Agent-based Simulation Platform Evaluation in the Context of Human Behavior Modeling

Michal Laclavík[1], Štefan Dlugolinský[1], Martin Šeleng[1], Marcel Kvassay[1],
Bernhard Schneider[2], Holger Bracker[2],
Michał Wrzeszcz[3], Jacek Kitowski[3], Ladislav Hluchý[1]

[1]Institute of Informatics, Slovak Academy of Sciences,
Dúbravská cesta 9, 845 07 Bratislava, Slovakia
{laclavik.ui, stefan.dlugolinsky, martin.seleng, marcel.kvassay, hluchy.ui}@savba.sk
[2]EADS Deutschland GmbH
Landshuter Straße 26, 85716 Unterschleißheim, Germany
{bernhard.schneider, holger.bracker}@cassidian.com
[3]Academic Computer Centre CYFRONET,
University of Science and Technology in Cracow, Poland
michalwrzeszcz@gmail.com, kito@agh.edu.pl

Abstract. In this paper we provide a brief survey of agent based simulation (ABS) platforms and evaluate two of them – NetLogo and MASON – by implementing an exemplary scenario in the context of human behavior modeling. We define twelve evaluation points, which we discuss for both of the evaluated systems. The purpose of our evaluation is to identify the best ABS platform for parametric studies (data farming) of human behavior, but we intend to use the system also for training purposes. That is why we also discuss one of serious game platform representatives – VBS2.

Keywords: agent-based simulation, human behavior modeling.

1 Introduction

Human Behavior Modeling is an important area of computational science with implications not only for social sciences, but also for economics, epidemiology and other fields. Scientific literature abounds in heterogeneous and highly specialized, theoretically founded concepts of human cognition, emotion and other behavior aspects. The task to find a simulation framework that would allow effective implementation of such conceptions for different aspects of real human behavior to interoperate is particularly challenging. Our motivation for this paper derives from the EDA project A-0938-RT-GC EUSAS (European Urban Simulation for Asymmetric Scenarios) whose goals and requirements provide the context and a guideline for our evaluation of the existing systems.

The EUSAS project focuses on asymmetric security threats in urban terrain. Its goal is to develop an all-in-one tool enhancing the mission analysis capabilities as

well as virtual training of real human beings (security forces) in a highly realistic 3D cyber environment. In virtual trainings, simulated characters (civilians) with highly realistic patterns of behavior would interact with real people (security forces), while in the mission analysis (Data Farming) mode both the civilians and the security forces would be simulated. A natural choice for the simulations of this kind is an agent-based simulation [1].

We have perused the existing surveys of agent-based simulation frameworks (ABS) with special respect to EUSAS-project goals. In the first round of the evaluation we reviewed a high number of various agent based platforms [1] based on published surveys and the information on the web. In the second round – "Evaluation by Implementation" - we evaluated in depth the two most promising ABS systems by implementing an exemplary scenario described in section 2, which reflects the main needs of the EUSAS-project.

Besides smooth incorporation in highly realistic virtual trainings, even more important was the ease of use in multi-parametric studies (Data Farming) where many instances of the same ABS run in parallel, each with different values of input parameters. The results of each run are stored in a repository for subsequent analysis.

Several ABS that we considered were based on Logo languages (derived from Lisp). Here, NetLogo [4] was the most relevant representative. Other platforms included Repast[1] or Mason [9], which can run high number of agents by executing each agent in small steps. In contradistinction to step-based implementations, there are also event-based or thread-based modeling toolkits, such as CoJack[2] or Jason[3]. Here, each agent is executed in a separate thread and behavior is updated based on events. The event-based approach is used in VBS2 serious game component, which we plan to use for virtual trainings in the EUSAS system. Step-based ABS platform, such as NetLogo, Repast or Mason, allow simulation of a higher number of agents, and models are easier to debug, although there is an extra effort involved in integrating them with the thread and event-based serious game component for the purpose of virtual training. Creation of a large number of threads (e.g. thousands) would be inefficient in any of the thread-based toolkits.

Since we did not have the resources to evaluate all the existing platforms by implementation, we first shortlisted the candidates based on the existing MAS surveys and then evaluated the two most promising candidates by implementing an exemplary human behavior scenario which represented our domain. Based on the surveys, MASON and NetLogo were identified as the two most promising systems, each with a slightly different strategy. Compared to MASON, NetLogo was more focused on educational purposes, but still with a good capability for simple and fast modeling, implementation, visualization as well as good visual analytical tools. Both MASON and NetLogo are step-based platforms using discrete-event simulation model.

Apart from simulations for multi-parametric studies, we also intend to conduct simulations where real humans can interact, in order to support virtual trainings. Therefore we have also explored the possibilities for integration with a virtual reality toolkit, such as VBS2.

---

[1] http://repast.sourceforge.net/

[2] http://www.agent-software.com.au/products/cojack/index.html

[3] http://jason.sourceforge.net/Jason/Jason.html

1.1 Existing Survey Literature

The most relevant survey of ABS is [7] from 2005, which tested 5 ABS on a simple (so called *Stupid Agent*) scenario [7]. The evaluated platforms were NetLogo, MASON, Repast, Swarm and Java Swarm. MASON was evaluated as the fastest. All the features could be implemented quite well but its extensions and support tools were not all in a good shape then. NetLogo was found to be the simplest for agent modeling and implementation with good analytical tools and visualization. According to our recent research, NetLogo and MASON have been the fastest evolving ABS platforms since then. Repast was evaluated quite high. Repast is a well known platform with current beta version of Repast Symphony, which would be worth to evaluate by implementation, however Repast has several implementations and it is not clear which version it would be best to evaluate. Repast claimed to support NetLogo models, so we tried to import our implementation of NetLogo model into Repast, but we did not succeed since errors cropped up during the import process. When Repast Symphony reaches a stable release, it might be a worthwhile candidate for evaluation.

In a 2002 study [5], Repast, Swarm, Ascape, Smalltalk, StarLogo and AgentSheet were compared. Only Repast can be considered from this list nowadays. The most recent survey of MAS platforms is [6] using similar approach to [7]. It covers many platforms we considered based on available literature. We do not provide the list here but they are listed in [2] and many of them are also listed on the Wikipedia page on agent-based simulation[6]. As already mentioned, some of these platforms were evaluated on a StupidModel Programming experience for execution speed as well as ability to fully or partially implement the chosen features. StupidModel[7] was broken down into 16 small tasks. It was implemented also in EcoLab C++ based Platform [8] and showed that EcoLab[8] was capable of handling this model with similar performance as MASON but with worse GUI capability. StupidModel, however, is not fully relevant for our purposes. We decided to evaluate MASON and NetLogo by implementing our exemplary scenario (section 2), a simplified generic version of the kind of scenarios envisaged for human modeling in the EUSAS-project.

1.2 Evaluated Features

In order to evaluate the chosen simulation frameworks, we have defined 12 generic evaluation aspects on which we focused while implementing the scenario. These points are generic and could be relevant for other kinds of simulations as well, but we have evaluated them specifically in the context of implementing a typical human behavior model:

– *Loading and Representing the Environment and the Scenario*: Here we describe the representation and implementation of the scenario and the physical environment. We also discuss the possibility to load the environment model from GIS data as well as support for 3D, 2D and layered environments.

[6] http://en.wikipedia.org/wiki/Comparison_of_agent-based_modeling_software
[7] http://condor.depaul.edu/slytinen/abm/StupidModel/
[8] http://ecolab.sourceforge.net/

- *Creating and Representing Agents*: We discuss how to create, represent and implement agents in the evaluated system, and how the agents perceive other agents or their environment.
- *Behavior Implementation*: Here we focus on behavior representation and implementation in the evaluated systems.
- *Movement Implementation*: In this point we discuss the support for the physical movement of the agents in the environment, how they pass around obstacles or how a coordinated movement of crowd is supported. This is related to Flocking[9] or Steering[10] behavior[11] of agents.
- *Visualization*: Support for the simulation visualization, but also for running the simulation with no visualization (especially for Data farming purposes).
- *Parameterization*: In order to run parametric studies (Data Farming), we have evaluated ABS support for simulation parameterization.
- *Model check-pointing*: Support for model check pointing – stopping, storing, loading and running simulation from the previously stored break-point.
- *Analytical Tools*: Support and analytical tools of ABS are discussed here.
- *Logging*: To analyze multi-parametric studies and the measures of effectiveness, we need to log the progress of the simulation. We discuss here the ABS support for logging.
- *Performance*: We discuss the perceived performance of ABS. In addition we provide performance measures for NetLogo and MASON for 10, 100, 1000 and 10000 civilian agents.
- *Standards:* We discuss possible related standards such as HLA or FIPA.
- *Development Environment* of evaluated platforms is discussed as well.

2 Human Behavior Modeling: Exemplary Scenario

In order to support tool evaluation with reference to the needs of human behavior modeling and the EUSAS-project as described in the introduction, an exemplary scenario [3] had to be defined. Hence, the exemplary scenario had to feature relevant aspects of human behavior in a given context, deriving from real world observations, and thereby reflecting the basic properties of the application context set by the EUSAS-project, but also to be kept as simple as possible in order to keep the implementation effort low and to enable rapid prototyping. Additionally the scenario should provide sufficient space for scenario evolution and should contain reactive and deliberative control structures for involved agents. Since the main focus of the paper

[9] http://www.red3d.com/cwr/boids/
[10] http://opensteer.sourceforge.net/
[11] http://www.shiffman.net/teaching/nature/steering/

lies upon technical evaluation of the simulation frameworks in order to select the one supporting the needs of the EUSAS-project best, the following description is intended to provide an overview about the scenario elements, not to present the underlying formal model for the different aspects of agent behavior.

The scenario comprises a civil protester and a soldier, both represented as agents acting in a common environment.

The environment is a 2D grid composed of quadratic cells sized 0.5m x 0.5m. Each cell is labeled to describe its nature, respectively the actions which may take place if an agent enters the cell. The labels are: fight area, stone picking area, safety area, soldiers area, barrier.

Depending on the internal state of the civil protester agent, he resists in a predefined safety area of the environment or shows aggressive actions against the local authority represented by the soldier agent. Aggressiveness of the civilian protester is expressed by picking up a stone, approaching the soldier agent and throwing the stone towards him. Fearful behavior in contrast is expressed by flight reactions into a predefined safety area. The soldier agent's behavior is based on a text book case, hence he behaves according to a given rule set and is not triggered by any human motives. Being threatened, the soldier agent is allowed to take countermeasures against the threatening civilian agent.

The behavior of the civilian agent requires the following elements: stimulating events in the environment, motives, action plans and predefined behavior patterns. Based on the psychological considerations in [11], the civilian agent architecture contains three motives: fear, anger and an observe-motive. The theory of cognitive appraisal for emotions [10] serves as a theoretical basis for modeling the emergence and temporary course of the emotional motives anger and fear. Accordingly, stimulating events in the environment (e.g. movements or actions of the soldier agent) being perceived and cognitively evaluated by the civilian agent influence the intensity of his emotional motives fear and anger. The concrete computation of the corresponding motive intensities is done with the help of differential equations. The observe-motive can be regarded as "fall-back-motive" with constant intensity. All available motives compete against each other; the motive with the highest intensity dominates the other motives and determines the concrete shape of behavior that the civil agent shows at a certain point of time.

Both the civilian and the soldier execute their actions according to individual internal actions plans. An action is defined as a non-interruptible, time-consuming operation performed by an agent. For each action, a set of preconditions is defined. An action plan is a list of actions to be performed one after another. Action plans can get interrupted. This happens if the dominant motive changes or the precondition for the next action in the plan is not fulfilled. In this case, the whole action plan gets rejected and the agent is forced to determine a new goal to reach and, consequently, to construct a new action plan.

3 Evaluation through Implementation

In this chapter we describe our experience with implementing the exemplary scenario described in section 2 in both MASON[12] and NetLogo[13]. Scenario environment is grid based but in both NetLogo and MASON we implemented it as continuous, so agents interact and move continuously with a small defined discrete step. Both the evaluated systems are step-based simulation systems based on discrete-events. Although VBS2 (the serious game training component) is not directly competing with NetLogo or MASON, the chosen candidate would be later integrated with it for training purposes. Therefore, at appropriate places, we also refer to our implementation experiments with VBS2 and discuss potential integration issues. Figures below show screenshots of the exemplary scenario in NetLogo (Figure 1, left) and MASON (Figure 1, middle).

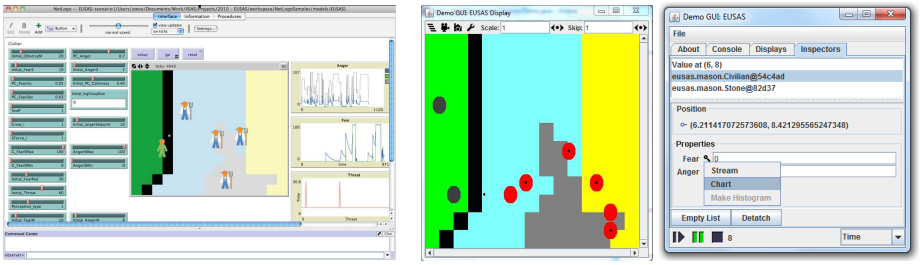

Fig. 1. Left: Exemplary Scenario implemented in NetLogo with variable sliders and charts; Middle: Exemplary Scenario in MASON; right: MASON console window, where inspector of agent variables is open.

3.1 Loading and Representing Environment

The NetLogo world is a two-dimensional grid of "patches". NetLogo supports three-dimensional environments, but the status of this feature is still experimental. Patches are the individual squares in the grid. Each patch is a square piece of "ground" over which the agents (turtles) can move. The way the world of patches is connected can change. World can be wrapped horizontally, vertically or in both directions (torus). In our exemplary scenario, we wanted to find a way how to load a map of areas into the NetLogo 2D world. We found it very convenient to represent the simulation scenario map by a bitmap image, where each pixel represents a patch of the world and the pixel color defines an area to which the patch belongs. To load the scenario map into NetLogo, we used a built-in command `import-pcolors-rgb <file>`, which reads an image file, scales it to the same dimensions as the patch grid (while maintaining the original aspect ratio of the image), and transfers the resulting pixel colors to the patches. After we load the map into the NetLogo world, we were able to refer to the patches from a desired area by the patch/area color.

[12] http://www.cs.gmu.edu/~eclab/projects/mason/
[13] http://ccl.northwestern.edu/netlogo/

In MASON we had to create a text file with an environmental matrix, i.e. with numbers representing the areas of the scenario environment. We had to implement the loading of this environment into MASON's environmental structures. Environment in MASON can be 2D or 3D, and for both a variety of demo implementations is available. We chose 2D environment and started with `IntGrid2D`, which can hold a matrix of integers. After the implementation we found that the agents were moving too jerkily (jumping abruptly from one field to another) so we changed the environment into 2 layers, where the agents were moving in `Continuous2D` layer while the area definitions remained in `IntGrid2D`. While creating the continuous layer, we were able to define a discretization of the area which helped us to integrate the two layers. So in MASON the users can define multiple layers of continuous or discrete environments to represent their scenario environment. These layers (environment variables) need to be defined in the main class representing the simulation, which, in turn, has to be derived from the `SimState` class. Through the instance of this class the agents can access the current state of the environment. We have created a `Demo` class which extends `SimState` and consists of `people` variable (`Continuous2D` layer) holding the agent positions and `grid` variable (`IntGrid2D`) defining the physical environment.

GIS support. In recent releases, NetLogo was equipped with a GIS extension[14] for loading the vector GIS data (points, lines, and polygons) and raster GIS data (grids). We have tested it successfully on OpenStreetMap[15] data.

MASON did not have a GIS support for a long time. This has changed in the past few months and currently MASON supports the GeoMason[16] extension, which we intend to test in the near future.

Both NetLogo and MASON can satisfy the modeling needs regarding the physical environment. Now they both have a GIS support, which simplifies loading of the existing environments to these tools and integration with VBS2 training component.

3.2 Creating and Representing Agents

A world in NetLogo is made up of agents, where each agent can perform its own activity simultaneously with and independently of other agents. There are four types of agents in NetLogo: turtles, patches, links and the observer. Except the turtles, all the other agent types are static. We represented soldiers and civilians as turtle agents. We also represented stones as turtle agents, to easily simulate their throwing.

An agent in MASON is an instance of a Java class that implements `Steppable` interface, where the method `step(SimState state)`needs to be implemented, representing the agent behavior. This method represents one agent simulation step in the environment and is called by the scheduler. We have implemented 3 agent classes (types): Soldier, Civilian and Stone. Compared to NetLogo, in MASON we can implement each agent in a separate file/Java class, which provides for better organization of software code. Agent instances are created in the same way as any

[14] http://ccl.northwestern.edu/netlogo/docs/gis.html
[15] http://www.openstreetmap.org/
[16] http://cs.gmu.edu/~eclab/projects/mason/extensions/geomason/

Java class instance, and are then scheduled by the `SimState` simulation. Once scheduled, we can retrieve their reference (pointer) which we need in order to destroy the agent, e.g. when a Civilian is arrested and should disappear, or when a stone is thrown and no longer needed. We create the Civilians and Soldiers inside the Demo class. A stone agent is created when the Civilian enters the *stone picking area* and is destroyed when it hits the Soldier or (if it misses) after a few more simulation steps.

VBS2 agents can be created through script commands, ASI, VBS2Fusion, or through special tools like OME (Offline Mission Editor) and RTE (Real Time Editor).

3.3 Behavior Implementation

In NetLogo, an agent consists of a function describing its behavior and a number of attributes (agent variables), which describe the agent state. The agent behavior can be implemented in several ways. NetLogo code examples include a state machine implementation approach using a turtle variable and the RUN command. A state machine consists of a collection of states with a different action associated with each state. In our implementation of the scenario, we used a different approach. We have used turtles to represent the soldier and civilian agents and we also defined some specific variables for these kinds of agents. The behavior of our agents depends on the agent variables, which hold the state and motive variables defined in scenario. In each simulation step, we recalculate all the agent motive variables reflecting the actual state in the environment and choose the motive with the highest value as action leading. The action related to the action leading motive is then executed.

In MASON, the agent behavior is implemented and called via `step(SimState state)` method. The parameter `SimState` represents the simulation instance, holding also the defined properties of the environment and simulation.

The simplest behavior is that of the Stone agent. Stone agent is created when the Civilian enters the *stone picking area*. Then it is just carried by the Civilian agent along its path. Civilian and Soldier are another agent types implemented according to scenario from section 2.

Agent behavior in MASON is implemented through the `step()` method, which is invoked at each simulation step for the environment as well as for the agents and their internal components (fear, anger, etc.). The agent can access the environmental state via the SimState instance passed to the `step()` method. The agent can also invoke the `getObjectsWithinDistance` method on `Int2D` or `Continuos2D` environment properties to locate the appropriate objects depending on its intentions.

VBS2 agents are represented as Finite State Automata or Finite State Machines. Agents behavior can be implemented using an FSM editor, by scripting in a text editor, through Application Scripting Interface or, finally, by VBS2Fusion API.

Overall, we felt that both NetLogo and MASON had the needed support for the behavior modeling. In both cases, the behavior implementation had to be step-based, which differed from VBS2 and other virtual reality tools that were thread and event-based. This difference may have an impact on the integration and behavior implementation.

3.4 Movement Implementation

NetLogo offers a lot of built-in variables and commands, which make the implementation of the agent movement easy and straightforward. One can define location by `setxy <x> <y>` (e.g. its initial position in the environment)**,** by `set heading towards <agent>` to set the heading of civilian to nearest stone for example or by `forward <distance>` to move agent forward in the heading direction by specified distance**.** Another useful command that we used a lot is `distance <agent>`.

To the best of our knowledge, the movement algorithms are not supported well in MASON. All we could do in MASON was to set up a new location for the agent in each step. In NetLogo, movement is supported much better because of its turtle nature. So in MASON we had to implement the basic step-wise movement towards the target. The implementation of Flocking or Steering behavior (movement) is also not directly supported. However, Flocking is implemented in one of the MASON demos called Flockers. We will try to reuse it and test it. For flocking behavior in NetLogo, the programmer simply defines the closest distance among the agents and NetLogo steers the agents so that this distance is guaranteed.

Agent movement in VBS2 is planned via the A-star algorithm. VBS2 is able to plan the optimal path also using the waypoints.

Overall, NetLogo definitely has a better support for agent movement (at least heading towards is supported) than MASON. In MASON, a few sample implementations are available but not directly supported. In addition NetLogo offers built-in turtle commands for hill climbing and descending into valleys according to a variable value of patches around the turtle. There is also a support for "cone of vision" in NetLogo, which allows a turtle to set its viewport (vision angle and distance) and ask tor agents that fall in the cone.

3.5 Visualization

In NetLogo, vector shapes are used to visualize turtles. Vector shapes are built from basic geometric shapes (squares, circles, and lines) rather than from a grid of pixels. Vector shapes are fully scalable and rotatable. NetLogo caches bitmap images of vector shapes (magnified by a factor of 1, 1.5, and 2) so as to speed up execution.

NetLogo can be invoked and controlled by another program running on the Java Virtual Machine. It is possible to embed NetLogo models in a larger application. There is an API for this purpose, but it is considered as experimental and is likely going to change in the future releases of NetLogo. When running NetLogo models by API, it is possible to turn off the GUI.

In MASON, a very useful feature is the strict separation of visualization and simulation. In order to run the simulation with the visualization one has to create a new class derived from the `GUIState` class, which then instantiates the `SimState` implementation. For visualization layers one can use Portrayals, which usually match the variables representing the environment. One can define how their values will be mapped to colors or how to draw the agents. We have implemented only 2D visualization, but 3D is also possible and included in MASON demos.

VBS2 is used to show highly realistic 3D environments. There is a problem with smoothly visualizing atomic actions in special cases, e.g. when a civilian wants to throw a stone but the leading motive changes, so it starts turning back towards the safety area in the middle of a throwing action.

Overall, both MASON and NetLogo have equally good support for visualization, but MASON supports 3D for a longer time. In MASON, multiple displays can be used and models can be run fully independently of visualization. In both NetLogo and MASON one can switch off the visualization. But only in MASON the simulation models are truly independent from the visualization, which makes it much faster – an important factor for multi-parametric studies (data farming).

3.6 Parameterization

NetLogo offers a tool called BehaviorSpace, which can run one model many times, systematically varying the model's settings and recording the results of each model run. BehaviorSpace lets the user to explore the model's "space" of possible behaviors and determine which combinations of settings cause the behaviors of interest. User can parameterize a particular variable by specifying a list of all its possible values, by defining an initial value, final value and increment, or the variable can be randomly varied within a specified range.

Since MASON is built in Java, parameterization of simulation can be easily implemented. Direct support for parameterization of simulation is provided in the form of a tutorial[17].

Both systems support the parameterization needed for our multi-parametric studies (data farming). With MASON it is probably easier to achieve a massive run-time job-level parallelism. On top of that, MASON also performs well when running more instances on a single machine with more CPU cores, and has a strong separation of the visualization and the behavior model.

3.7 Model check pointing

When running a model with NetLogo GUI, it is possible to manually stop the simulation and save (export) its whole world state into a file. NetLogo automatically saves all the values of all the variables, both built-in and user-defined, including all the observer, turtle, and patch variables, the drawing, the contents of the output area (if it exists), the contents of any plots and the state of the random number generator. The resulting file can be then read back into NetLogo and simulation can continue from the saved state. This export/import functionality is provided by the built-in commands `export-world <file>` and `import-world <file>`.

MASON too has a good support for the model check-pointing – storing simulation at any time to a disk file. Later the model can be re-loaded and the simulation re-started from the same point. We have tested this feature and it worked well.

[17] http://www.cs.gmu.edu/~eclab/projects/mason/extensions/webtutorial1/

VBS2 game can be saved at any time and there is no problem in restarting it from several checkpoints made during the game to test alternative branches of the scenario.

Both NetLogo and MASON support the model check-pointing, but MASON also claims cross-platform compatibility.

3.8 Analytical Tools

Results of the NetLogo simulation can be displayed to the user in the form of a plot or a monitor. The first is the traditional way of displaying data in two or three-dimensional space. Monitor is another popular form consisting of a number of frames, each of which represents a concrete attribute of a simulation and its current numerical value. Users can export this data to a file in order to read and analyze it later with other applications, e.g. a spreadsheet. We have tried to visualize some state and motive variables of a civilian agent in plots (see charts on left side of Figure 1).

NetLogo Profiler extension helps measuring how many times the procedures in the model are called during a run, and how long each call takes. The profiler extension is new and experimental and is not yet well tested or user friendly. NetLogo System Dynamics Modeler is used to describe and understand how things in a model relate to one another. Instead of modeling behavior of individual agents and use them as the basic building block of a model, the populations of agents is described as a whole by differential equations.

MASON simulations can run directly as Java code without visualization. When running with visualization, simulations are controlled through the Mason Console (Figure 1, right) that allows starting, pausing and stopping. Users can load the stored models and run them from specific checkpoints. They can also record the simulation as a movie or take a screenshot. It is possible to set delays and choose one of multiple displays. Multiple displays are used when we need to have more than one view of the simulation. Similarly as in NetLogo, the users can inspect[18] all the public agent variables (but setter and getter methods need to be implemented). Their changes can be displayed as a Chart (JFreeChart extension) or streamed into a file.

VBS2 comes with the AAR (After Action Review) tool, which can be used for replaying and analyzing the whole mission to find crucial moments in the scenario.

Here, NetLogo was a traditional winner, but now MASON also has a good support for the analysis of variables evolving in time by streaming or drawing charts.

3.9 Logging

NetLogo uses the Log4j package for logging. NetLogo defines eight loggers (Globals, Greens, Code, Widgets, Buttons, Speed sliders, Turtles, Links), which are configured through a configuration file.

To the best of our knowledge, MASON does not support the logging functionality directly. We have implemented it using log4j. In each agent we have implemented the logging method, which receives the text label (usually describing actions) as input and

[18] http://www.cs.gmu.edu/~eclab/projects/mason/docs/tutorial0/index.html

outputs all the information about the agent – its location, variable states (fear, anger), motives and the text label. This provided us all the needed functionality for logging.

VBS2 has its own logging module, but there are also several script commands, which can be used for logging whatever else might be required.

NetLogo has a direct support for logging. In MASON one can use the existing Java libraries such as log4j to log the simulation data.

3.10 Performance

Performance of MASON was evaluated in [7, 8] and NetLogo in [7], where it turned out that MASON was the fastest platform. We have evaluated it by running our exemplary scenario with varying numbers of agents and extending the physical area so as to accommodate them properly. We achieved this by copying the same base scenario area 10, 100 or 1000 times by placing a new copy of the base area on top of each other. We have then tested the performance by running the simulation 10 times for 1000 steps. Since one base area accommodates 10 civilians and 5 soldiers, the evaluated numbers of agents were (1) 10 Civilians versus 5 Soldiers; (2) 100 Civilians versus 50 Soldiers; (3) 1,000 Civilians versus 500 Soldiers; and, finally, (4) 10,000 Civilians versus 5,000 Soldiers. In the last case we have run only 10 steps of the simulation for MASON. This step was not successful at all for NetLogo, because even with 1GB of Java heap space, NetLogo did not succeed in starting with 15,000 agents. Since NetLogo was much slower, we only run 10 steps for 1,500 agents.

In this way the systems were evaluated for up to 15,000 agents. This number did not include the stones, which were created and destroyed on demand. We have run the evaluation on the machine with two Intel(r) Core(TM) i7 CPU 860 2.80 GHz processors and 3GB RAM. The operating system was Windows 7 (32-bit version).

Number of Agents	15	150	1500	15000
NetLogo 1 step (ms)	0,48	27,60	18281,95	
MASON 1 step (ms)	0,10	0,59	21,51	2474,30
MASON speed vs. NetLogo	4,8 x	46,8 x	849,9 x	

Table 1: Performance evaluation summary

MASON and NetLogo performance is shown in Table 1. One simulation step took about 22 milliseconds for MASON and about 18 seconds for NetLogo for the middle option (No.3) with 1,500 agents. So MASON was almost 850 times faster. MASON speed is quite impressive and acceptable for real-time operation with virtual reality tools for about a thousand agents. NetLogo could be used well for a hundred of agents. While evaluating the performance we have switched off the logging for both MASON and NetLogo. With logging to file, the performance of MASON was 2-3 times slower. With logging both to file and to console the execution was 9-10 times slower. During the actual simulation the logging is needed, but the execution time of one step with 1,500 agents is still under 1/10 of second (about 66 milliseconds), which is still acceptable. For 15,000 agents, one simulation step took about 2.5 seconds for MASON (for NetLogo it did not even start), which is not acceptable for virtual reality trainings, but still acceptable for (off-line) Data Farming. All the

simulations were executed without GUI, but even with GUI the time of the simulation was still acceptable for 150 agents for both NetLogo and MASON. We did not measure and evaluate the exact time requirements of the simulations with GUI. In general, MASON is much faster than NetLogo. Additionally, we have tested the MASON performance on a single machine with four MASON instances running in parallel. Intel Core i7-720QM (4 cores) and 8GB RAM machine was used. One run of a single instance of MASON was 3.74 times faster than this parallel execution of four instances, which is a very good result. We did not perform this test for NetLogo.

In our test of VBS2, we have used the FSM combined with scripting implementations and the conclusion was that VBS2 could run 100 civilians and 20 soldiers with no delays at all (just in the initialization of the scenario there were some delays). We did not test VBS2Fusion, which suppose to be 200 times faster than ASI.

3.11 Standards

In this section we discuss related standards such as HLA or FIPA and their support in the evaluated platforms.

FIPA standards[19] are relevant mainly for mobile and intelligent autonomous agents and are not so much related to agent based simulation. FIPA covers agent communication, management and transportation (for mobile agents). For agent based simulation only agent communication can be relevant, but in simulations this is limited to a few concrete communication messages so it is not so crucial whether an ABS supports FIPA or not. Neither NetLogo nor MASON support FIPA standards.

DIS and HLAstandards[22] are more relevant for ABS, especially if we want to integrate realistic civilian simulation with soldier/police virtual training as intended in EUSAS project. VBS2 serious game supports both HLA and DIS. Anyhow, rather than HLA or DIS, we plan to use the plug-in functionality in VBS2 and CORBA[23] technology for real-time communication between ABS and VBS2 in EUSAS project, which would be easier to develop (e.g. no need to create a FOM - Federation Object Model). However since MASON is Java based, HLA based integration can be supported by using poRTIco[24] or Java port of CERTI[25] for example. NetLogo, integration through HLA would be also possible but not so straightforward.

3.12 Development Environment

In multi-agent systems developers face problems with debugging the agents since they run in separate threads. Both NetLogo and MASON[26] are step based, so models can be easily debugged as any procedural or object oriented program.

---

[19] http://fipa.org/specifications/

[22] http://www.sisostds.org/ProductsPublications/Standards/IEEEStandards.aspx

[23] http://www.corba.org/

[24] http://www.porticoproject.org/

[25] https://savannah.nongnu.org/projects/certi/

[26] In MASON, agent routine (step) is scheduled as an event, but there is only one event scheduled at one time.

NetLogo has its own development environment, which offers a lot of usable tools such as the source editor, interface builder or agent monitors. NetLogo environment allows users to run models and inspect their properties. Debugging can be done mainly by executing one step of simulation and watching how the agent variables change and how the visualization of the simulation changes. Developer can interact with the model by Command center on-the-fly, where it is possible to execute custom commands.

MASON is Java based library. Any Java IDE can be used to develop in MASON. We have used Eclipse[27]. There is also tutorial available on how to use MASON with Eclipse. Standard Java debugging procedures can be used easily to develop, debug and test MASON models.

Our experience is that simple well organized libraries such as MASON [9] are easier for programmers familiar with Java than more complex ABS IDEs, such as Repast Symphony [1].

4 Discussion and Conclusion

In this paper we have summarized literature surveys of ABS and evaluated two candidates – MASON and NetLogo by implementing exemplary human behavior scenario. Recently, there have emerged interesting new candidates, such as Repast Symphony or Janus[28] with its JaSIM[29] extension, which we might evaluate along these lines in the future.

Table 2 provides a summary of the evaluated features in MASON and NetLogo. Both are almost equal in many features. NetLogo is better in the physical movement support and some analytical tools. MASON is much faster, supports strong separation of visualization and behavior models, has a better support for 3D environment and is based on Java, which makes it far easier to integrate with other systems.

Features	NetLogo	MASON
Language	Logo, Java for simulation control	Java
Enviroment	2D, 3D experimental	2D, 3D
GIS support	Yes	Yes
Movement	Heading angle + step	just set(x,y)
Stearing/Flocking Behaviour	Not directly	Not directly
Visualization	2D, 2D as 3D	2D, 3D
run with no visualization	possible but not strictly separated	separated behaviour and visualization models
Parametrization	possible	possible
Model check-pointing	Yes	Yes, platform independent
Analytical Tools	Charts, Streamning, variable bars, snapshot	Charts, Streamning, snapshot, video recording
Logging	support using log4j	not direct support but log4j can be used
Performance	good for tens of agents	good for thouslands of agents

Table 2: Evaluated features summary

NetLogo has proved its reputation as an ABS platform where the simulation models can be implemented quickly and straightforwardly. A bit problematic is the development of complex models, which cannot be structured well – each source file is limited to include only one external source file. The integration with the serious game

[27] http://www.eclipse.org/
[28] http://www.janus-project.org/
[29] http://www.multiagent.fr/Jasim_Platform

component is difficult, because it would require developing a custom plug-in for NetLogo.

Regarding MASON, we have appreciated its rapid improvements over the past few years, with new plug-ins and tools (such as GIS support) continually being created. Its performance is impressive – it can support thousands of agents in one simulation. It is Java-based, which helps in its integration with the external systems (e.g. serious game component – VBS2). Similarly, the logging functionality can be implemented through other Java-based components, such as log4j.

Overall, we were greatly impressed by the NetLogo modeling support, functionality and the overall system, which makes it an extremely valuable tool for educational purposes, and for scientific model development and analysis. Had we simply looked for a handy standalone agent-based simulation tool for a limited number of agents, NetLogo easily could have been our choice. Regarding the specific goals and requirements of the EUSAS project, however, we had to conclude that MASON's speed, flexibility and extensibility were more important and made it the best-suited candidate for the job.

Acknowledgments. The paper was supported by EDA project A-0938-RT-GC EUSAS (European Urban Simulation for Asymmetric Scenarios) and Slovak Scientific Grant VEGA 2/0184/10.

References

1. Macal C., North M.: Tutorial on agent-based modelling and simulation, In Journal of Simulation, Vol. 4, No. 3, 151–162, 2010
2. EUSAS Consortium, Technical report, D2.5 Deliverable, Evaluation of existing simulation frameworks, 2010.
3. EUSAS Consortium, White Paper on Agent Modelling, Annex to D3.2 Deliverable: Documentation of the modelling requirements Behaviour Patterns, 2010.
4. Bakshy, E., & Wilensky, U. Turtle Histories and Alternate Universes; Exploratory Modeling with NetLogo and Mathematica. In M. J. North, C. M. Macal & D. L. Sallach (Eds.), Proceedings of the Agent 2007 Conference on Complex Interaction and Social Emergence (pp. 147-158).
5. Nigel Gilbert, Steven Bankes, Platforms and methods for agent-based modelling, doi: 10.1073/pnas.072079499, PNAS May 14, 2002 vol. 99 no. Suppl 3 7197-7198
6. Rob Allan, Survey of agent based modelling and simulation tools, 2010, Technical Report, DL-TR-2010-007, Science and Technology Facilities Council, ISSN 1362-0207
7. S.F. Railsback, S.L. Lytinen and S.K. Jackson Agent Based Simulation Platforms: Review and Development Recommendations Simulation 8:9 (2005) 609-23, http://www.humboldt.edu/ecomodel/documents/ABMPlatformReview.pdf
8. Russell K. Standish. 2008. Going Stupid with EcoLab. Simulation 84, 12 (December 2008), 611-618. DOI=10.1177/0037549708097146
9. Luke, S., Cioffi-Revilla, C., Panait, L., & Sullivan, K. (2004). MASON: A new multi-agent simulation toolkit. In Proceedings of the 2004; SwarmFest Workshop.
10. Cañamero, D. (1997). Modeling Motivations and Emotions as a Basis for Intelligent Behaviour. Proceedings of the First International Symposium on Autonomous Agents (Agents '97), pp. 148-155, Marina del Rey, February 1997, The ACM Press, New York
11. Dörner, D (1999): Bauplan für eine Seele. Rowohlt Verlag, Reinbek bei Hamburg.