The Common Agricultural Policy (CAP) has been gradually transformed from directly supporting prices and production to a decoupled scheme, where farmers receive a payment per hectare regardless of their production decisions. Within this framework and given the multitude of CAP's objectives, ranging from market competiveness to multifunctionality of agriculture, the inclusion of the heterogeneity of farms on modeling CAP will continuously arise in the immediate future as a key research question. In this paper we make a brief discussion of the aspects of farm heterogeneity within the agricultural policy modeling context and we show that Agent Based Modeling approach, coupled with Object Oriented System Analysis, is a very good alternative for considering all aspects of diversity of farms. Finally we present Agroscape, a flexible ABM Agricultural Policy Framework that can easily incorporates both behavioral and capacity heterogeneity presenting a proof-of-concept case study.

Keywords: Agricultural Policy, Agent Based Modeling, Object Oriented System analysis, Farm heterogeneity
JEL Codes: Q12, Q18, C63

1 Introduction

“Heterogeneity” (from Greek ετερογένεια / heterogeneia, i.e. of different family/gene) is defined as “the quality of being consisted of dissimilar or diverse ingredients or constituents”. The term can refer to two things: (a) the fact that farms, within a specific geographical area, differ from each other in almost all of their aspects (local heterogeneity) and (b) that between two distant regions the distributions of farm characteristics are different (inter-regional heterogeneity). In this paper we focus on local heterogeneity.

The need for employing a discussion focusing on farm heterogeneity emerges from the last (European) Common Agricultural Policy (CAP) reform. More specifically, the CAP-2020 reform introduces a number of new objectives, such as advancing agricultural productivity, securing fair farmers’ income, even in less advantaged areas, maintaining food security through the promotion of the respect of certain standards, safeguarding the sustainable management of natural resources and the protection of the viability of rural economy.

The proposed means to achieve the above goals are also innovative. Indicatively, there are measures to facilitate collective investment; assist small farms to develop; foster knowledge transfer and innovation; enhance competitiveness; promote food chain organization & risk management; restore, preserve &

enhance ecosystem services; promote resource efficiency and transition to a low-carbon economy; promote social inclusion, poverty reduction and economic development in rural areas. All of those measures have a local and non-aggregate dimension and so the individual characteristics of farms should be taken into account by policy modelers.

Furthermore the new CAP departs from a status where strict rules were applied for all member states, introducing implementation flexibility. The member states are equipped with a toolbox of measures that can fine tune in order to cater their specific needs. Consequently, national policy makers are required to make more focused decisions. In this manner considering farm heterogeneity is applicable in the design of the CAP policy.

In this paper we discuss some aspects of farm heterogeneity within the agricultural policy modeling context arguing that Agent Based Modeling (ABM) approach is a very good alternative for dealing with it. Finally we present an ABM framework, capable of incorporating the various forms of farm heterogeneity and provide some proof-of-concept results.

2 Theoretical Considerations

2.1 Aspects of farmer's heterogeneity

Given that the scope of the paper is focused on agricultural policy the discussion will be confined to aspects of farm heterogeneity that concern CAP. This is already a wide scope, because as mentioned above, this policy is concerned with many facets of an agricultural system (technical, economic, environmental, and social).

There is an evident first layer of farm heterogeneity which could be termed as "**endowment heterogeneity"**. Farms own land of different soil quality and have different initial land and labor endowments. This kind of heterogeneity is most often included in farm models mainly by differentiating farm yields and variable costs.

A second layer could be the **"managerial ability heterogeneity"** referring to both technical and economic efficiency of the farm management. In this layer only operational (short-term) and tactical (mid-term) managerial decisions are included. As usually observed in farm efficiency analysis case studies, there is a significant variation between farms in managerial ability (Nuthall, 2001) and it is of interest to policy makers since it is an essential factor in successfully explaining varying agricultural output and supply relationships. There exist several studies providing explanations of this variation. As Nuthall (2009) argues, managerial ability is related to education, training and intelligence, age and experience and also "to social capital which involves the networks a manager may have, as well as the relevant components of the current culture".

A third layer of heterogeneity is about the strategic orientation of farms, i.e. theirs long-term goals and how the take decisions, which could be termed as **"decision making heterogeneity"**. In the literature it is widely recognized that not all farms are profit maximizers. Apart from the "satisficing" principle and the notion of bounded rationality that Simon (1957) introduced, there is the case of the multiplicity of goals –potentially modeled in a multi-objective problem, for example in Karanikolas et al. (2013). There is evidence that the age of the farm manager is a significant factor shaping his behavior.

2.2 Including farmer's heterogeneity in Agricultural Policy models

Farm heterogeneity could be included in the agricultural policy discussion in two modes. Firstly as a factor that differentiates the impact of a policy from one farm to another, or from one group of farms to another, and thus is used in the result analysis stage in order to get a more detailed view of the final state of the system (result analysis mode).

Secondly as a factor that has an endogenous effect on the results of the policy, and thus is included directly into the model (modeling mode). An example of the latter can be found at Liu et al. (2007): “the socioeconomic differences among people in a relevant case study lead to different choices and behaviors, which in turn result in very different ecological outcomes than one would find were everyone to have the same preferences for ecosystem services”. Another case of heterogeneity “modeling mode” would be in the case of Agricultural Value Chains (Nolan, 2009). As noted, “modern agricultural systems (production, distribution, marketing) are in a state of transition, with increasingly numerous and heterogeneous agents interacting in the value chain”. Comprehension of the function of this value chain should include the endogenous modeling of the heterogeneity of the relevant agents.

For short to mid-term models it is reasonable to assume that ignoring endogenous effects of heterogeneity will be a good approximation of reality. On the other hand, for long term modeling, like human – nature interaction is, heterogeneity becomes a crucial element of the modeling process.

Nevertheless, representing heterogeneity in policy models can have some intrinsic limitations. As Heckelei (2013) notes, current agricultural databases have a limited level of detail. Thus heterogeneity ends, at the good scenario, being inferred through statistical methods . Also the spatial heterogeneity should be specifically relevant for a particular policy impact otherwise the benefits are questionable. This argument apparently holds for any kind of heterogeneity.

3 The Agent Based Modeling approach

3.1 The Agroscape ABM Framework

3.1.1 Design Principles

The Agroscape ABM is a modeling framework with the aim of facilitating the modeling of a variety of agricultural production systems. Its design is based on the following principles:

(a) The agricultural production system, being a coupled human – nature system, is a complex adaptive system. It contains reciprocal and feedback loops exhibiting nonlinearities like thresholds (Liu et al., 2007). Since the agricultural policies are increasingly focusing on environmental goals, the modeling process should incorporate this complexity.

(b) A good approach for modeling complex systems is Object Oriented Analysis and Design (OOAD). Briefly, OOAD is about structurally and functionally decomposing a system into smaller units with less complexity and less responsibilities (Booch, 2007). The collaboration of those simpler components is considered to provide the functionality of the system-as-a-whole (Solms, 2014) and thus the question of modeling the complex system is transformed in the questions of modeling many smaller and simpler (non-complex) components and their interactions. More technically OOAD is about applying the principles of abstraction, encapsulation, modularity and hierarchy to the system under consideration. Although OOAD is very closely related to software design, the last two decades is applied to other disciplines as well.

(c) The Agent Based Modeling is an adequate approach for modeling complex adaptive systems using OOAD. It is evident that ABM fits very well with the concepts of Object Oriented Programming, representing agents as software objects and focusing on their interaction.

3.1.2 Main Modeling Elements

Agroscape is programmed in Repast Simphony[1], which is a Java ABM programming framework providing an ABM scheduling namespace and many visual enhancements. Since our framework is actually build upon the Repast Simphony Object model we present briefly its essential elements:

1. *The Context* interface is actually a *Collection* that holds simulation objects. *Contexts* can include other *contexts*, thus providing the ability to the modeller for creating hierarchies of collections of agents. *Contexts* support *Projection* and *Data Layers* classes. All objects in Repast start their life in a root *Context*.
2. *The Projection* interface is a collection of relations between simulation objects. Spatial or network relationships are represented by corresponding projections. A *Projection* is always attached to a specific *context*, imposing a structure upon the contained agents. Apart from Continuous and grid space implementations there are also GIS and Network implementations of this interface
3. *The Data Layer* interface allow the efficient handling of the interaction between agents and data. A *Data Layer* can be either an abstract matrix attached to a *context* or a matrix attached to a Grid Projection thus storing one value for each grid's cell (*GridValueLayer*).

One can see in detail the class hierarchy of Repast in its API documentation[2].

Regarding Agroscape, the skeleton of the framework is shown in the UML 2.0 class diagram on Figure 1. The two core classes are *Farmer* and *Space*, the former being a POJO (Plain Old Java Object) and the latter a *Grid Projection*, i.e. a pixeled surface where all activity is taking place. The simulation can also contain many *PropertyGridValueLayer* (a *Data Layer* class) objects*,* i.e. spatial properties (e.g. soil quality, crop suitability, nitrates concentration etc.). *Farmer* is related to *Space* indirectly through *Plot*, which is a logical grouping of space and this is realized through a *LandRegistryAuthority* class that is responsible for the bookkeeping of the ownership of the Plots. The actual ownership relation is between a *HumanAgent* and a *Plot*, since it is expected that also non-Farmers might own land. *Farmer* is also related to *Space* by the *residentIn* association. Additionally, all *Farmer* objects exist within a *FarmerContext* object. The latter can contain an arbitrary number of *Network* objects, representing various kinds of relationships between farmers, for example a social network, an information exchange network, etc. The activity of the agents is realized through attached behaviors, contained in *FarmerContext* and explained in more details right after. The *behaviors* scheme provides modeling extensibility, since new behaviors are easy to be added.

[1] http://repast.sourceforge.net/repast_simphony.php
[2] http://repast.sourceforge.net/docs/api/repast_simphony/

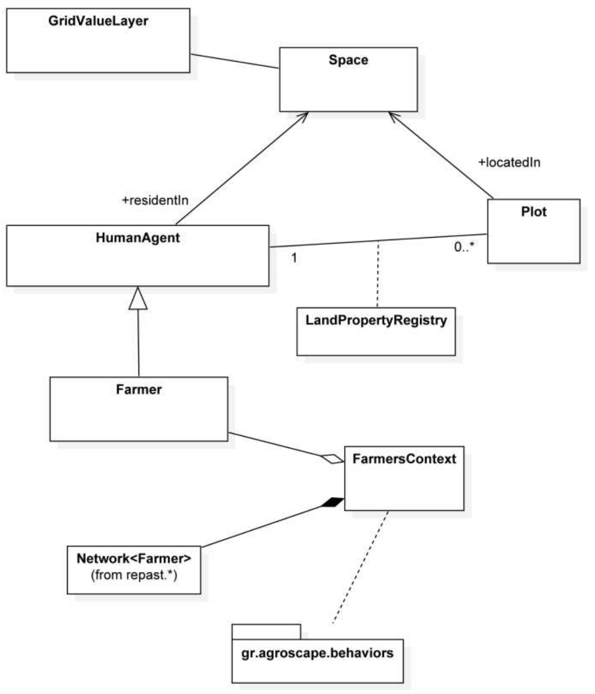


Figure 1, UML class diagram of the Agroscape skeleton

3.1.3 The Behavior Package

The aspired flexibility of Agroscape is founded on the idea of the “behavior package”. The agents that are defined in the skeleton of the framework are idle by default. In order to act they need to be attached to one or more behavior classes. A behavior class is an implementation of an action, like taking production decisions, realizing production, making transactions in a land market, deciding for the adoption of a new technology, etc. It is an OOAD way of programming agents in the simulation’s timeline.

If ones tries to implement many kind of behaviors for a specific agent with the conventional way of hard-coding them directly into its namespace, the maintenance of the code becomes cumbersome and very possibly conflicts appear between them. On the other hand, the “behavior package” is an innovative and pluggable approach to implement and addi new behaviors to various agents of the simulation is easy and also keeps the modeling complexity in manageable levels.

That is because the behaviors are independent of each other and use only the core classes of the simulation, as described above, without being affected by their properties. The modeler of a certain behavior is also absolutely responsible for scheduling its operations and implementing the logic, without being affected by other already implemented behaviors. Furthermore one behavior that is attached to an agent can also use objects from another behavior of the same agent, since they are all connected to the same agent object. Finally modelers have the possibility to attach different set of behaviors to different agents.

In the implementation side, the idea is based on *IScheduledBehavior* interface which defines a single *getAnnotatedClass* method, returning an object containing *ScheduledMethodAnnotation* annotations. This annotation class defines several properties of a method's scheduling, like the start and interval ticks of the simulation clock. In this way the modeler have full flexibility on the timing of behaviors. What is left is to model the behavior of the agent.

In order to do so, he has to extend the abstract *AFarmerBehavior* class that contains a reference to the *Farmer* object that exhibits the behavior and also implements *IScheduledBehavior*. Also, since frequently all agents that are attached to a specific behavior will need to either access common classes, or communicate with each other, all behaving objects are contained in an extension of a *ABehaviorContext*. This context also contains a *IscheduledBehaviorDataLoader* object that is responsible for loading all the behaving objects into the context and also adding their behavior to the simulation's schedule.

A simple illustrative example of a behavior is given in Appendix.

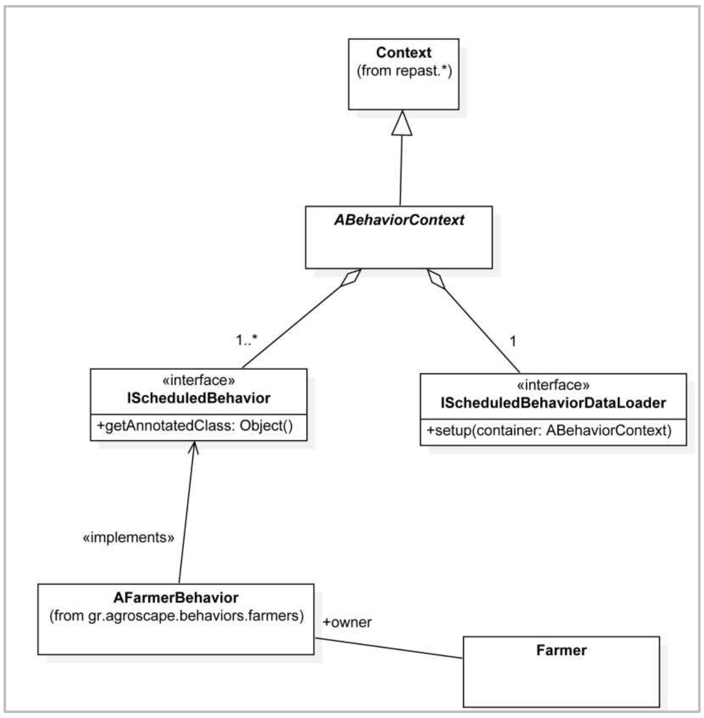

Context
(from repast.*)
ABehaviorContext
1..*
1
«interface»
IScheduledBehavior
+getAnnotatedClass: Object()
«interface»
IScheduledBehaviorDataLoader
+setup(container: ABehaviorContext)
«implements»
AFarmerBehavior
(from gr.agroscape.behaviors.farmers)
+owner
Farmer

Figure 2, Class diagramm of Behavior package

3.2 Catering for Farm Heterogeneity with the Agroscape framework

Our approach for representing heterogeneity deviates from searching for a suitable typology, where a classification according to certain macro-indicators is performed and certain farm types are derived, e.g. Amico et al. (2013). Rather than performing a statistical analysis of the observed outputs we facilitate the analysis of the production system in terms of system components and theirs relations and embed the heterogeneity in agents' state, actions and interactions.

Following the "behavior package" approach, the various forms of heterogeneity is not modeled directly into a Farmer object but rather is embodied in the individual behavior classes. This approach, although does not seem very natural, is not limiting at all, because any behavior class can exchange information with the skeleton classes of the model.

More specifically, as far as land endowment heterogeneity is regarded, any behavior can have access to the plots that a farmer use and so has access to spatial diversity through the *PropertyGridValueLayer* class. For capital and labor endowment heterogeneity, a behavior can introduce attributes that express this fact, as shown in the case study.

As far as “managerial ability heterogeneity” is concerned, one approach would be to model managerial ability as a [0,1] coefficient that is multiplied with expected yield to give the actual yield, differentiating efficient farmers. Another approach would be to make a low-level modeling of the technical or economic decisions that a farmer is following, embedding managerial details. A relevant modeling of the production realization is shown in Daydé et al. (2014). Although that approach seems to be much more complex, the Agroscape framework could easily facilitate it, showing that the OOAD benefits.

Also decision making heterogeneity is very easily modeled within the proposed framework. Farmers can easily be attached to different specific decision making behaviors interacting with other simulation elements very easily. Furthermore already implemented decision making models can be incorporated to an Agroscape model, utilizing the flexibility and the power of the Java programming language. For instance, since there are many farm models written in GAMS mathematical programming software, a special adapter class could be crafted to use the already written code for a production behavior.

In order to illustrate the above arguments, a simple proof-of-concept case study has been implemented, where farmers own land of different crop suitability and exhibit varying behavior. One can first examine the appendix for an even simpler “hello-world” example.

3.3 A proof-of-concept case study: The *arableCropProduction* Behavior

Arable Crop yearly decisions have been extensively used in agricultural policy modeling over the last decades. An elementary model can be represented as a linear programming problem where an individual farmer f is supposed to choose a cropping plan $\vec{x}^f$ and input use among technically feasible activity plans $A^f \cdot \vec{x}^f \leq \vec{b}^f$ so as to maximize gross margin gm^f. The optimization problem for the farmer f can be expressed as follows (Kremmydas et al., 2012):

$$\begin{cases} \max\limits_{x^f} \ gm^f\left(\mathbf{x}^f, \boldsymbol{\theta}^f, \boldsymbol{\kappa}\right) \equiv \sum\limits_{i=1}^{n} \left\{ \left[\left(p_i \mid p_i^f + ps_i \right) y_i^f + ls_i - c_i^f \right] x_i^f \right\} \\ \text{s.t.} \quad \mathbf{A}^f\left(\boldsymbol{\theta}^f\right)\mathbf{x}^f \leq \mathbf{b}^f\left(\boldsymbol{\theta}^f\right) \qquad \mathbf{A} \in \mathbb{R}^{m\times n} \\ \qquad\qquad \mathbf{x}^f \geq 0 \qquad\qquad \mathbf{x} \in \mathbb{R}^{n} \end{cases} \tag{1}$$

Where

The $m \times n$ matrix A^f and the $m \times 1$ vector $\vec{b}^f$ represent respectively the technical coefficients and the capacities of the m constraints on production. The vector of parameters θ^f includes yields for crop i (y_i^f), variable costs (c_i^f), prices dependent on quality (p_i^f) and subsidies linked to crop quantity (ls_i^f).

Symbol stands for the vector of general economic parameters which includes prices not dependent on farm (p_i) and subsidies specific to crop cultivated area (ps_i).

Below we describe the transformation of such an elementary model to an agent based model, following the "Behavior package" approach.

3.3.1 The ArableCropProductionBhv

Following the OOAD principles we decompose the above linear programming problem domain to its constituent elements, representing each as a different class. We also provide additional elements in order to represent the spatial dimension, not currently available to the above formulation, and we finally introduce classes for different decision making strategies (linear programming being one member of the set of strategies). All of the derived classes are related to the skeleton classes of the Agroscape framework that has already been presented in 3.1.2.

Thus the derived classes of the domain are:

1. *ArableCropCultivation* represents the various crop cultivations that are available, e.g. maize, durum wheat, barley, etc.
2. *ArableCropProductionDecision* is a map from a *Plot* (see 3.1.2) to an *ArableCropCultivation*, denoting the fact that the farmer's decision is actually the assignment of an arable crop to each of the owned plots.

   ExpectedCropPrices, *ExpectedPlotCropVarCost* and *ExpectedPlotCropYield* are the corresponding elements of the farmer's objective function. So a farmer has certain expectations about the next year's prices of a crop output (*ExpectedCropPrices*) and he has an expectation regarding the variable cost and the yield of the "Crop x Plot" combinations. In the current paper exercise we implemented the formation of those expectations to be really simple (taken from pre-defined values plus/minus a random number). In the future we could implement a more realistic but complex modeling of how those expectations are formed (e.g. prospect theory, evolutionary algorithms, through networking with other farmers, etc.) and here the power of the OOAD approach emerges: Even if we would insert into the model such complex procedures, we need only to change the respective *Expectation* class while the rest of the model would left intact. That is because we first tackled the complexity of the relationships between the various classes of the domain, modeling the points of contact between them, and encapsulated the complexity of the classes into themselves. In this way OOAD releases the modeling process from the burden of dealing with the complexity of each element of the system at the same time.

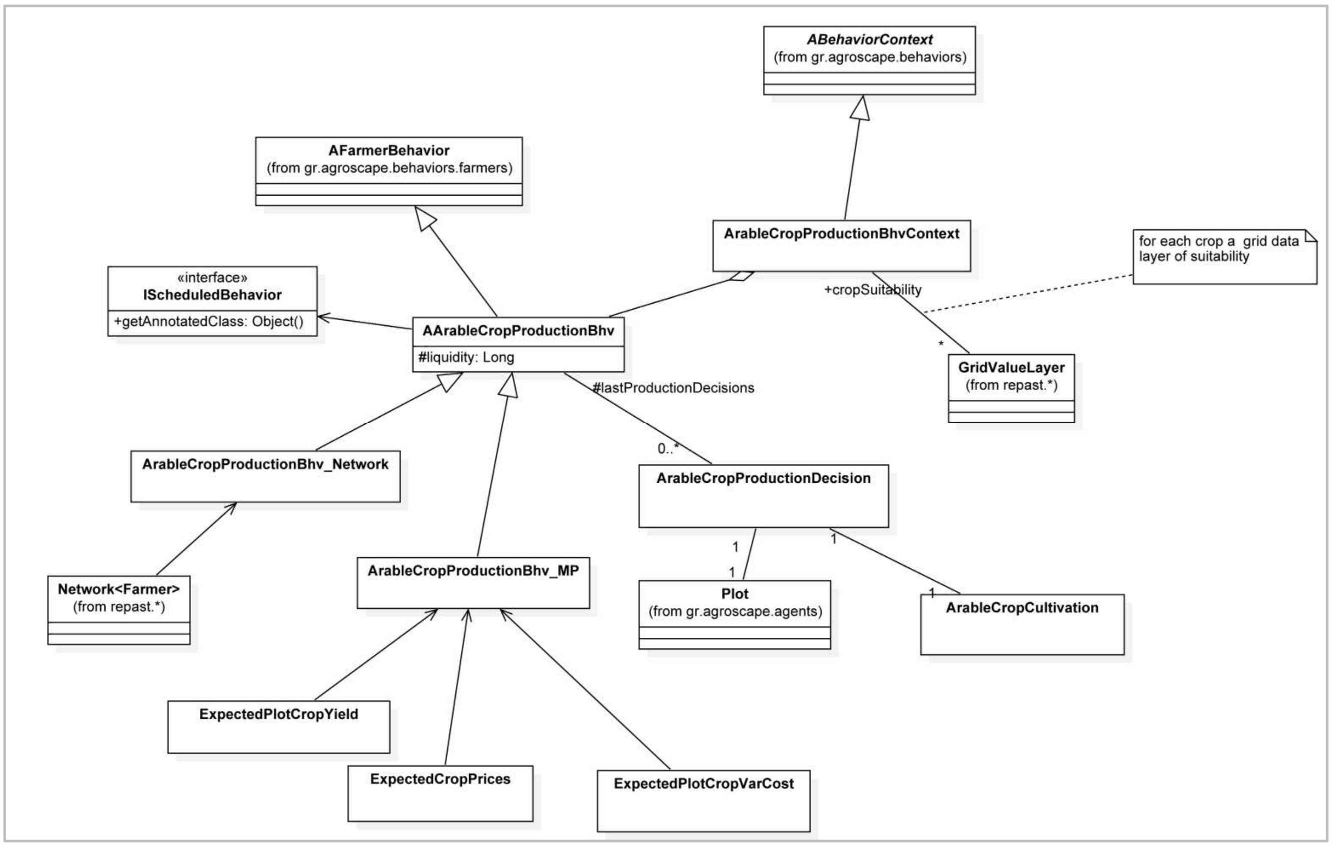


Figure 3, An overview of the arableCropProductionBehavior

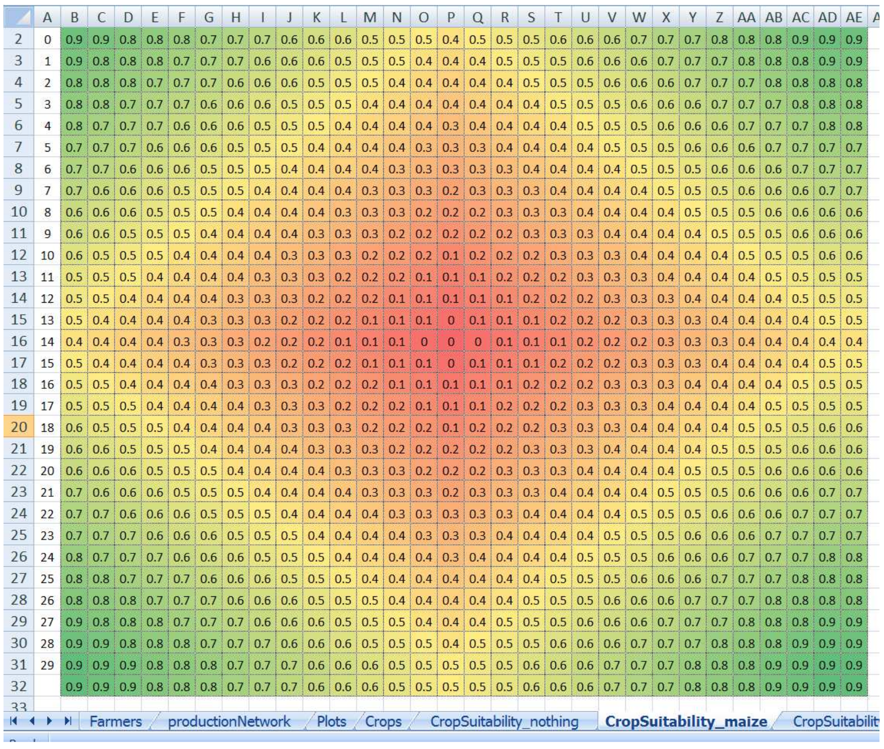
3.3.2 Data and Results

The data management issue on modeling agricultural policy cannot be overlooked. The transformation and loading of data for such models is usually cumbersome, especially if ones goes to plot-level detail or include spatial data, since modeling software (like GAMS) does not provide explicit data handling mechanisms.

Although the data used in this exercise was fictitious, we followed the OOAD approach in the data management aspect, unbinding the mechanics of the model with the data loading process and hopefully giving more flexibility to potential modelers.

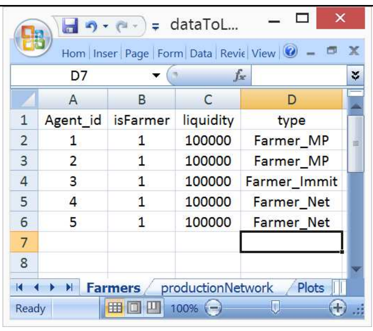
In order for loading data into the Agroscape framework, the *IScheduledBehaviorDataLoader<T>* shall be implemented. This interface defines the *setup(ABehaviorContext<T> container)* method that is called for each behavior loaded during the initialization of the simulation and the *ABehaviorContext<T>* top context is passed. The implementation should be done so as all *AArableCropProductionBhv<T>* objects are loaded in the container.

In our example case we kept all data in an excel sheet, as shown in figures 4 - 7. In order to load the data we created an *ExcelDataLoader* class implementing the required interface. In Figure 8 we show the essential part of the code. The class creation takes the excel workbook and the setup method, that is called from the simulatin initation procedure provides the container that the data loader class loads *AArableCropProductionBhv* objects with private and unexposed functions. If one wants to load data through a different excel structure, then he has to change the internal functionality of *ExcelDataLoader* (or write a new implementation) without being concerned about the stability of the overall model.



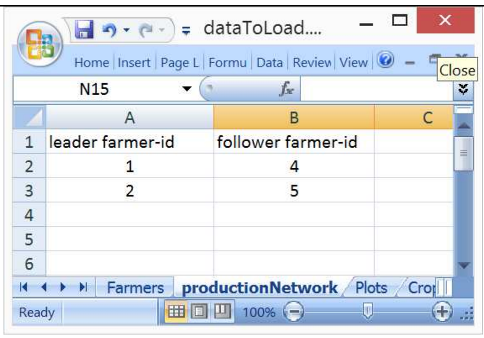
	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC	AD	AE
2	0	0.9	0.9	0.8	0.8	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8	0.8	0.9	0.9	0.9
3	1	0.9	0.8	0.8	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8	0.8	0.9	0.9
4	2	0.8	0.8	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8	0.8	0.8
5	3	0.8	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8	0.8
6	4	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8
7	5	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.7
8	6	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7
9	7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7
10	8	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.6
11	9	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6
12	10	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.1	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6
13	11	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.5
14	12	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5
15	13	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.1	0.1	0.1	0	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5
16	14	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.1	0.1	0.1	0	0	0	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.4
17	15	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.1	0.1	0.1	0	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5
18	16	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.1	0.1	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5
19	17	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.1	0.1	0.1	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.5
20	18	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.1	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6
21	19	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6
22	20	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.2	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.6
23	21	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.2	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7
24	22	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7
25	23	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.3	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.7
26	24	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.3	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8
27	25	0.8	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8	0.8
28	26	0.8	0.8	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8	0.8	0.8
29	27	0.9	0.8	0.8	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.4	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8	0.8	0.9	0.9
30	28	0.9	0.9	0.8	0.8	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.4	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8	0.8	0.9	0.9	0.9
31	29	0.9	0.9	0.9	0.8	0.8	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8	0.8	0.9	0.9	0.9	0.9
32		0.9	0.9	0.9	0.8	0.8	0.8	0.7	0.7	0.7	0.6	0.6	0.6	0.5	0.5	0.5	0.5	0.5	0.6	0.6	0.6	0.7	0.7	0.7	0.8	0.8	0.8	0.9	0.9	0.9	0.9
33																															

Figure 4, Excel Data for the Crop Suitability of maize



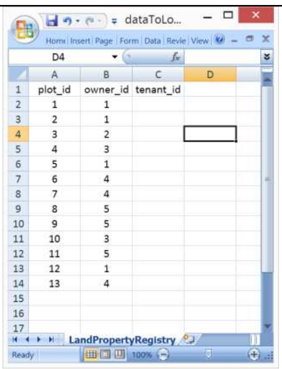
	A	B	C	D
1	Agent_id	isFarmer	liquidity	type
2	1	1	100000	Farmer_MP
3	2	1	100000	Farmer_MP
4	3	1	100000	Farmer_Immit
5	4	1	100000	Farmer_Net
6	5	1	100000	Farmer_Net
7				
8				

Figure 5, Data for assigning decision strategies to farmers



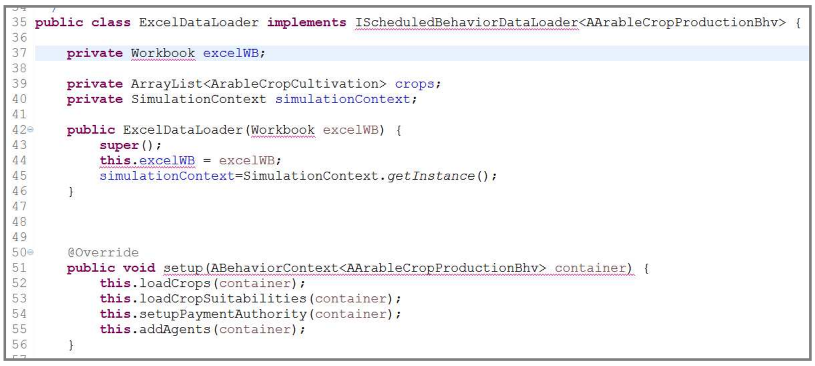
	A	B	C
1	leader farmer-id	follower farmer-id	
2	1	4	
3	2	5	
4			
5			
6			

Figure 6, Data for creating the network



	A	B	C	D
1	plot_id	owner_id	tenant_id	
2	1	1		
3	2	1		
4	3	2		
5	4	3		
6	5	1		
7	6	4		
8	7	4		
9	8	5		
10	9	5		
11	10	3		
12	11	5		
13	12	1		
14	13	4		
15				
16				
17				

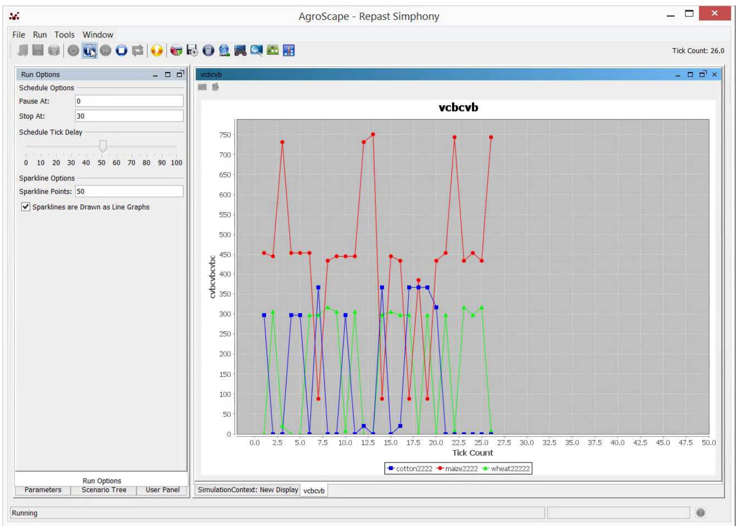
Figure 7, Data for Land Property Registry



```java
public class ExcelDataLoader implements IScheduledBehaviorDataLoader<AArableCropProductionBhv> {

    private Workbook excelWB;

    private ArrayList<ArableCropCultivation> crops;
    private SimulationContext simulationContext;

    public ExcelDataLoader(Workbook excelWB) {
        super();
        this.excelWB = excelWB;
        simulationContext=SimulationContext.getInstance();
    }



    @Override
    public void setup(ABehaviorContext<AArableCropProductionBhv> container) {
        this.loadCrops(container);
        this.loadCropSuitabilities(container);
        this.setupPaymentAuthority(container);
        this.addAgents(container);
    }
```

Figure 8, ExcelDataLoader essential part

After loading the data the simulation was run for a certain number of iterations, recording the decisions of the farmers. We recorded data through the relevant time-saving Repast Simphony mechanism. For example the allocation of the crop to plots was recorded in a video and a graph of the total land per crop was also easily configured to output. The vibration of the surface allocated to the crops is to the feedback mechanism of the deterioration and restoration of the cropSuitability feature of the model.



Tick Count: 26.0

Schedule Options
Pause At: 0
Stop At: 30

Sparkline Options
Sparkline Points: 50
Sparklines are Drawn as Line Graphs ✓

Running

Figure 9, The time series of total land per arable crop

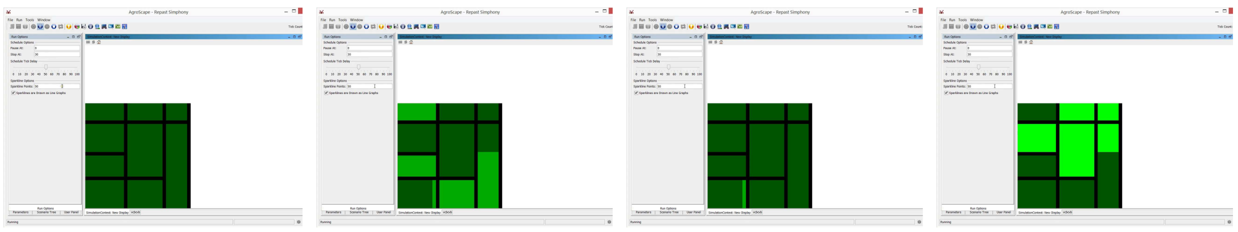


Figure 10, Crops to Plots allocation evolving through time

3.4 Conclusions and Future Research

In this paper we attempted, firstly to prove that agent based modeling should be considered as a well suited modeling approach for dealing with farm heterogeneity in agricultural policy modeling and secondly to propose an agent based modeling framework (Agroscape) that relies heavily on the OOAD principles

The advantages of this approach are:

- Endowment and Managerial farm heterogeneity can be represented as easily as in other approaches. Furthermore space is inherently represented in ABM simulation systems whereas this in not the case in general.
- Managerial heterogeneity can be modeled more efficiently compared to other approaches
- If the OOAD principles are followed, the managerial heterogeneity modeling can be carried out without the complexity "explosion" of the modeling process. The latter is present when one tries to build models that deal with many and different aspects of a system at the same time.

We implemented a proof-of-concept case study with just 5 farmers following two different decision making strategies for selecting an arable crop to cultivate in one of their owned plots (30X30 grid containing 13 plots).

Future work could include:

- Incorporate a Land market througth the behavior package mechanism
- Incorporate otherproduction decision behaviors, like Animal Husbandry production decisions (independently or jointly with arable crop decisions), permanent crop installation and handling decisions, etc
- Model other key players of the Agricultural value chain and investigate on the interaction with farmers (e.g. an information exchange network).
- Implement a real case policy evaluation case

4 References

Booch Grady, et al., 2007. Object-Oriented Analysis and Design with Applications (3nd Ed.).Pearson Education, Inc., Boston, MA, USA.

D. Kremmydas, A. Petsakos, and S. Rozakis. 2012. Parametric Optimization of Linear and Non-Linear Models via Parallel Computing to Enhance Web-Spatial DSS Interactivity. *Int. J. Decis Support Syst. Technol.* 4, 1 (January 2012), 14-29. DOI=http://dx.doi.org/10.4018/jdsst.2012010102

D'Amico Mario, et al., 2013. Agricultural systems in the European Union: an analysis of regional differences. New Medit, N. 4/2013.

Daydé, C., Couture, S., Garcia, F., Martin-Clouaire, R. (2014). . Investigating operational decision-making in agriculture. In: Proceedings of the 7th International Congress on Environmental Modelling and Software, June 15-19, San Diego, California, USA. Presented at 7th Congress on Environmental Modelling and Software (IEMSS 2014), San Diego, USA (2014-06-15 – 2014-06-19). Downloaded from http://prodinra.inra.fr/record/258496

Daydé, C., Couture, S., Garcia, F., Martin-Clouaire, R., (2014). . Investigating operational decision-making in agriculture. In: Proceedings of the 7th International Congress on Environmental Modelling and Software, June 15-19, San Diego, California, USA. Presented at 7th Congress on Environmental Modelling and Software (IEMSS 2014), San Diego, USA (2014-06-15 – 2014-06-19). Downloaded from http://prodinra.inra.fr/record/258496

European Commision, 2013. MEMO: The common agricultural policy (CAP) and agriculture in Europe – Frequently asked questions, Brussels 26 June 2013, accessed from http://europa.eu/rapid/press-release_MEMO-13-631_en.htm.

Heckelei Thomas, 2013. General methodological issues on farm level modelling. Chapter in Farm level modelling of CAP: a methodological overview, Ed. Langrell Stephen, JRC Scientific and Policy Reports,

Herbert A. Simon, 1957. Models of Man: Social and Rational. New York: John Wiley and Sons, Inc

Jianguo Liu , et al., 2007, Complexity of Coupled Human and Natural Systems, Science 317 (5844), 1513-1516. [DOI:10.1126/science.1144004]

Jianguo Liu , Thomas Dietz, Stephen R. Carpenter, Marina Alberti, Carl Folke , et al., 2007. Complexity of Coupled Human and Natural Systems, Science 14 September 2007: 317 (5844), 1513-1516. [DOI:10.1126/science.1144004]

Juliano J. Assunção, Maitreesh Ghatak, 2003. Can unobserved heterogeneity in farmer ability explain the inverse relationship between farm size and productivity, Economics Letters, Volume 80, Issue 2, August 2003, Pages 189-194, ISSN 0165-1765, http://dx.doi.org/10.1016/S0165-1765(03)00091-0.

McConnell, D. J. & Dillon, John L. & Food and Agriculture Organization of the United Nations., (1997). Farm management for Asia : a systems approach. Rome : Food and Agriculture Organization of the United Nations

Nolan, J., Parker, D., Van Kooten, G. C. and Berger, T., 2009. An Overview of Computational Modeling in Agricultural and Resource Economics. Canadian Journal of Agricultural Economics/Revue canadienne d'agroeconomie, 57: 417–429. doi: 10.1111/j.1744-7976.2009.01163.x

Nuthall, P., 2009. Modelling the origins of managerial ability in agricultural production. Australian Journal of Agricultural and Resource Economics, 53: 413–436. doi: 10.1111/j.1467-8489.2009.00459.x

Nuthall, P.L., 2001. Managerial ability — a review of its basis and potential improvement using psychological concepts. Agricultural Economics, 24: 247–262. doi: 10.1111/j.1574-0862.2001.tb00028.x

Ondersteijn, C. J. M. & Giesen, G. W. J. & Huirne, R. B. M., 2003. "Identification of farmer characteristics and farm strategies explaining changes in environmental management and environmental and economic performance of dairy farms," Agricultural Systems, Elsevier, vol. 78(1), pages 31-55, October.

Pavlos Karanikolas, Stelios Rozakis, Dimitris Kremmydas, 2013. "The multiplicity of goals in tree-cultivating farms in Greece," Working Papers 2013-2, Agricultural University of Athens, Department Of Agricultural Economics.

Rob J.F. Burton, The influence of farmer demographic characteristics on environmental behaviour: A review, Journal of Environmental Management, Volume 135, 15 March 2014, Pages 19-26, ISSN 0301-4797, http://dx.doi.org/10.1016/j.jenvman.2013.12.005

Solms Fritz, 2014, Object-Oriented Analysis and Design using UML, Book, accessed from http://www.fritzsolms.net/sites/default/files/documents/ObjectOrientedAnalysisAndDesignUsingUML.pdf

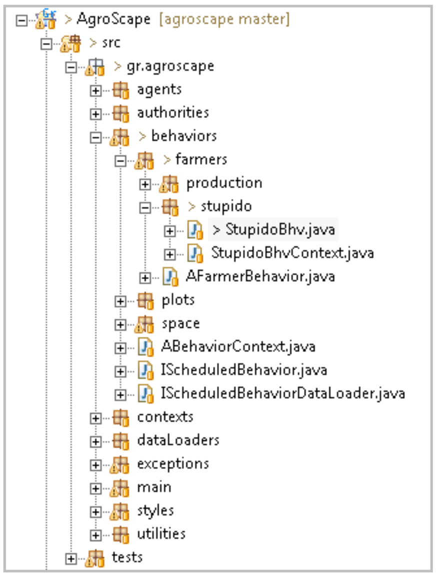
5 Appendix, A “hello world” example: The Stupido Behavior

The implementation of a “hello-world” behavior will now be analyzed. The farmer that is attached to this behavior prints the value of an internal *stupidoProperty* every tick and updates this property every two ticks. The fact that the required *stupidoProperty* attribute is contained within a *StupidoBhv* object and not within a *Farmer* object enables the controlling of complexity to a manageable level for an arbitrarily large number of behaviors. That is because every new behavior can be developed without being affected by other behaviors, since their namespace can be absolutely independent and thus farmers can be attached to any behavior without programming conflicts. One can see the structure of the behavior’s files in Figure 11.

StupidoBhv (Figure 13) is the actual behavior object, extending the *AFarmerBehavior<StupidoBhv>* class. The *AFarmerBehavior<T>* class (Figure 14) is actually enforcing the connection between the behavior object and the farmer object that should be contained there. Since there is an association between the behavior and the farmer objects, one behavior object can use the other behavior objects attached to the same farmer. Finally, in order for the *StupidoBhv* to take action in the simulation’s timeline, it has to implement the *IScheduledBehavior<T>* interface. This is actually realized in two steps. First the *getAnnotatedClass* (lines 34-36, Figure 13) is implemented, returning the behaving object itself. Second, in the returned class (in this case any *StupidoBhv* object), *@ScheduledMethod* annotations have to be inserted accordingly. One can see that this is done in lines 23 and 28 (Figure 13) scheduling the *setRandom* (every 2 ticks) and *print* (every 1 tick) methods respectively.

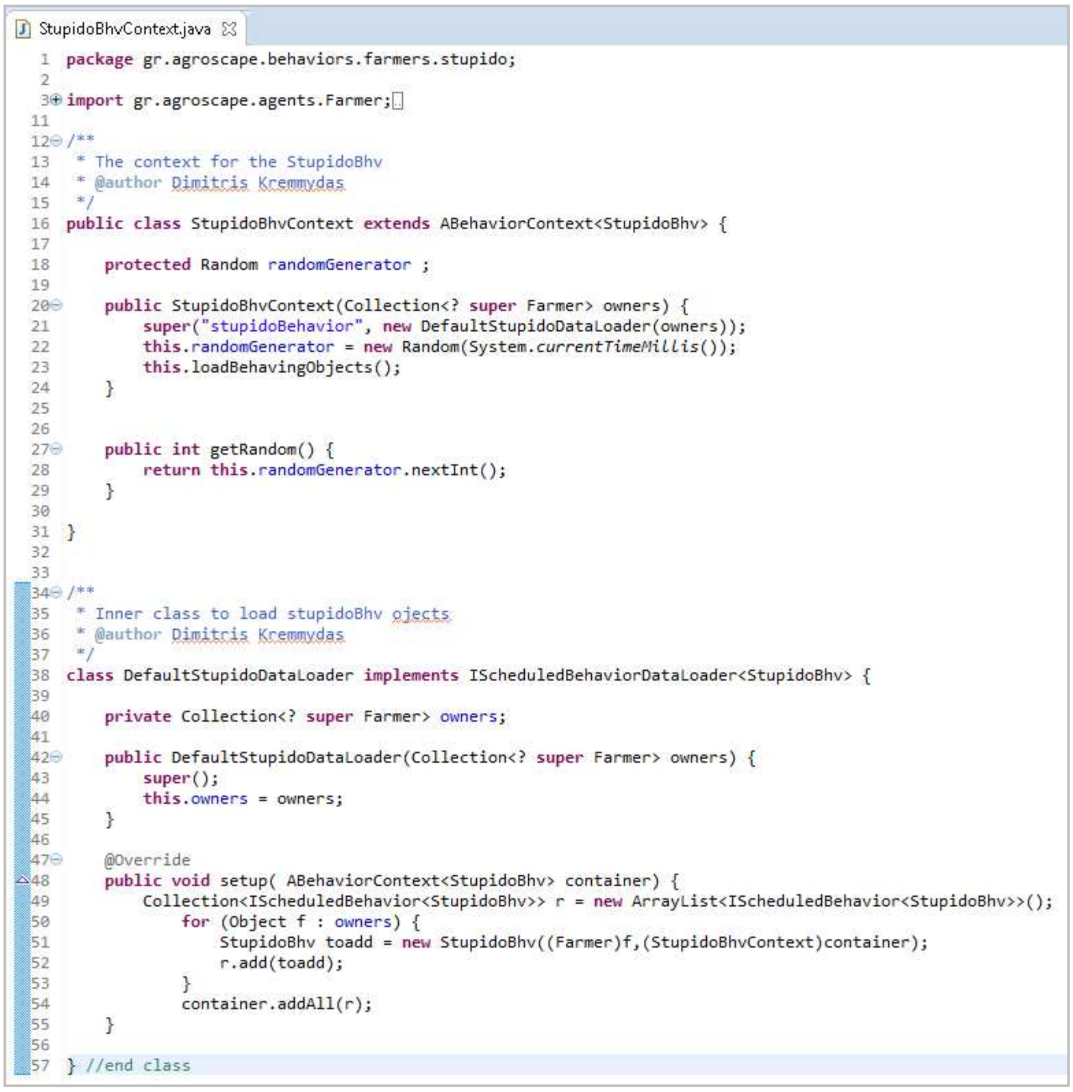
The *StupidoBhvContext* (Figure 12) is the class that contains all the *StupidoBhv* objects and acts as a container of common functionality. The access of the individual behavior objects is facilitated through the container attribute in *StupidoBhv* (line 13, Figure 13). In our case we need a common random generator which is defined in line 18 of *StupidoBhvContext* (Figure 12). This generator is initialized and used by all contained behaving objects.

Also one should note that agents in the behavior context are loaded using a IScheduledBehaviorDataLoader<T> interface. The implementing classes take a collection of farmers, create the behaving objects and add them to the context and to the simulation timeline.



```
> AgroScape [agroscape master]
  > src
    > gr.agroscape
      agents
      authorities
      > behaviors
        > farmers
          production
          > stupido
            > StupidoBhv.java
            StupidoBhvContext.java
          AFarmerBehavior.java
        plots
        space
        ABehaviorContext.java
        IScheduledBehavior.java
        IScheduledBehaviorDataLoader.java
      contexts
      dataLoaders
      exceptions
      main
      styles
      utilities
    tests
```

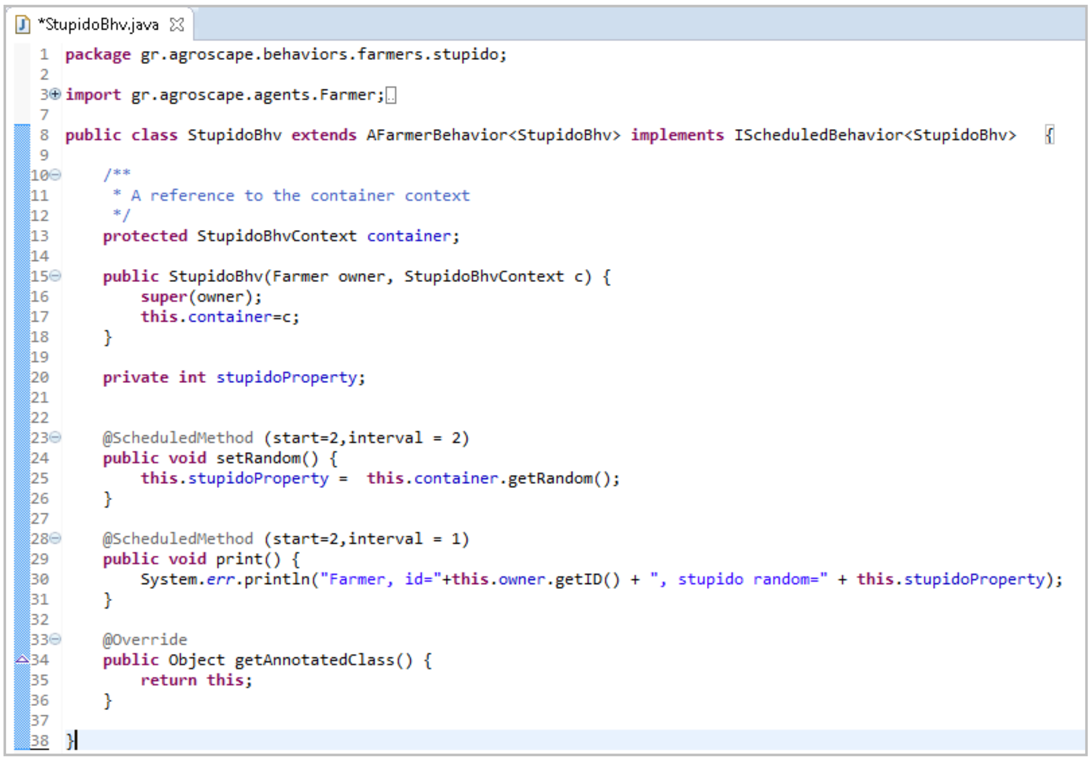
Figure 11, The structure of the files in the Stupido behavior



StupidoBhvContext.java

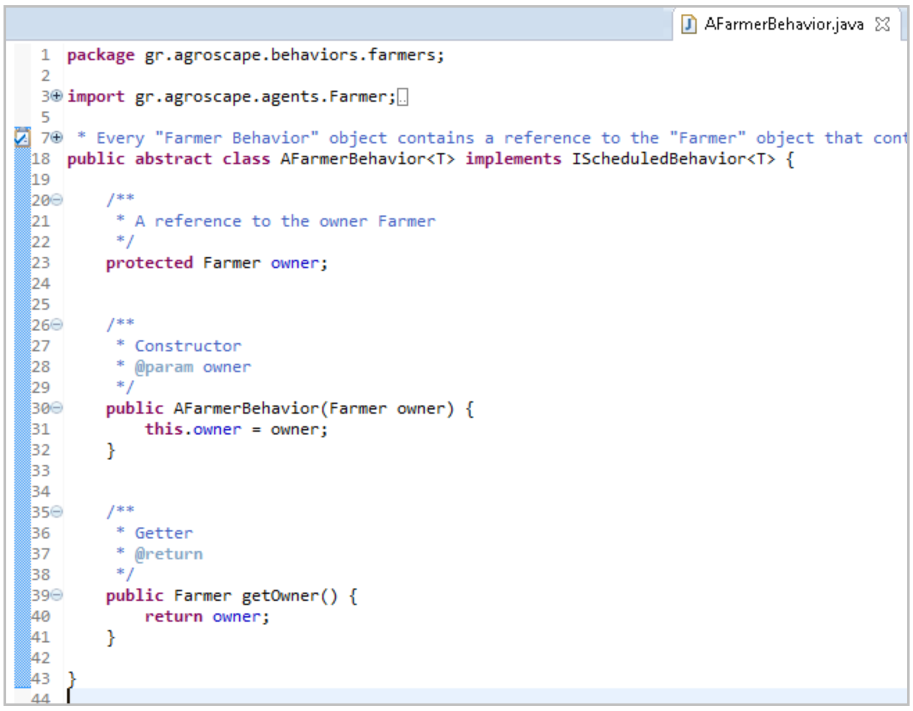
```java
package gr.agroscape.behaviors.farmers.stupido;

import gr.agroscape.agents.Farmer;

/**
 * The context for the StupidoBhv
 * @author Dimitris Kremmydas
 */
public class StupidoBhvContext extends ABehaviorContext<StupidoBhv> {

    protected Random randomGenerator ;

    public StupidoBhvContext(Collection<? super Farmer> owners) {
        super("stupidoBehavior", new DefaultStupidoDataLoader(owners));
        this.randomGenerator = new Random(System.currentTimeMillis());
        this.loadBehavingObjects();
    }


    public int getRandom() {
        return this.randomGenerator.nextInt();
    }

}


/**
 * Inner class to load stupidoBhv ojects
 * @author Dimitris Kremmydas
 */
class DefaultStupidoDataLoader implements IScheduledBehaviorDataLoader<StupidoBhv> {

    private Collection<? super Farmer> owners;

    public DefaultStupidoDataLoader(Collection<? super Farmer> owners) {
        super();
        this.owners = owners;
    }

    @Override
    public void setup( ABehaviorContext<StupidoBhv> container) {
        Collection<IScheduledBehavior<StupidoBhv>> r = new ArrayList<IScheduledBehavior<StupidoBhv>>();
            for (Object f : owners) {
                StupidoBhv toadd = new StupidoBhv((Farmer)f,(StupidoBhvContext)container);
                r.add(toadd);
            }
            container.addAll(r);
    }

} //end class
```

Figure 12, The StupidoBhvContext.java code



```java
package gr.agroscape.behaviors.farmers.stupido;

import gr.agroscape.agents.Farmer;

public class StupidoBhv extends AFarmerBehavior<StupidoBhv> implements IScheduledBehavior<StupidoBhv>    {

    /**
     * A reference to the container context
     */
    protected StupidoBhvContext container;

    public StupidoBhv(Farmer owner, StupidoBhvContext c) {
        super(owner);
        this.container=c;
    }

    private int stupidoProperty;


    @ScheduledMethod (start=2,interval = 2)
    public void setRandom() {
        this.stupidoProperty =  this.container.getRandom();
    }

    @ScheduledMethod (start=2,interval = 1)
    public void print() {
        System.err.println("Farmer, id="+this.owner.getID() + ", stupido random=" + this.stupidoProperty);
    }

    @Override
    public Object getAnnotatedClass() {
        return this;
    }

}
```

Figure 13, The StupidoBhv.java code



```java
package gr.agroscape.behaviors.farmers;

import gr.agroscape.agents.Farmer;

 * Every "Farmer Behavior" object contains a reference to the "Farmer" object that cont
public abstract class AFarmerBehavior<T> implements IScheduledBehavior<T> {

    /**
     * A reference to the owner Farmer
     */
    protected Farmer owner;


    /**
     * Constructor
     * @param owner
     */
    public AFarmerBehavior(Farmer owner) {
        this.owner = owner;
    }


    /**
     * Getter
     * @return
     */
    public Farmer getOwner() {
        return owner;
    }

}
```

Figure 14, The AFarmerBehavior source code