MALSIM: Multi-Agent Learning Simulator and Benchmarking Tool

Drew Wicke

January 27, 2012

The multi-agent learning simulator (MALSIM) will be a simulator and benchmarking tool for multi-agent learning algorithms written in Java. The goal of MALSIM will be to provide multi-agent learning researchers a common framework in which to compare and test various MAL algorithms. MALSIM will help to eliminate some of the empirical testing issues that are prevalent in the multi-agent learning community. In order to create MALSIM, technical challenges will be encountered, various technologies used and a user-friendly GUI created.

In the multi-agent learning community, when testing a new algorithm empirically, the results are usually based on few trials and compared to few opposing algorithms. MALSIM will provide the multi-agent learning community the needed generic testbed for benchmarking and simulating new learning algorithms. MALSIM will have generic mechanisms to simulate, test and visualize experiments. Currently, there are a couple of simulators made for multi-agent systems; however, they do not provide a benchmarking mechanism [12, 6, 11]. Multiagent Learning Testbed (MALT) provides services similar to that of MALSIM [16]. MALT is extensible and offers a built in mechanism to graph and visualize the results of experiments. However, MALT is coded in MATLAB and is made for two player repeated matrix games. GAMUT, a benchmarking and game generating tool for game theoretic algorithms, is also similar to MALSIM, since many multi-agent learning algorithms are made to play game theory games [14]. GAMUT does not provide a GUI. Based on the usefulness of previously created tools, the need for a common, generic and extensible testbed for multi-agent learning algorithms is apparent.

In order to provide this tool, technical challenges will be encountered. These challenges include generic implementation and the inherent difficulties of parallel programming. The tool must be generic enough to provide the standard features of any simulated game and yet stay functional for all types of multi-agent learning algorithms. Challenges such as eliminating deadlock and race conditions and providing communication between threads will be addressed. Another challenge may be implementing multi-agent learning algorithms based on pseudo code descriptions of the algorithm.

MALSIM will be written in Java in order to be platform independent. Multi-agent learning algorithms will be implemented such as Fictitous Play [9], Hyper-

Q Learning [15], WoLF (Win or Learn Fast) [8] and ADL (Adaptive Dynamic Learner) [10]. GAMUT will also be integrated into the platform in order to easily generate matrix games. XStream, a tool that serializes Java objects to and from XML, will be used to provide a mechanism to save and load the tool's model [4]. Also, JChart2D or LiveGraph may be used to provide a way to visualize the results of the experiment [1, 3]. MPJ or JMPI, both pure Java ports of MPI, or mpiJava, a Java MPI wrapper, may be used to make MALSIM capable of running on a cluster computer [5, 13, 7].

A GUI as shown in figure 1 will be provided. The GUI will allow the user to set up the experiment by choosing the game to be played, the agents to be in the tournament, the mechanism to choose competitors and other properties. The GUI will also provide a list of current games being played and a way for users to pause them. The user will be able to save the settings of the experiment and, when the experiment is finished, the user will be able to view and save the results of the experiment. The GUI will also give the user the option to view and create graphs to compare and contrast the data from the experiment.

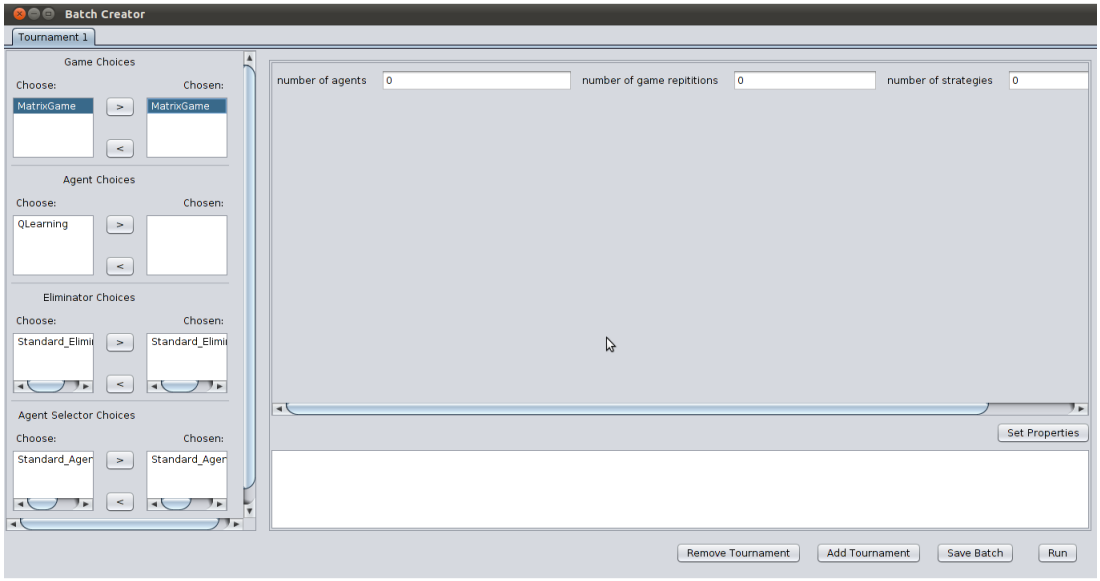

Figure 1: The MALSIM GUI for setting the parameters for the experiment.

References

[1] Jchart2d. *http://jchart2d.sourceforge.net/*.

[2] Jfreechart. *http://www.jfree.org/jfreechart/*.

[3] Livegraph. *http://www.live-graph.org/*.

[4] Xstream. *http://xstream.codehaus.org/*.

[5] M. Baker, B. Carpenter, G. Fox, S.H. Ko, and X. Li. mpijava: a java mpi interface. In *Proceeding of the International Workshop on Java for Parallel and Distributed Computing*, 1998.

[6] T. Balch. Teambots 2.0, 2000.

[7] M. Bornemann, R. van Nieuwpoort, and T. Kielmann. Mpj/ibis: a flexible and efficient message passing platform for java. *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, pages 217–224, 2005.

[8] M. Bowling and M. Veloso. Multiagent learning using a variable learning rate. *Artificial Intelligence*, 136(2):215–250, 2002.

[9] G.W. Brown. Iterative solution of games by fictitious play. *Activity analysis of production and allocation*, 13(1):374–376, 1951.

[10] A. Burkov and B. Chaib-draa. Multiagent learning in adaptive dynamic systems. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 41. ACM, 2007.

[11] C. JIANG, W. HAN, and Y. HU. Repast-a multi-agent simulation platform [j]. *Journal of System Simulation*, 8, 2006.

[12] S. Luke, C. Cioffi-Revilla, L. Panait, and K. Sullivan. Mason: A new multi-agent simulation toolkit. In *Proceedings of the 2004 SwarmFest Workshop*, volume 8. Citeseer, 2004.

[13] S. Morin, I. Koren, and C.M. Krishna. Jmpi: Implementing the message passing standard in java. In *Proc. Int. Parallel and Distributed Processing Symposium (IPDPS 2002), Ft. Lauderdale, FL*, 2002.

[14] E. Nudelman, J. Wortman, Y. Shoham, and K. Leyton-Brown. Run the gamut: A comprehensive approach to evaluating game-theoretic algorithms. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pages 880–887. IEEE Computer Society, 2004.

[15] G. Tesauro. Extending q-learning to general adaptive multi-agent systems. *Advances in Neural Information Processing Systems 16*, 16, 2004.

[16] E. Zawadzki, A. Lipson, and K. Leyton-Brown. Empirically evaluating multiagent learning algorithms. Technical report, Working Paper, November, 2008.