



## PET User Guide

*Participatory Extension*

*Created by Vilmos Kozma and Marton David Ivanyi*

*For ELTE-IKKK*



*2007 October*

## Contents

PET User Guide .....	1
Created by Vilmos Kozma and Marton David Ivanyi .....	1
For ELTE-IKKK .....	1
2007 October .....	1
Contents .....	2
1 Introduction .....	3
1.1 Agent-Based Modeling .....	3
1.2 MASS .....	3
1.3 PET .....	4
1.4 Purpose .....	4
1.5 System requirements .....	4
1.5.1 Minimum System Requirements .....	4
1.5.2 Optimal System Requirements .....	4
1.6 Roles .....	4
1.7 Login .....	4
1.7.1 Login failed .....	6
1.8 User navigation .....	6
2 Models and Simulations .....	7
2.1 Creating simulations .....	7
2.2 Joining simulations .....	7
2.3 User settings (for simulations) .....	8
2.4 Simulation control .....	9
2.5 Controlling agents .....	10
2.5.1 Release agent .....	11
2.5.2 Reproducing a simulation .....	11
3 Conclusion .....	12
4 Appendix A - Model Descriptions .....	13
4.1 Fire .....	13
4.2 Hunter .....	13
4.3 Zerosum .....	13
4.3.1 Mathematical model .....	14
4.3.2 Short analysis .....	14
4.3.3 Agent strategies .....	14
4.3.4 References for Zerosum .....	15

# 1 Introduction

## 1.1 Agent-Based Modeling

Agent-based modeling is a branch of computer simulation. It models the individual, together with its imperfections (e.g., limited cognitive or computational abilities), its idiosyncrasies and personal interactions. The approach builds the model from 'the bottom-up', focusing mostly on micro rules and seeking to understand the emergence of macro behavior. Participatory simulation - a branch of agent-based simulation - is a methodology building on the synergy of human actors and artificial agents, excelling in the training and decision-making support areas. In participatory simulations some agents are controlled by users, while others are software governed.

## 1.2 MASS

The Multi-Agent Simulation Suite (MASS) is a software package intended to enable modelers to utilize the tools of agent-based simulation in various fields, without having to develop heavy programming skills.

MASS consists of four applications built around a simulation core. The simulation suite has its own core called the Multi-Agent Core (MAC), but it is also able to run on the popular Repast core. Being multi-core enables modelers to verify that results are core-independent, thus we plan to further develop this option. The Functional Agent-Based Language for Simulation (FABLES) is a programming language and its integrated modelling environment specially designed for creating agent-based simulations. The Model Exploration Module (MEME) is a tool that enables orchestrating experiments, managing results and has support for their analysis. The Participatory Extension (PET) is an optional web-based environment for multi-agent and participatory simulations. The fourth element of MASS, the Visualization Package does not translate into a standalone application. It consists of the various implementations of charts and visualizations used in all the other software.

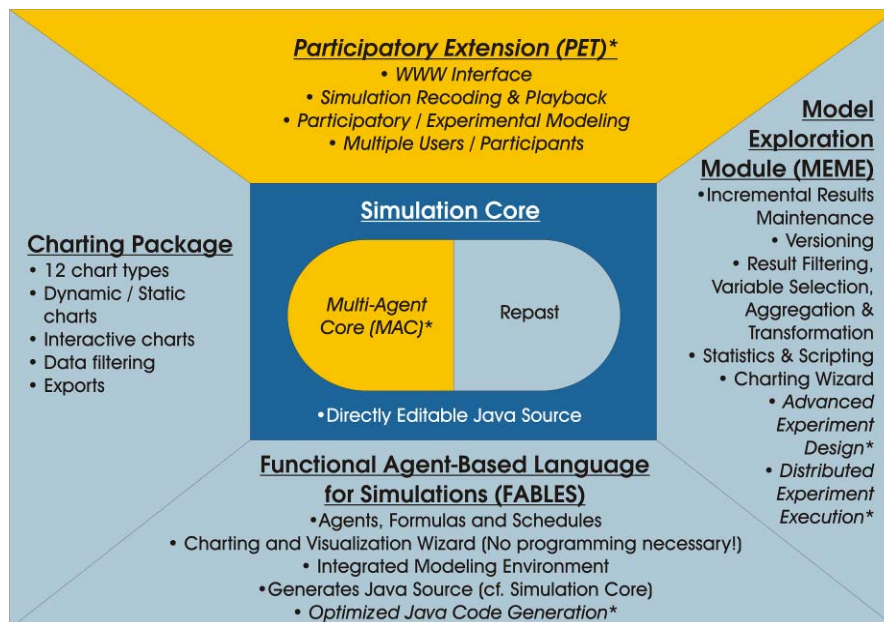


Figure 1 - Multi-Agent Simulation Suite

## 1.3 PET

PET is the web-based environment designed for administrating and running models and multi-agent simulations. It is based either on its native core, the Multi-Agent Core (MAC) or on Repast J. PET is part of the Multi-Agent Simulation Suite (MASS).

## 1.4 Purpose

This user guide is prepared for the users of the Participatory Extension (PET). As this paper is intended for the end user of the environment it does not go into the issues of programming models or administrating PET. Those issues are discussed in the 'PET model writing manual' and 'PET Administrator Guide' respectively.

## 1.5 System requirements

### 1.5.1 Minimum System Requirements

To run PET a Java 1.5 compatible platform or later (Pentium 166MHz or faster, minimum 64 MB RAM) is needed.

### 1.5.2 Optimal System Requirements

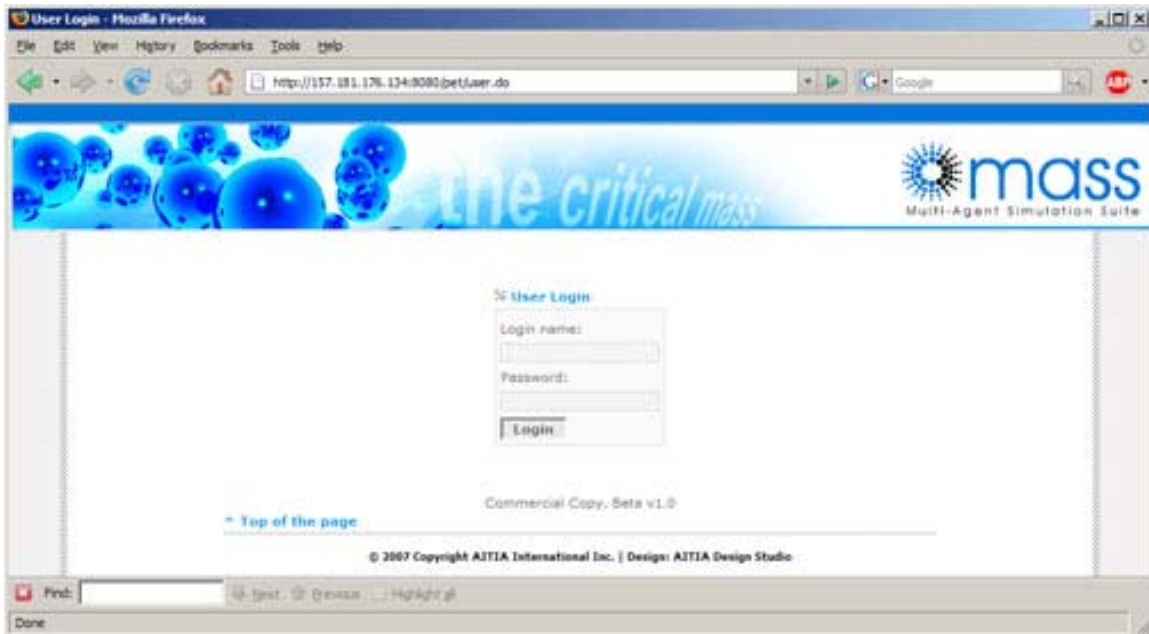
Equivalent to a Pentium 1000MHz or faster and 256 MB RAM.

## 1.6 Roles

There are two main types of users in the Participatory Extension; end users who run and participate in simulations and administrators, who create models and administrate end users for example. This guide is prepared for the first group while administrative issues are covered in the *PET Administrator's Manual*.

## 1.7 Login

In order to start PET open your web browser – Mozilla Firefox or MS Internet Explorer preferably – and type in the address provided by your administrator. PET uses a web interface and you're required to log in to it with the username and password given to your system administrator is just like on an ordinary Internet site. See the screenshot below for an example of the login screen.



**Figure 2 - Login**

If you see a login field similar to the above, simply type in your user name and password in their respective fields and click 'Login'. Upon authentication the User main screen should appear (see below).



**Figure 3 - Main screen - Simulation menu**

### 1.7.1 Login failed

If you have mistyped either the username or the password you get the following message:



**Figure 4 - Login failed**

Any other kind error or error messages occurring during the login procedure are of outside causes. In these cases you need to contact your system administrator for further assistance.

## 1.8 User navigation

User side navigation is organized around simulations. All existing and available simulations are listed on the main screen (Figure 3) appearing after login. The user can connect to simulations (see Figure 7), create new ones (see Figure 6) from here, and after joining a simulation can read descriptions and access every other option.



**Figure 5 - Navigation bar on the Simulation control page**

Once on the Simulation Control page (Figure 5) available options and commands are presented in a three level hierarchy on the user side. Simulation control buttons and commands are located on the simulation control bar (dark grey surface above).

## 2 Models and Simulations

The Participatory Extension operates through a hierarchy of model families, models, and simulations. Models are set up by administrators from model families by declaring setting the number of agents in the particular version of the model and doing various other configurations. A simulation is an instance of a model available that can be run, replayed and participated in (i.e. numerous simulations can belong to the same model). End users see models, and simulations grouped by models (see Figure 3). End users can reach their menu of models, hence available simulations, by clicking on the 'Simulations' link on their user surface (see Figure 5).

### 2.1 Creating simulations

End users create simulations by starting new instances of the models available for them. Simulations are created and associated (owner) with the user starting them by clicking on the *Start new* button (see below) in the listing of available models.



Figure 6 - Start simulation

Note that the administrator can also build simulations. The user creating the simulation automatically becomes the owner of that simulation. Besides administrators only the owner can delete a simulation.

### 2.2 Joining simulations

Users can play simulation started by themselves or owned by someone else if there are agents available to be controlled in the simulation and/or the user is allowed to see the particular simulation.



Figure 7 - Join/Connect

End users can join simulations from the main screen by clicking on *Connect* in the simulation row – if there is one – of the given model. This row provides basic information; simulation name, whether the user owns the simulation, whether the simulation has been played before, the simulation's status, the previously mentioned Connect command, and a *Delete* command that is only active if the user owns the particular simulation. (See above.)

## 2.3 User settings (for simulations)

Although all simulations appearing on the user side of PET are “ready to go”, in some models the user can change the values of agent properties. These models are set up by administrators with default agent property values that can be modified by the user who starts the simulation. Simulations with modifiable settings have an extra option - *Agent properties* - on their *Simulation Control* page (see below).

**Simulations**

↳ Fire - User parameter test - Fire 7 » [Agent list](#) | [Agent properties](#)

Modifiable agent properties


Agent list

**Figure 8 - Simulation with modifiable property values**


On the *Modifiable agent properties* page (see Figure 9 below) all the agents with modifiable properties are listed. The agents with the same modifiable properties are grouped together. All agent properties can be listed by checking *Show read-only properties*.

Modifiable agent properties


Show read only properties

 **Fire** Agent id: 586 Update


Property	Class	Value	Variety
Increment	java.lang.Double	1000.0	1

 **Person** Update 3 instances.

Property	Class	Value	Variety
Eyeshot	java.lang.Double	30.0	1

 **Person** [List instances](#) Update 2 instances.

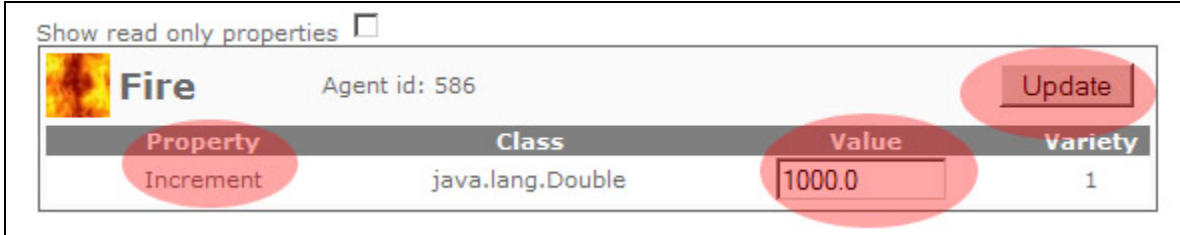
Property	Class	Value	Variety
Eyeshot	java.lang.Double	30.0	1
Location X	java.lang.Integer		2
Location Y	java.lang.Integer		2

 **World** Agent id: 585 Update

Property	Class	Value	Variety
Heat spread on walls	java.lang.Double	0.95	1

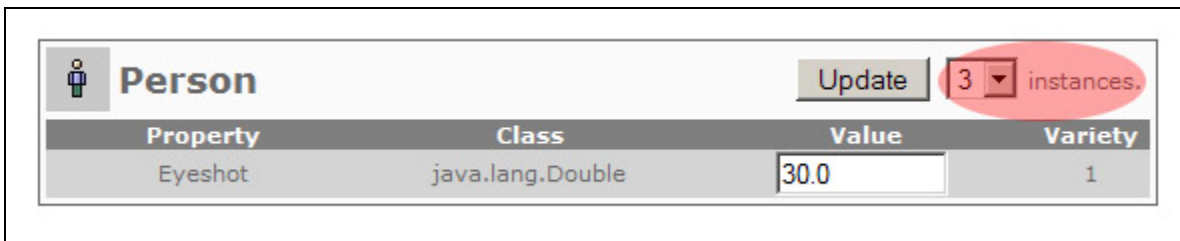
**Figure 9 - Modifiable agent properties list**

In order to change a property, type in the desired value in its *Value* field and press the 'Update' button (see below where we set the property Increment to 1000.0).



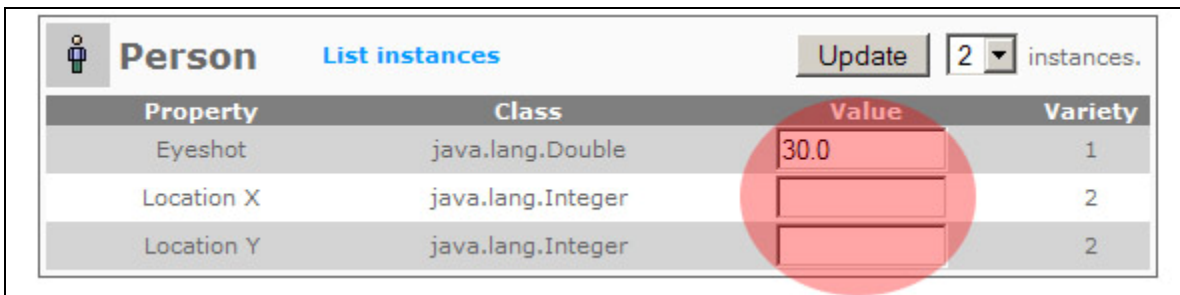
**Figure 10 - Press *Update* to change property value**

If there are a number of agents with the same modifiable property, the number of instances for which the value can be modified can be selected from the drop down list on the side of the *Update* button (Figure 11).



**Figure 11 - More than one agents with the same modifiable property value**

With multiple agents with multiple properties it is possible to list and provide values for all instances separately by clicking on 'List instances' link (Figure 11).



**Figure 12 - Two agents with the same three modifiable properties**

The variety column shows the number of different values of the given property in the agents.

## 2.4 Simulation control

By selecting a simulation the user is taken to the *Simulation control page*. From here user can start, pause, resume and replay their simulations, as well as observe visualizations and control their agents in participatory models. Note, that the controlling functions are only available for the owner of the simulation



**Figure 13 – Simulation controls**

The first two of four control buttons numbered on the above picture represent *Simulation mode* (1.) and *Replay mode* (2.). On the above figure the simulations hasn't been started yet. Once started by pressing button 3 the first button becomes blue (active), indicating that it is in *Simulation mode*. If paused (4.), the simulation can be switched to *Replay mode* by pressing button 2. To start the replay, simply press *Play* (3.). It can be switched back to simulation mode by pausing and pressing button 1.

## 2.5 Controlling agents

In participatory models the users can control all or some of the agents, while the computer controls the rest. To "get hold of" an agent *Connect* has to be pressed in the *Simulation menu* (Figure 7).

If there is only one controllable agent in a participatory models PET automatically takes the user to that agents *Control agent* page when joining the simulation (see below).



Figure 14 – Agent control

On the screenshot above the visualization is not embedded in the agent control page. This is rather the exception than the rule, by clicking on *visualization* a 3D visualization applet is launched in this case.

Under *Actions* all the actions the agent can take in the model are listed. On the screenshot above these are all related to movement, but this could be anything the creator of the model specifies. It is important to mention that when a user clicks on an action the action is not performed at once. Instead, the selected action is placed into a buffer and the action is performed when the agent is activated. In normal cases this happens only once in each simulation round. The buffer can hold only one action at the same time. If the buffer is not empty and a new action is taken, the new action overwrites the older one, and the old one is never performed.

If there is more than one controllable agent in the simulation the user is taken to the list of (controllable) agents (see Figure 15) where agents can be selected for both view and control.

Agent list

Choose an agent!

Id	Type	Actions
7	World	1.  2.
8	Fire	
9	Person	
10	Person	
11	Person	

Figure 15 - 1. Observe agent; 2. Control agent

Note that on the figure above the highlighted icons are blue, which represents that agents 9, 10 and 11 are controllable.

### 2.5.1 Release agent

In order to release an agent – so it can manage on its own or be controlled by other users – simply press the *Release agent* command on the *Control agent* page.

**Release agent** Mode: Simulation

**Name:** Fire 4  
**Type:** Fire  
**Created at:** Wed Aug 22 04:16:21 CEST 2007  
**Tick count:** 0

Figure 16 - Release agent

### 2.5.2 Reproducing a simulation

In PET simulations can be replayed any number of times. This is possible because all actions that are taken by the users are logged, and by using the replay function the simulation always performs the same run.

## 3 Conclusion

PET is simulation software providing a quality platform for modelers to create participatory simulations in which individuals and synthetic agents can co-operate simultaneously. Its user interface is intuitive and easy to use allowing the users to learn managing the software quickly. The user interface and the administrator's interface are separated well which makes the software more manageable. This document introduces the interface developed for end users. The end user interface allows the users to create, control and participate in simulations, to control agents and set their properties during runtime.

## 4 Appendix A - Model Descriptions

This appendix describes the model-families and their configured model instances coming with the default system installation.

### 4.1 Fire

This model is a simple example of simulating emergency situations. The model's *world* is a building with rooms that are separated by walls. All of a sudden a fire breaks out at a certain location in the building. Fire spreads at a constant rate, but walls slow it down a little. Everybody in the building tries to find an exit to flee. Some succeed, but usually some don't. We provide the model with 2-D and 3-D visualization for the agents and space. The simulated agents don't know the location of the exits and only have a limited vision. However, they follow each other, try to move away from fire and explore their neighborhood.

Two somewhat different ready-to-run models are provided with PET under the Fire model family:

1. *Watch the Fire*: You can be an observer of a spreading fire and the agents trying to escape from a building seen from above (2D visualization).
2. *Escape the Fire*: You are one of the agents trying to escape the fire. The building is displayed from within (3D<sup>1</sup> visualization)!

### 4.2 Hunter

Hunter is a basic model for two types of agents (technically three, as MASS considers space as an agent too). Some agents are *hunters* while others are *preys*. They "live" in a two-dimensional periodic space. As their names imply, the hunters try to catch preys, meanwhile preys will try to avoid them. When a hunter catches its prey, the prey disappears and is replaced to a random location in the space, or removed from the game depending on model configuration. Hunters and preys are very similar in their behavior to some extent, they both look for the (closest) opposite type of agents each turn, but hunters move toward preys, while preys move away from hunters.

The programming of the Hunter model is described in detail in *PET Tutorial!*

Two somewhat different ready-to-run models are provided with PET under the Hunter model family:

1. *Watch the Hunt*: You are an observer of the hunt.
2. *Don't Be a Prey!*: You are controlling one of the preys in the hunt. Try to avoid the hunters!

### 4.3 Zerosum

The Zerosum (Huber and Kirchler; Yoneda et al) model is a simple information-based zero-sum game on the futures market. It reveals a basic fact that the most and least informed players in the market earn more, than the mildly informed ones.

---

<sup>1</sup> Running 3D visualizations launches an external Java application. Please allow for the visualization applet to download before the first run in 3D. Note that users might experience problems while running 3D visualizations with some video settings.

### 4.3.1 Mathematical model

The game is played by  $2M + 1$  players.

1. At the beginning of each round Player  $i$  ( $0 \leq i \leq 2M$ ) shows his reservation price  $R_i$ , by which he makes the following contract: he buys (sells) a unit of a future product if  $R_i$  is higher (lower) than the actual price  $P$ .
2. The auctioneer gathers all the reservation prices and declares the median of them as  $P$  so that demand equals supply in the futures market.
3. After the future market is closed, the true value of the commodity  $V$  is revealed. It is determined exogenously as the sum of  $2M$  stochastic variables:  $X_1, X_2, \dots$ , and  $X_{2M}$ , which are determined to be 0 or 1 independently with one another with the same probability 0.5 for each round.
4. Those who bought (sold) the commodity in the future market must sell (buy) it in the spot market at the true value  $V$  to close their accounts. Hence each player's profit is determined as soon as  $V$  is revealed. Those players with  $P < R_i$  buy the commodity at  $P$  in the future market and sell it at  $V$  in the spot market so that they each earn  $V - P$  of profit, while those players with  $R_i < P$  sell the commodity at  $P$  in the future market and buy it at  $V$  in the spot market so that they each earn  $P - V$  of profit. Needless to say, profit is loss if it is negative and the sum of all profit is always zero:  $\Sigma(P - V) + \Sigma(V - P) = 0$ .

We assume that before he determines  $R_i$ , Player  $i$  can correctly see the first  $i$  values of  $X_1, X_2, \dots$ , and  $X_i$  (Player 0 cannot predict any of them). Apparently Player  $i$  has an advantage over Player  $j$  ( $j < i$ ), for the former knows what the latter knows ( $X_1, X_2, \dots$ , and  $X_j$ ) as well as what the latter does not ( $X_{j+1}, X_{j+1}, \dots$ , and  $X_i$ ). Hence, we will refer to  $i$  as "information level".

### 4.3.2 Short analysis

Player  $2M$ , the most informed player who can see every  $X_i$ , most probably earns positive profit in the long run. In fact he can choose  $\Sigma X_i$  as his reservation price  $R_{2M}$ ; then, as is readily checked, he earns zero profit if  $P$  happens to be equal to  $V$  or positive profit in more plausible cases where  $P \neq V$ . However, Player 0, the least informed player who knows none of  $X_i$  may not suffer negative gain in the long run. He may always continue to choose such a high (low)  $R_0$  that makes him buy (sell) in the future market for every round or he may choose  $R_0$  randomly for each period. In either case he is able to expect zero profit in the long run, because  $P < V$  and  $V < P$  is realized with the same probability.

A question emerges from these special circumstances, i.e., from the fact that the game is zero-sum. If the most informed player can expect a positive profit while the least-informed player can expect zero profit, some middle-informed player must suffer a loss. Yet it seems strange that a player who is better informed earns less profit than a player who is less informed.

### 4.3.3 Agent strategies

The agents play the following two simple strategies:

The expected-value strategy:

$$R_i = (\Sigma X_i) + 0.5 \times (2M - i)$$

Here the right-hand side represents the value of  $V$  that Player  $i$  expects; the first term is the sum of the stochastic variables, whose values Player  $i$  knows, while the second term stands for the expectation of the sum of  $X_{i+1}, X_{i+2}, \dots$ , and  $X_{2M}$ , whose values he does not know.

The extreme strategy:

$R_i = (\sum X_i)$  with probability 0.5

$R_i = (\sum X_i) + (2M - i)$  with probability 0.5

Here  $(\sum X_i)$  represents the minimum value of  $V$  that Player  $i$  expects while  $(\sum X_i) + (2M - i)$  stands for its maximum value that Player  $i$  expects.

Two somewhat different ready-to-run models are provided with PET under the Zerosum model family:

1. *One against the market 1*: All the agents are controlled by the computer you are an observer on the market. This is a one against the masses setting, where all but one agent plays the expected-value strategy.
2. *One against the market 2*: This is basically the same as 'One against the market 1' but the strategies are transposed, most agents play the extreme strategy.
3. *Control the market 1*: Users can control agent number 0, 10, 20, 30, 40, 50, 70, 80 or 90 out of 100 agents. All the other agents play the expected-value strategy.
4. *Control the market 2*: This is about the same as the previous one, but the artificially controlled agents play the extreme strategy.

In the latter two simulations users can control agents, hence make pricing decisions in the market.

#### 4.3.4 References for Zerosum

Jürgen Huber and Michael Kirchler "The Value of Information in Markets with Heterogeneously Informed Traders – and Experimental and a Simulation Approach", presented at 9th Workshop on Economics and Heterogeneous Interacting Agents (WEHIA2004), Kyoto University, Kyoto, Japan

Hiroyasu Yoneda, Gen Masumoto and Sobei H. Oda: „Marginal Contribution of Information to Profit in a Zero-sum Game“, presented at *EES2004: Experiments in Economic Sciences - New Approaches to Solving Real-world Problems*