The MASON Simulation Toolkit: Past, Present, and Future

Sean Luke[1], Robert Simon[1], Andrew Crooks[1], Haoliang Wang[1], Ermo Wei[1], David Freelan[1], Carmine Spagnuolo[2], Vittorio Scarano[2], Gennaro Cordasco[3], and Claudio Cioffi-Revilla[1]

[1] George Mason University, Washington, DC, USA
{sean, simon}@cs.gmu.edu
{hwang17, ewei, dfreelan, acrooks2, ccioffi}@gmu.edu
[2] Università degli Studi di Salerno, Salerno, Italy
cspagnuolo@unisa.it vitsca@dia.unisa.it
[3] Università degli Studi della Campania "Luigi Vanvitelli", Naples, Italy
gennaro.cordasco@unicampania.it

Abstract. MASON is a widely-used open-source agent-based simulation toolkit that has been in constant development since 2002. MASON's architecture was cutting-edge for its time, but advances in computer technology now offer new opportunities for the ABM community to scale models and apply new modeling techniques. We are extending MASON to provide these opportunities in response to community feedback. In this paper we discuss MASON, its history and design, and how we plan to improve and extend it over the next several years. Based on user feedback will add distributed simulation, distributed GIS, optimization and sensitivity analysis tools, external language and development environment support, statistics facilities, collaborative archives, and educational tools.

Keywords: Agent-Based Simulation · Open Source · Library

1 Introduction

MASON is an open source single-process simulation core and visualization toolkit in Java, designed to be used for a wide range of models, but with a special emphasis on agent-based models involving up to millions of agents. MASON has support for geographical information systems (GIS) and social networks, among other areas. Agent-based models (or ABMs) have taken hold not just in the sciences [17, 18, 10], but also in engineering areas such as distributed systems, swarm robotics, multiagent learning, and artificial life: for example, swarms of drones, driverless cars, air traffic control, and factory floor robots [11, 19, 16]. MASON was designed to serve both of these worlds.

Swarm-style multiagent simulation toolkits have developed along two lines. The first line to emerge were libraries geared for easy development of simple models. These toolkits usually were single-threaded, generally tied the model to its visualization and other facilities, and often deemphasized efficiency. Examples include SWARM [9], NetLogo [5], and perhaps early versions of Repast [7].

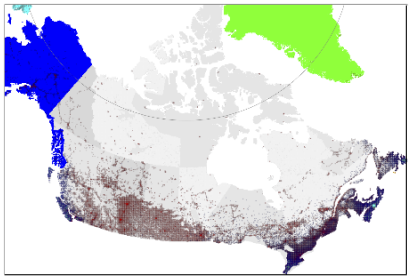

Fig. 1: Model of impact of climate change on Canadian communities [14].

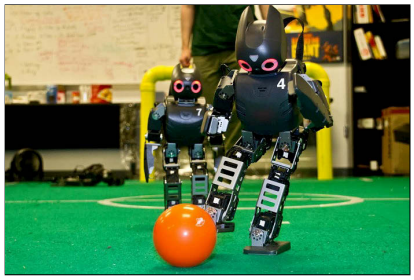

Fig. 2: MASON running on-board collaborative real time soccer robots [22].

The second line consisted of tools meant for large, complex simulations that might be run many times. These toolkits emphasized efficiency and extensibility more heavily, but were still generally single-process. MASON was among the first toolkits in this second line, and introduced many (for ABMs) unique features, including multithreaded models, separation of model and visualization, fully self-contained models, model serialization and migration, 3D visualization, and an orthogonal, consistent, and small design emphasizing efficiency. Other toolkits (for example Repast) have also advanced in many of these areas since then.

As computational cost decreases and models become more complex, we think that a new trend is emerging in ABM models: a third generation of simulation tools which give multiagent systems researchers access to high performance distributed simulation and the capabilities made possible by it: such as distributed GIS models and automated model validation and optimization. Some high-profile tools have made strides in some of these directions (such as FLAME [4] and Repast HPC [8], among others). At the same time, these third-generation toolkits must be clean, easily customized, and provide significant coding support. It would be desirable, though challenging, to marry these features with the traditional ease of use and accessibility afforded by some earlier systems.

Following the recommendations of a 2013 MASON community workshop [20], we are extending MASON to a full-featured third-generation toolkit. In this paper we discuss MASON's history, its architecture, where we think it needs improvement, and our plans for enhancing MASON over the next few years.

2 Development History

Though it has found use in the social science and computational biology ABM fields, MASON was originally meant for multirobotics and multiagent learning, and to evaluate evolved swarm behaviors produced by the ECJ evolutionary computation toolkit [3]. However after discussion with the GMU Center for Social Complexity, MASON's team realized that the modeling overlap between swarm robotics, AI, and ABMs in the social sciences was unusually high, and so decided to create a general-purpose library to serve the ABM community writ large.

Fig. 3: Modeling the spread of cholera in the Dadaab refugee camp complex in Kenya [15].

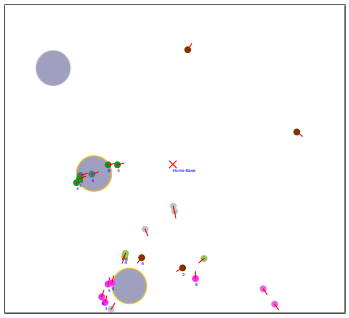

Fig. 4: Box-pulling model with 625 swarm robots [21].

MASON had several design goals from the very beginning. *First,* MASON was designed to have a small, high-performance, self-contained simulation core so that many models could be run in parallel, or could involve up to millions of agents. *Second,* MASON was designed to produce guaranteed identical results regardless of architecture when possible. *Third,* MASON was created with a Model-View-Controller (MVC) architecture with complete separation between the model and the visualization, and with model serialization. *Fourth*, as it came from the robotics community, MASON was meant to support a wide range of visualization facilities, including both 2D and 3D support. *Fifth,* MASON was designed to be very easily modified and extended.

These goals are hardly unusual in the general simulation community. However to our knowledge, among the major ABM toolkits at the time (such as SWARM, Repast, NetLogo and Ascape [1]) MASON's combination of design goals was original. MASON was released at Agent 2003, and we think it has had a significant impact on both the design and implementation of ABM tools since then.

Because of its emphasis on customization, efficiency, and generality, MASON has been used in a wide range of models, from small to very large, and in fields from robotics to the social sciences. As illustration, Figures 1–4 show four uses drawn from our own experience. For example, we have used MASON to build a 10-million agent model of permafrost thawing and its consequences on Canadian communities [14] (Figure 1); and we have also used MASON running on-board cooperative soccer-playing robots during RoboCup [22] (Figure 2).

3 MASON's Design

As a roughly MVC architecture, MASON is broken into two pieces, as shown in Figure 5. The first part is the *model* (the simulation proper) and the second part is the *visualization*. Unless one chooses to have model objects display themselves, the model and visualization are entirely separated, enabling model serialization and the removal or reinstatement of the visualization mid-run.

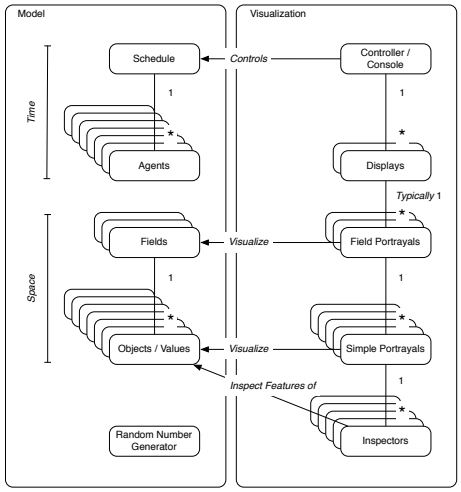

Fig. 5: Illustration of MASON's Top-Level Architecture

Model and Visualization A MASON simulation is encapsulated in a top-level model object which contains a simple real-valued time *schedule* on which are registered one or more *agents* to be called at some time in the future. Additionally, the model may hold one or more *fields* to represent space. MASON provides many fields, such as square or hex grids of objects or values, continuous space, and graphs or multigraphs. Many fields can be 2D or 3D; bounded, toroidal, or unbounded; and sparse or dense; and you can create your own as you see fit. Last but not least, MASON provides utilities for multithreading and a high-quality random number generator. Models can be fully serializable, self-contained, and capable of running side-by-side in multiple threads or in the same thread.

MASON provides 2D and 3D visualization tools, plus plug-in visualization facilities such as for GIS. Model visualization is encapsulated in a special top-level object. This contains a *controller* whose job is to start, stop, and otherwise manipulate the schedule. The most common controller is a window called a *console*. The controller also manages some number of windows called *displays* that handle 2D or 3D visualization. A display helps the user manipulate and visualize various fields by stacking together one or more *field portrayals*. A field portrayal often portrays individual objects or values in fields by calling forth a custom *simple portrayal* designed to visualize that particular object or value. Objects may choose to portray themselves as well. If the user selects portrayed objects with the mouse, a simple portrayal may create *inspectors* to provide object details, trace the objects through charts and graphs, and so on.

Utilities and Extensions MASON has many utilities to support model design. These include random number distributions, Java Bean Properties inspectors, GUI widgets, movie and picture generation, and chart creation. Several of MASON's utility objects have since found their way into other ABM toolkits (like NetLogo).

Finally, MASON is often extended to serve special functions. Foremost is *GeoMASON*, which adds high-quality GIS capabilities to MASON, including first-in-class raster and vector data model integration, standardized data and file formats, query algorithms, visualization, and integration with external GIS tools. GeoMASON sits atop the Java Topology Suite (JTS).

MASON is also integrated with ECJ [3], a popular, massively distributed evolutionary computation toolkit with which MASON was intended to dovetail. ECJ can be used to optimize ABM models in parallel, which are then assessed on-the-fly in MASON. MASON also has extension libraries for social networks and for 2D rigid body kinematics, among other tools.

Finally but critically, D-MASON [13] extends MASON into a distributed model toolkit intended to run over a large number of machines. When developing D-MASON, the University of Salerno chose MASON as its target platform largely because of its emphasis on ease of extensibility.

Coding Style MASON was written in Java because ECJ was in Java. This is not a controversial decision: Ascape, Repast, and NetLogo also target the Java Virtual Machine. Java has enabled MASON to be portable, provide replicable code, and be efficient. But MASON lacks certain hallmarks of modern Java, such as generics, annotations, and lambdas. Some of this is cargo-cult programming, but much of it is due to efficiency considerations. For example, MASON has a special replacement for `java.util.ArrayList`, because until recently ArrayList's *get(), set(),* and *add()* methods had flaws which prevented them from being inlined. HotSpot has since worked around these errors, and so going forward we may adopt ArrayList, along with similar workarounds obviated by recent Java improvements.

4 Where is MASON Going?

Taking a critical look at MASON, there are many opportunities for improvement. It is now over 15 years old, and was originally developed for Java 1.3, and so has warts and misfeatures stemming from its age. While reasonably high performance, it is also still fundamentally a non-distributed toolkit. And it is missing important functionality, such as good statistics tools, a testing facility, and so on. Going forward, we will be making many improvements to the system.

In 2013 we organized an NSF-sponsored workshop [20] that proposed nine community recommendations for how MASON could be enhanced to assist in cutting edge research in the future:

1. Give MASON better external language support and plugins for integrated development environments (IDEs).
2. Add advanced output and statistics architectures.
3. Add parallel and distributed facilities.
4. Upgrade MASON to reflect recent Java language changes.
5. Add a testing regimen.
6. Identify facilities to make MASON more useful to the science and technology education community.

7. Add collaborative archives for the MASON community to share models.
8. Add automated validation and parameter sweeping.
9. Improve MASON's GIS support and integrate it with distributed capabilities.

After the workshop, we applied for and received a three-year NSF grant (1727303) to improve MASON along these lines, and are now in its second year. Following the recommendations above, our plans to improve MASON fall into four areas. First, we are making MASON *more robust.* Second, we are building a *distributed* version of the software (including distributed GIS support) that runs over MPI on cloud computing platforms. Third, we are adding a variety of tools to make MASON more *friendly to coders.* Fourth, we intend to make MASON more *friendly to the ABM community* at large. We discuss these goals below.

4.1 Making MASON More Robust

Like much open source research software, MASON was built with few tests or automated quality control checks. Fortunately MASON has exhibited few serious bugs over the years, but it badly needs a testing harness. This is an interesting challenge for stochastic models because of the semi-random nature of the results generated. When results differ from expected results, is this because of a bug (or bug-fix) or is it due to the vagaries of the random number generator?

We are constructing a test harness that will run MASON through a battery of parameters and compare them with expected outputs. This will take advantage of MASON's replicability such that, given a fixed set of parameters and random number generator seeds, the outputs should be identical. We are building unit tests for MASON, and are devising stochastic distribution-based tests using common MASON models. These tests will be run many times over a large number of random number generator seeds. If changes cause distributions (mean, variance, etc.) to deviate significantly, this will raise a flag.

4.2 Making MASON Distributed

This task will consume the majority of our efforts over the next few years, and entails an integrated distributed model facility, distributed and improved GIS, and both large- and small-scale model optimization and validation. To support massively scaled agent-based models, we want MASON to take advantage of many compute cores by distributing agents and allowing them to run in parallel. The potentially high degree of coupling (due to arbitrary and dense agent-agent interaction) in ABM scenarios like social networks poses a challenge to distributed simulation as the interprocess communication can overwhelm the work done on individual cores. However many ABM models are distributed spatially, which can be straightforwardly distributed. We will target distributed spatial scenarios but aim to make it possible to distribute social networks, etc. when feasible.

We also recognize that distributing simulations presents a *brittle abstraction* to the model developer: to achieve increasing speedups, he generally must cater more and more to the specifics of the underlying distribution system. Many modelers will not wish to do this. Our goal will be to provide multiple API layers,

whereby a modeler can choose to make his model distributed with few changes to the original, or (if he wishes) delve deeper into the details of distributed simulation in order to achieve higher performance.

Approach Our work builds on D-MASON, which partitions the simulation space into regions, and assigns to agents in each region a *worker* to manage their scheduling, migration, and regional synchronization. A multicast channel (or *topic*) is assigned to each region, and workers subscribe to topics associated with the regions that overlap with their interest areas in order to receive relevant messages. Although D-MASON is an important first step towards a distributed MASON, it uses a simple space partitioning approach that is efficient for local communication, but inefficient for global communication.

The new distributed version of MASON maintains much of the publish-subscribe approach of D-MASON. Our current efforts are based on Parallel Discrete Event Simulation (PDES), a robust and scalable approach for distributed situation [12]. A PDES simulation consists of a distributed set of *logical processes* (LPs) executing in parallel. LPs generate events that often need to be processed by other LPs. The LP abstraction provides a clean and modular method for achieving scalable performance.

Two major components in MASON's distributed architecture are data sharing between LPs and load balancing. In terms of data sharing, in distributed MASON, the simulation field is partitioned into several axis-aligned (hyper)rectangular regions. Each LP holds one region and processes all the agents that are located in that region. The basic idea is to assign, when possible, each LP its own CPU core. For performance reasons we use peer-to-peer message passing via MPI.

For many models, and notably spatially distributed models, MASON agents most often need to access nearby data, known as their *area of interest* (AOI). To support quick access of data within the AOI, each LP not only stores the data in its own region, but also maintains a cache of some data from its neighbors, called the *halo area.* Part of the LP's own data is cached by its neighbor LPs. This cache is called the *shared area.* The sizes of these two areas are defined by AOI. After each simulation step, each LP will pull the data from its neighbors into its halo area and at the same time send the data in the shared area to its corresponding neighbors, in a process called *halo exchange.* Access of data outside an agent's AOI is supported via remote procedure calls (RPCs) between LPs. Data is provided in a synchronous fashion, meaning the caller will get the value once the RPC call returns.

Load balancing among active processors, where each LP tries to balance the workload among its immediate members, is critical for performance. This is done as follows. Each node measures its runtime every step, and when a node performs load-balancing, it collects the runtimes from its neighbors. Based on the runtimes, it chooses a neighbor and expands or shrinks its region such that the variance of runtimes among the node and its neighbors is minimized. For speed in making load-balancing decisions, we assume the runtime is linear in the size of the region.

Each partition adjustment can only shift the border by at most its own AOI to avoid additional data exchange between nodes, since each node already has part

of its neighbors' data in its halo area. This restriction might seem to slow down the load balancing, but we think that this is preferable because by limiting the adjustment and avoiding additional data exchange, the overhead is minimized and therefore the local load balancing can be done more frequently, better adapting to the change in workload. Another optimization is to avoid expensive coordination and synchronization activities by not allowing two neighboring LPs to perform load-balancing at the same time. To enforce this, we have implemented a graph coloring algorithm in the system so that at each step only the LPs with a designated color may balance their loads with neighbors.

Purely local load balancing runs the risk of getting caught in a local optimum. For this reason we plan to implement a hybrid local-global load balancing policy, using a hierarchy of LPs in a tree-structure, such as a K-D tree or a Quad Tree.

Distributed GIS Support As part of making MASON distributed, it is crucial to also make GeoMASON distributed. However GIS presents unique and difficult challenges. Geospatial data generally comes in *raster* (grid) or *vector* form. GeoMASON raster data maps straightforwardly to MASON grid data structures and so is easy to apply in a distributed fashion. But vector data can span large areas: for example, assuming we are distributing spatially, a river or a road might span our entire network of machines. We tackle this by breaking the vector data into three kinds. Point data can be easily distributed using standard MASON data structures. Most non-point data is static and immutable (roads, rivers): we can simply give copies of it to every single processor. Finally, *mutable non-point* data is typically static and ideally less common, so we may distribute it with the non-point data but embed each such object with a pointer to a secondary object located on a specific (distributed) machine where mutable information is held.

Distributed Optimization, Automated Model Validation, and Parameter Sweeping A major challenge faced by multiagent simulation is the complexity of validating models. MAS involves many heterogeneous variables, agents, agent behaviors, and interactions, and the model developer only knows the proper settings for some of these. To validate the model, one must optimize the remaining variables so as to match known ground truth, a laborious task. Model validation is often an optimization problem: the researcher hunts for parameter settings resulting in a model with low output error, and which is insensitive to certain parameters.

As a massively parallel stochastic optimization tool, ECJ is designed for exactly this task: the researcher defines known parameters and their values, then optimizes the remaining variables by running many models in parallel on back-end machines. MASON was designed from the ground-up to work with ECJ in this regard, but at present the two can be integrated only with considerable knowledge of both. We are working to make using MASON + ECJ for validation as simple as writing some code to assign a fitness to a simulation result, and then pressing a button on the MASON GUI. This is possible because MASON models are entirely self-contained and serializable. When a model is started, it is presented with a vector of parameters to use, and when it is done, it returns an *assessment* of the resulting run. ECJ will do the rest of the work.

Another common modeling task is sensitivity analysis. To address this, we will soon be releasing a parameter-sweep facility in MASON in which you can specify independent and dependent variables, then do parameter sweeps through the independent variables while logging dependent variable results. This is available in other tools (such as NetLogo): our goal is to initially permit parallelism and ultimately cloud distribution of the simulations during the sweep.

4.3 Making MASON More Coder-Friendly

IDE Support We are building MASON tools for Eclipse, and potentially NetBeans. First, we are adding *code templates* that allow users to generate code skeletons for common MASON patterns. The goal is to reduce the drudgery in dealing with MASON's high degree of boilerplate. Second, we are adding several *wizards* that walk the user through the process of creating a model, where he can choose from common model scenarios, parameters, and visualization options, and finally generate easily modifiable model code. We currently have this working in-house.

External Language Support A common request has been to provide some degree of external language support for MASON, particularly for languages which target the Java virtual machine (such as Jython, Scala, Clojure, and so on). This is not difficult given that these languages have Java function call support. But the primary difficulty is that many of these languages are slow in accessing Java data directly, as they would need to do when working with MASON. We can at least provide API support for the languages as a first step, then consider how we might encapsulate common ABM coding patterns in MASON utility functions so as to spend as much of the application runtime on the pure-Java side, where it is often much more efficient. We have proof-of-concept support for several languages.

Output and Statistics MASON can make charts and graphs and track variables in the GUI, but does not have library support to output statistics. We are remedying this. We think the best way for an experimenter to do this is to have MASON dump statistics into files designed to be directly entered into a tool such as *R* [6]. But we will also consider integrating well-vetted implementations of basic statistical analyses, such as descriptive statistics, difference tests, and confidence intervals, perhaps from a library such as Apache Commons Math [2].

4.4 Making MASON More Community-Friendly

Collaborative Archives and Facilities One major goal in MASON is to allow people to easily distribute and collaborate on public models. We presently offer several ways to do this, including a contributions section in the MASON repository. However, we want to go further. We hope to develop a special online repository to enable researchers to distribute models as `jar` files. We are exploring how to enhance MASON to advertise available models from this repository and enable users to download and run them (taking a cue from similar facilities such as in NetBeans). This would enable researchers to distribute models more easily, and also give educators and new users immediate access to a large and useful collection of educational demos and tutorials.

Education Aids and Examples Building on the collaborative archive and improved external language support, we hope to make MASON friendlier to science and engineering education by extending MASON's GUI to be more friendly to beginning users, creating a "simple" restricted Java API to MASON for students to use in lieu of the full API (for building simple models), and adding a significant number of new educational examples drawn from several disciplines.

4.5 Development Plan

Our three-year development plan is as follows. In all three years we will develop distributed MASON and distributed GIS facilities. In the first year we have also focused on automated model validation and GUI facilities. In the second year we will work on the test harness and unit and integration tests, as well as the collaborative archive. In the final year we will work on statistics utilities, external language support, and educational aids.

5 Conclusion

Since its introduction in 2003 MASON has proven to be a successful open-source agent-based modeling toolkit, with a particular emphasis on high performance, flexibility, and ease of customization. But MASON can be improved in many areas, including making it fully distributed, adding optimization and sensitivity analysis, and making the tool more friendly to newcomers. These plans are ambitious but achievable, and we hope that they will serve to make MASON a strong foundation for ABM development over the next decade.

A critical part of this project is ABM community involvement. We are forming a group of MASON power-users and critics to help us revise our approach, and we invite interested modelers and developers to participate in the effort.

6 Acknowledgments

MASON's development team has included Sean Luke, Gabriel Catalin Balan, Keith Sullivan, Liviu Panait, Haoliang Wang, Ermo Wei, David Freelan, Sean Paus, Daniel Kuebrich, Joey Harrison, Paul Wiegand, Maciej Latek, and Ankur Desai, with support from Claudio Cioffi-Revilla. MASON's rigid-body kinematics package was developed by Christian Thompson. GeoMASON was developed by Mark Coletti and Keith Sullivan. D-MASON was developed by Carmine Spagnuolo, Vittorio Scarano, and Gennaro Cordasco. Thanks also to James Olds, Dan Rogers, Ken De Jong.

GMU modelers who helped stress-test MASON include Bill Kennedy, Jeff Bassett, Brian Hrolenok, Atesmachew Hailegiorgis, Tim Gulden, Katherine Russell, Tony Bigbee, Mark Rouleau, Sarah Wise, Guillermo Calderón Meza, Lance Sherry, and Rob Axtell. Thanks also to our many international users and testers.

MASON has been supported by grants from NSF, DARPA, ONR, the US Army, and the Naval Research Laboratory; most via recently NSF Grant 1727303.

Bibliography

[1] Ascape agent-based modeling toolkit. http://ascape.sourceforge.net/
[2] Commons Math. http://commons.apache.org/proper/commons-math/
[3] ECJ metaheuristics library. http://cs.gmu.edu/∼eclab/projects/ecj/
[4] FLAME multiagent simulation tool. http://www.flame.ac.uk
[5] NetLogo simulation platform. http://ccl.northwestern.edu/netlogo/
[6] The R project for statistical computing. http://www.r-project.org/
[7] Repast agent-based modelling toolkit. https://repast.github.io/
[8] Repast HPC large-scale modeling platform. https://repast.github.io/
[9] SWARM agent-based simulation toolkit. http://www.swarm.org/
[10] Axtell, R.: Social science as computation. Tech. rep., Center for Economic and Social Dynamics, Brookings Institution, Washington, DC (2001)
[11] Bassett, J.K., De Jong, K.A.: Evolving behaviors for cooperating agents. In: International Symposium on Methodologies for Intelligent Systems. pp. 157–165. Springer (2000)
[12] Boukerche, A., Lu, K.: Optimized dynamic grid-based DDM protocol for large-scale distributed simulation systems. In: IEEE International Parallel and Distributed Processing Symposium (2005)
[13] Chiara, R.D., Mancuso, A., Mazzeo, D., Scarano, V., Spagnuolo, C.: Bringing together efficiency and effectiveness in distributed simulations: the experience with D-Mason. Simulation: Transactions of The Society for Modeling and Simulation International (2013)
[14] Cioffi-Revilla, C., Rogers, J.D., Schopf, P., Luke, S., Bassett, J., Hailegiorgis, A., Kennedy, W., Froncek, P., Mulkerin, M., Shaffer, M.: MASON NorthLands: A geospatial agent-based model of coupled human-artificial-natural systems in boreal and arctic regions. In: European Social Simulation Association (ESSA) (2015)
[15] Crooks, A.T., Hailegiorgis, A.B.: An agent-based modeling approach applied to the spread of cholera. Environmental Modelling and Software (62) (2014)
[16] Enright, J.J., Wurman, P.R.: Optimization and coordinated autonomy in mobile fulfillment systems. In: AAAI Workshop on Automated Action Planning for Autonomous Mobile Robots (2011)
[17] Epstein, J.M., Axtell, R.: Growing Artificial Societies: Social Science From the Bottom Up. MIT Press (1996)
[18] Gilbert, N., Troitzsch, K.G.: Simulation for the Social Scientist. Open University Press (1999)
[19] Parker, L.E.: The effect of heterogeneity in teams of 100+ mobile robots. In: Multi-Robot Systems Volume II: From Swarms to Intelligent Automata, pp. 205–215. Kluwer (2003)
[20] Payette, N., Bujorianu, M., Ropella, G., Cline, K., Schank, J., Miller, M., Jonsson, S., Gulyas, L., Legendi, R., Bochmann, O., de Sousa, L., Voudouris, V., Kiose, D., Szufel, P., Saul, S., McManus, J., Scarano, V., Cordasco, G., Hollander, C., Wiegand, P., Kazakova, V., Hrolenok, B., Rogers, J.D.,

Schader, M., Luke, S., Jong, K.D., Coletti, M., Schopf, P., Cioffi-Revilla, C., Sullivan, K., Talukder, K., Elmolla, A., Wei, E.: Future MASON Directions: community Recommendations (Report of the 2013 MASON NSF Workshop). Tech. Rep. GMU-CS-TR-2013-9, Department of Computer Science, George Mason University (2013)

[21] Sullivan, K., Luke, S.: Learning from demonstration with swarm hierarchies. In: Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS) (2012)

[22] Sullivan, K., Wei, E., Squires, B., Wicke, D., Luke, S.: Training heterogeneous teams of robots. In: Autonomous Robots and Multirobot Systems (ARMS) (2015)