Adaptive Artificial Enemy for Embedded Simulation

Gregory Harrison, Eric Worden, Jason Smith, Jonathan Brant, Dave Maynard, Tom Wonneberger
Lockheed Martin Global Training and Logistics
Orlando, FL
gregory.a.harrison@lmco.com, eric.w.worden@lmco.com, jonathan.c.brant@lmco.com,
david.s.maynard@lmco.com, thomas.wonneberger@lmco.com

ABSTRACT

Embedded training, and many training exercises played against constructive entities can tend to be repetitive, with similar enemy actions occurring at similar times in the exercise. Evolutionary Algorithms with Genetic Programming (GP) and learning classifier systems (LCS) are seen as a means to provide adaptive enemies that not only process their actions using less rigorous behavior, but also adapt their behavior to the student being trained. This has wide ranging implications from learning a particular student's behavior, Course-Of-Action (COA) planning to After Action Reviews (AARs). Results of experiments in this technology will be presented, as well as indication on how to integrate this architecture with a training management system to provide long-term adaptation for improved student instruction. Additionally, the process of installing a machine learning system into a virtual simulator is discussed as it relates to this effort. New techniques had to be developed, particularly in the mapping of the motion commands to the paradigm of the machine learning technology being used, and especially with regards to the real-time update rate that is expected in the interface with a human participant.

ABOUT THE AUTHORS

Gregory A. Harrison is a Principal Investigator in Internal Research and Development at Lockheed Martin Global Training and Logistics (GTL), and a member of the Group Technical Staff. He has contributed to the Modeling and Simulation and Maritime business, and authored many papers and patents for intelligent software and hardware applications. He received the PhD in 1997 from University of Florida. He is a Graduate Faculty member of Florida Institute of Technology (FIT).

Eric W. Worden is a researcher and Software Systems Architect with Lockheed Martin GTL, with over 18 years of experience working in Decision Support and Logistics Systems supporting the USAF, USMC and US Navy. He also has experience in Modeling and Simulation with constructive simulation and scenario generation. Mr. Worden's expertise is in data mining, evolutionary algorithms, artificial intelligence, CAS, semantic web technologies and database systems.

Jason T. Smith graduated from the University of Central Florida (UCF) with a BS in Computer Science. He worked in the field of Modeling and Simulation at the Institute of Simulation and Training while attending college. Mr. Smith is currently a software engineer at Lockheed Martin GTL.

Jonathan Brant is a Software Engineer with Lockheed Martin Global Training and Logistics. He graduated from UCF with a BS in Computer Science and completing his MS in Computer Science with a concentration in artificial intelligence including evolutionary computation.

David S. Maynard is a Software Engineering Lead with Lockheed Martin GTL within Virtual World Labs. He received his BS in Computer Science from Florida State University and is working toward an MS in Computer Science from FIT with an emphasis in genetic algorithms. He has had more than 16 years in the Open-Source / Commercial Video Game Industry's community with specialties in Opposing Force Bot Programming, Massively Multiplayer Online Architectures (MMOA) & Virtual Worlds.

Thomas R. Wonneberger is a Software Engineer with Lockheed Martin GTL. He graduated from Rensselaer Polytechnic Institute with a BS in Electrical Engineering. Prior to Lockheed, he worked as a Test and Automation Engineer on the iSeries servers at IBM.

Adaptive Artificial Enemy for Embedded Simulation

Gregory Harrison, Eric Worden, Jason Smith, Jonathan Brant, Dave Maynard, Tom Wonneberger
Lockheed Martin Simulation, Training & Support
Orlando, FL
gregory.a.harrison@lmco.com, eric.w.worden@lmco.com, david.s.maynard@lmco.com,
jason.tr.smith@lmco.com, thomas.wonneberger@lmco.com, jonathan.c.brant@lmco.com

CONCEPT

Embedded training, and other virtual simulations, can benefit from having a larger variety of scenarios available, so that the user will not have to repeat the same training each time. Ideally, the training will also help work on developing aspects of the subject matter that the student may need more work on. Using adaptive agents as the adversary in training that involves interaction with the enemy can help to provide a training experience that gets more difficult or more focused on the student's needs by having the enemy adapt to better fight against the student.

The approach taken here is to automatically develop enemy behavioral programs using genetic evolutionary techniques, and to execute these programs in real-time using a framework that allows interaction with the environment such that the machine learning system is given the correct cues to adapt as needed. Not only does this involve having an agent framework, but also the environment needs some changes to provide the cues to the agent system. To accomplish this, a Complex Adaptive System (CAS) environment was created as an overlay on the virtual environment. It would supply punishments and rewards that allow the agent to get meaning from events that occur in the environment. In addition to the CAS-generated reinforcements, a coach-type fitness function is programmed to reward the agent for correct doctrine, and steps in the right direction toward accomplishing the mission.

The agent learning system that was incorporated in this effort is a Learning Classifier System (LCS) (Holland, 1995) constructed using Genetic Programming (GP) (Koza, 1994). An LCS is generally considered to use a binary representation for its genetic system, in other words being programmed with Genetic Algorithms (GA). Instead, we chose GP, since it allows for direct development of software programs. The chromosome is made of software functions and argument or terminals, and does not require interpretation as the binary GA does. Also the GP chromosome can have various lengths, and is not made to be some fixed length, like most GAs are. The LCS we chose is a John Holland type of LCS with a message board. It is not the XCS (special Classifier System) (Wilson 1995) variety, since XCS did not allow direct incorporation of message lists, making interaction with the environment harder, and not supporting the evolution of a program chain linked together by message communication. The use of a message list allows small efficient programs to be evolved to carry out tasks. There is no need to maintain a large amount of individuals, as in XCS, when the LCS has already determined the best rule list (BRL) using the shortest amount of code possible. Having an identifiable BRL also allows the learned concepts to be persisted, as will be described later, forming a memory system, and allowing techniques that were evolved to fight against a certain student to be saved in a memory for that particular student. The combined system is called an LCSGP, for Learning Classifier System, Genetically Programmed.

One of the goals of this type of research is to investigate the emergent properties of intelligent agents that operate in a battlespace represented as CAS. The agents are allowed to learn and optimize their behaviors with regards to the architecture of the CAS, the mission at hand i.e. the scenario, the rules of warfare, and the behaviors of the adversary. In some of the current research efforts, scenarios were simulated in a three dimensional battlespace, with terrain that caused the development of different emergent techniques in the agents. This can be seen in Figure 1. Two agents were used: the blue agent and the red agent, and they learned their behaviors and missions separately, but with regards to the behaviors of the other agent.

The results provided an adaptive course of action (COA) plan for the agents. A COA would define the plan for the agent to accomplish. But plans generally change upon contact with the enemy, and thus the system instead develops an action plan with contingencies and mission-restoring capabilities built in, due to the default hierarchies developed automatically by LCS learning. The agent can learn what to do in the general case, but in special cases it can do something else if it senses a slight difference from general operation.

Figure 1. The agents, the center of gravity (COG), blue's home base and the terrain can be seen in this depiction of the battle space rendered in jME.

Training with respect to enemy action allows various sets of rules to be developed that respond to emergent or adaptive behavior by the adversary or to changes in the environment. Interoperability with a human agent is a goal of this research, to provide adversaries that adapt to the behavior of the human participant. This allows enhanced Train-to-Fight, Fight-to-Win capabilities, as the training becomes more like an actual battle, with the enemy changing tactics based upon the tactics of the friendly forces.

The RAND Corporation has also developed a Genetic Algorithm-based system that evolved red force and blue force paths, where blue had to travel to a goal and red would influence the evolved path (Pernin, 2008). This work developed paths based upon milestones, to provide a selection of paths for blue to follow to travel between two points in a battlespace. Our research into this had already begun at the time of the release of the RAND report, having started with the development of intelligent agents for Joint Strike Fighter worldwide logistics, beginning in 1997. The RAND Corp. work and this work are similar in many respects in that both are intended to provide paths through an environment with response to an adversary and both used genetic evolution techniques to develop improved paths. In contrast, our current work develops a set of rules to follow for the agent, not a set of milestones for a path. Our work uses CAS and LCS technologies to adapt the path and behaviors based upon the current state of the environment instead of providing a fixed path. The destination milestones are still embodied in the agents as goals, but the paths now may change in real-time due to adaptations occurring in the CAS. Inter-agent communication is enabled with message board functionality for future combined force simulation. And ideally, training could be conducted using current Situational Analysis (SA) information.

CHALLENGE

The challenge is multifold in this effort, and includes adapting the intelligent agent learning system to operate in real-time with agents and to interface with a full 3D environment where the battles will take place, using CAS technology and representation to guide agent learning. The 3D virtual environment must be modified to accept instructions from the intelligent agent and to provide situational awareness (SA) to the individual agents. The environment must also provide the rewards, punishments, and the simulation of the physical environment, including the effects of terrain and distance that the agents operate in to learn their behaviors. Then the challenge remains to have the intelligent agents learn to operate in this physical environment and learn to accomplish their missions. The missions are not programmed into the agents, but rather the rewards and punishments are provided and the agents, using machine learning, learn to obtain all the major rewards and minimize their punishments, and thus accomplish the mission.

Interoperability with a human agent is an important challenge being explored that is intended to allow

benefits both to the agent behavior rule sets and the human participant. The agent rulesets get an opportunity to evolve against a human adversary possessing their own ideas of how to conduct the mission. And the human adversary may develop new skills and help to break developed habits as the software agents learn to exploit them. Note, the human participant will not receive rewards and punishments as they use the system, but their performance will be graded, thus providing a form of automated After Action Review for the human participant(s).

As the human participant trains in a mission against the adaptive enemy agent, the adaptive enemy agent may use simulation to evolve better techniques to fight the human. Spawning multiple simulations of the battle that execute in faster than real time, and allowing numerous virtual agents to adapt new techniques against a simulated human adversary, can lead to novel, computer-generated, counterattacks and courses of action chosen to take advantage of that particular human participant. The knowledge that is gained by the agent system can be stored for future use in the training program for that person.

COMPONENTS

The components involved in accomplishing the adaptive enemy concept include the CAS environment, the intelligent agents with their architecture and operational theory, the simulator, the training of the agents and how the agents have been integrated into simulations. One aspect of the development of the embedded agents that became important to include is the ability to persist the memories of how to perform the tasks that the agents have learned, and potentially to tie these memories to particular students that are training against these agents. These aspects are described in the following sections.

Complex Adaptive System

Complex Adaptive Systems (CAS) are systems that contain many components that interact and adapt both as the result of external agency or from the occurrence of internal events. These may include natural systems such as lakes, the atmosphere, and human interaction, or systems that people have designed, such as the stock market, a battle plan, or a commercial enterprise in the marketplace. These systems have some characteristics in common including emergent properties and reward/punishment mechanisms. These properties are meant to be sensed by agents within the system. Some other CAS properties include nonlinearity, adaptivity, and modularity (Yang, 2008).

Emergent properties in a CAS arise from the complex interconnections, placements of the system components, and the system architecture. Patterns of operation arise as a result of attempting to perform missions within the system. Some patterns will work, and some will be prevented from occurring. These emergent patterns exercise the overall architecture of the CAS as it appears to the agents. Architecture, in this case, refers to the layout of components in the CAS and how they are allowed to work together, such as with the use of physics to provide constraints. Different patterns will emerge for different systems. For instance, the daily routines of a high-rise apartment dweller will be different than the routines of people who live on a farm. These emergent behaviors allow the individual agents to carry out their intended tasks and obtain a system-wide goal. Emergent behavior is more than just what each agent does but something that is a global result of the behaviors of all the agents in interaction with the environment.

The reward and punishment mechanisms included in the CAS allow the agents to learn from their behaviors, such that those behaviors that were rewarded tend to be repeated, and those behaviors that were punished tend to be stopped. By virtue of the schemata concept in genetic algorithms, behaviors are encoded as organizations of genetically-evolved rule data through which various schematas or building blocks of behavior and genetic characteristics are contained (Holland, 1996). The better schematas reproduce throughout the populations of rules and agents in the system, being led by fitness pressure from the rewards and punishments. Rewards can be exogenous, in that they are directly provided from the environment, or endogenous, in that they are internal, feel-good, rewards that the agent supplies itself for having done the correct thing. An example exogenous reward would be where the price of a purchased stock increases greatly after being bought. An endogenous reward would be such as when someone's health improves after starting an exercise program. These rewards form the basis for adaptation in a CAS by agents that have the ability to learn.

The CAS system in this research uses different environments, including a 3D representation of the environment where agent intervisibility is affected by terrain. This is augmented by an overall monitoring system that determines if rewards are due to the agent for behaviors in that environment. The CAS does not have to be 3D; it may be a networked architecture, or a conceptual architecture, such as the stock market. It is typified by a system of suitable complexity with many interacting and adapting parts, such that as a whole it cannot be described in a closed mathematical sense,

and many behaviors that arise within may defy prediction. Thus a CAS simulation provides a uniquely appropriate means to find out how the mix of components within may be expected to behave. Multiple thresholds exist or emerge where the surpassing of some conceptual or numerical threshold can result in very different behavior. One of the benefits to analyzing systems using CAS simulation concepts is to be able to identify these thresholds. The challenge thus becomes to adapt simulations of various systems to use CAS technology and to exercise these simulations to learn the characteristics of the systems under different influences, and discover the threshold tipping points. More information on the CAS theory and its use in system analysis may be found in (Miller, 2007) and from the (Santa Fe Institute, 2009).

Intelligent Agents

The term agent can be applied to any entity, biological, mechanical or otherwise, that can perform actions, intelligent or not. In terms of CAS, we want an agent that can perceive its environment through its detectors and take action on its perception through its effectors (Holland, 1995). The agent is "intelligent" because it learns how to interact with its environment by exploration, adapt, and then take action based on the information it's gathered (Russell, 2003). Learning is accomplished much like early human learning: through exploration and constant reinforcement.

Each agent contains a population of rules and memory, thereby allowing the agent to maintain its respective knowledge. Each rule in an agent is termed an individual for purposes of genetic evolution of the agent. The agents have both an internal and external memory. Internal memory allows the agent to store innate knowledge like goals and agent state, while external memory allows the agent to respond to things that it has sensed before in the environment. The top level architecture of the agent is shown in Figure 2. The agent perceives information from the environment through its sensors. It then processes the data through its knowledge and then takes the appropriate actions through its effectors.

Figure 3 shows the architecture of an LCSGP rule within the agent, with each rule having four parts to it. Each section, the internal and external antecedents and consequents are evolved similarly to an Automatically Defined Function (ADF), as defined by (Koza, 1994), in that they may each have different functions and terminals pertinent to its operation, but the four ADF's we use are constrained to be parts of an if-then rule, serving as sections of the LCS architecture for the individual rules. The External Antecedent looks for any external messages which pertain to its task as well as taking status of the environment as needed. The Internal Antecedent looks for any internal messages and checks internal memory as needed. Each rule has to be able to purchase at least one message and sell at least one message in order to be a viable rule allowed to be part of the population. The Internal Consequent performs any internal actions such as posting internal messages that other rules can buy (bucket brigade style) and adjusting internal memories. The External Consequent performs any external actions such as Move, Shoot, Destroy, etc.

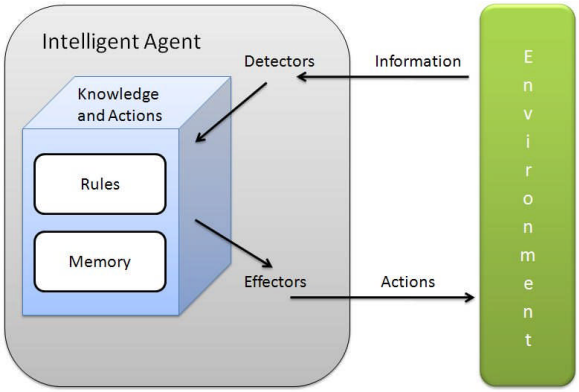


Figure 2. The top-level agent architecture indicates internal processing and an interface to the environment.

The set of rules in the LCS is also the population of individuals that are used in the evolutionary architecture. The fitness of these rules is based upon how much money they have at the time that the GP runs. The rule individuals have money that they use to bid on and buy messages, and they make money when a message they have posted is bought or when they receive a reward for an action that they have accomplished. A rule that fired when a reward was received may have been dependent upon many other rules that had fired beforehand, and these earlier rules are rewarded by fitness passing in a bucket-brigade manner through the buying and selling of messages on the message boards. Many repetitions of the exercise in the simulator are needed to determine the balance of money in the rules, so that the fitness of rules that do not contribute can decrease through taxes and bid costs, while productive rules increase their net worth and can thus be identified. This is the reason to use a simulation that allows many repetitions to be performed to test a given rule set. Periodically, the rule set is then operated upon by Genetic Evolutionary principles to apply fitness pressure to evolve the rule set to obtain steadily increasing overall fitness results in performing the mission. More information about the

learning system used in the intelligent agents can be found in (Harrison and Worden, 2007).

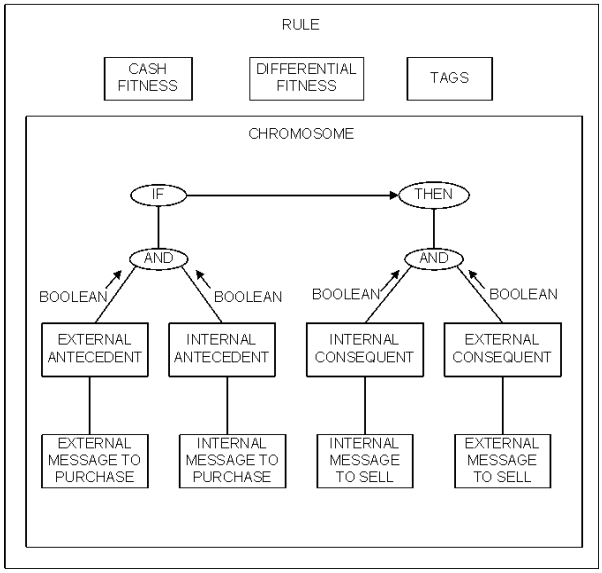


Figure 3. The antecedents and consequents of an individual rule contain messages to buy and sell in the agent artificial economy, and also contain functions that operate in the CAS environment.

The agent architecture can be split into two sections; the intelligent learning system and its avatar or character in the simulation, akin to the brain and the body. Thus a cyberspace instantiation of the system is permitted where the agent intelligence is located on a remote computer, or set of computers, while the actions it takes in the end environment are separated by interfaces that may be controlled through a network. We are currently using the Internet Inter-Orb Protocol (IIOP) to connect the agent brain to various bodies on the simulator, with visual entities that participate in a potentially distributed simulation, including standards such as Distributed Interactive Simulation (DIS). Other architectural representations may use mobile intelligent agents that transfer their entire software representation to the remote computer that is conducting the simulation. This option relies upon an agent docking standard to interface the itinerant agent to the internal simulation at that machine and its artificial economy. This system has used the IBM Aglet architecture (Lange, 1998) in the past, to support agent mobility, but the interface mechanism of the new system has been found to provide a suitable technology for the control of avatars in a virtual world type of simulation.

The agents pass their intelligence onto other agents using evolutionary algorithms and rule migration. Evolution takes place on an agent-by-agent basis, each agent evolving separately. Through evolution, genetic material is transferred between agents with generally only the fittest agent genes being able to evolve. The algorithm cycles between elitist and diverse over the generations.

Example Scenario

In order to explore the use of the CAS rewards to influence the learning of behaviors in the intelligent agents, and to test the learning capabilities of the agents, a simple scenario was designed that included two agents on opposing forces, a targeted destruction mission, and a 3D physics-based environment in which to operate. Thus, the agents could be placed in an appropriately CAS-enabled simulator, and allowed to evolve under fitness control to learn the missions.

In this example scenario the blue agent's goal is to destroy a Center of Gravity (COG) belonging to the red agent and ultimately return to its starting position i.e. home. These locations can be seen on the image in Figure 1. The concept of Center of Gravity goes back to (Clauswitz, 1830) and refers to a source of power or security for a group of people, such as a well for a village. The blue agent is supposed to destroy a COG belonging to the red agent and return home. The red agent is supposed to protect its COG. Both agents may use whatever terrain and environment features discovered throughout the learning process that best benefits them. Both agents also have the capability to attack each other as a means to accomplish their mission.

Learning

The LCSGP applies fitness pressure defined by the job that the agent is learning. In our example, each agent has a main task, which, for the blue agent, is to destroy red's COG, and return home safely. Red's main task, however, is to protect its COG by killing the blue agent. Given this, their main goals are in direct opposition to each other, meaning their fitness (or lack thereof) is based not only their own actions, but also the actions of the opposing agent.

Each job has a fitness scale defined for it that will promote positive emergent behavior as well as discouraging negative emergent behavior. An example of this is that the current blue job is awarded for shooting red, but only half as much as it is deducted for getting shot, which means if blue were to shoot it out with red, it will lose, and because its main objective is not to kill red (although, it does get rewarded for doing so), but instead to destroy the COG and to get home

without dying, the blue agent will, based on fitness pressure, avoid a direct shoot out with red, and instead focus more on its own main objectives. The learning can be stopped when the fitness values stop changing; having reached a stasis for the current environment and agent, at which point the best rule list may be saved for re-use. Alternatively, the system can run continuously, with agents adapting to changes in the environment and the tactics of other agents by sensing the CAS environment and evolving its rule set to optimize its response to the changed situation.

The fact that an agent's fitness is not only based on its own actions, but also its opponent's actions means, that while blue may be able to completely learn its task quickly without the red agent, once the red agent (or a human) is introduced and is learning from blue, blue will have to constantly evolve better and better rules to be able to be able to still complete its job. At some point, a balance will arise where both agents have come as far as they can go with their starting conditions, and changing these will invariably tip the scale for or against them due to CAS thresholding behavior.

Knowledge Persistence

Throughout the learning process, a best rule list (BRL) is constructed and continuously modified as new, more advantageous rules are discovered. Different BRLs apply to different tasks. Differential fitness is a term that describes how much money an agent earned running a given job. Following simulation execution, the BRL and its associated differential fitness are written to non-volatile memory, which could be a file or a relational database. The BRL can then be loaded back into volatile memory before running another simulation at a later date. Such a persistence model is consistent with the way in which humans store experiential information for later use (Baddeley, 1997). The knowledge persistence mechanism may aid or supplement the reinforcement learning process as the agent would not be starting from a "clean slate." They would already have a set of known good rules that were evolved via a system of rewards/punishments to start from.

Implementing knowledge persistence allows for greater operator flexibility in that, if the simulation is terminated prior to the system reaching stasis, it can be resumed in the future with the previous state of the BRL intact. As previously noted, this also facilitates the knowledge reinforcement process required for continuous evolution.

Persistence is particularly advantageous in evaluating the effectiveness of different jobs based on their comparative differential fitness. That is, a BRL from one simulation could be loaded into another job with differing simulation parameters, thus facilitating evaluation of the BRL under varying environmental stimuli. By seeding multiple simulations with the same BRL but different environmental parameters, it is possible to establish the "global" applicability of that BRL. Over time, one could potentially determine an optimal BRL "seed" that exhibited historical precedence for producing high fitness BRL's for a finite number of genetic evolutions and under any given parametric configuration.

SIMULATION ENVIRONMENT

The initial simulation environment used in rendering the CAS was the jMonkey Engine (jME) due to its Java implementation and completeness of the engine. jME is a java based game engine created by Mark Powell in 2003. Figure 1 is based on a screen shot of the agents in the jME environment.

To integrate the agents into the jME simulation environment, all agents were given an internal memory and a means of polling and interacting with the environment. An agent's internal memory would then store information it noticed about the environment the last time it checked. This included whether or not the agent could see other agents, what direction the other agents were in last time it looked, and what objectives the agent has currently completed.

The agents were instantiated and inserted into the same environment for learning behaviors as outlined in the example scenario section. They were controlled as outlined in Figure 4. Multiple threads were started, and each agent ran in its own thread. The main game ran in a third thread. The main thread spawned off the two agent threads. The battlespace shown in Figure 1 was rendered, and learning was started by the agents in their individual threads. They executed their LCSGP state machines, buying and selling messages internally and to the environment, and kept practicing their missions over and over, continually striving to improve their behavior. When started from tabula rasa, a clean slate, the agent rules were completely random, and training could take many hours. The persistence of agent memory should help alleviate this issue.

The agent rule lists were somewhat more tightly coupled to the environment than was suitable, and attempts to store off the knowledge tree for the agent's memory proved difficult, since serializing the large tree of aggregated objects that represented the LCSGP

population or BRL also required serializing the entire virtual environment. The agent system has since been re-architected to force a separation of the brain and the body, through Java Remote Method Invocation (RMI) IIOP, and now the brain can run on a different machine than the body, and does not require serializing the simulation environment in order to persist the best rule list.

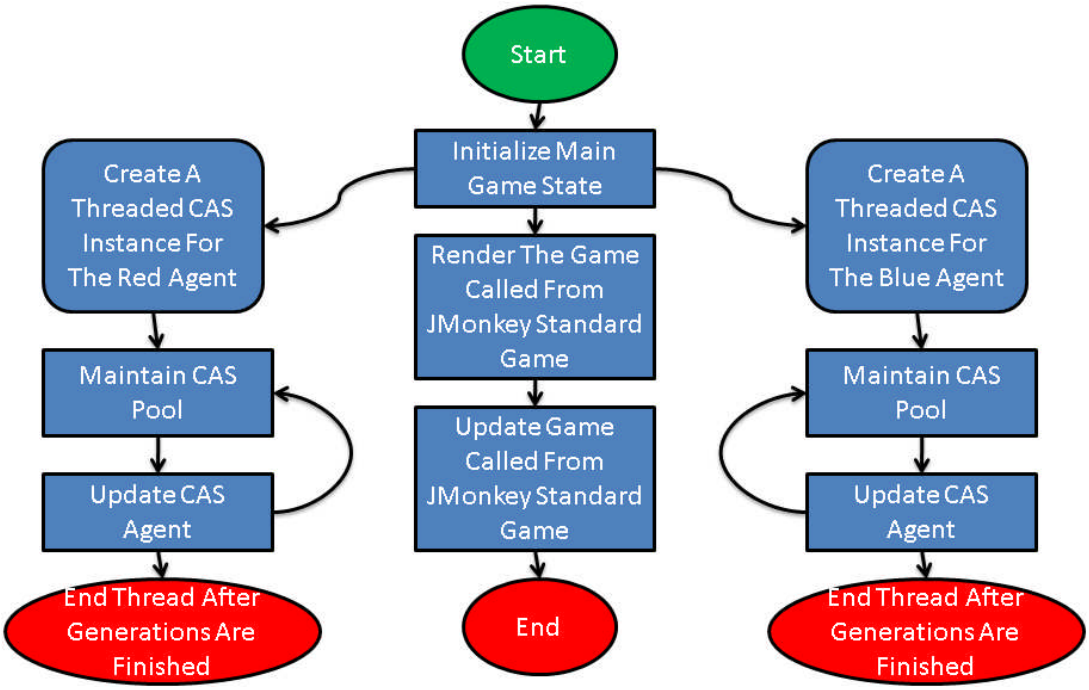


Figure 4. The parallel execution threads for the two competing agents and the joint real-time environment are seen here for two learning agents.

After the initial 3D designs in jME, the decision was made to move the target simulation environment for the intelligent agents to the Lockheed Martin Prepar3D virtual environment. In the architecture developed to interact with this environment, agents are added to and information is received from the simulation using the SimConnect API built into Prepar3D. This allows us to have the brain of the agent running on another computer constantly searching for an answer. The communication to the IIOP agent body server comes from the brain IIOP skeleton which interfaces to the LCSGP brain. This is replicated in reverse to allow the CAS to send message to the brain. This simulator retargeting also allows the agents to interact with a DIS environment.

Integrating the agents into the simulation environment required components to be added to the simulation environment as well as the agent. To create a complex adaptive system environment we ensured that the simulation environment must give rewards or some form of intelligent stimulus back to the agent to permit the machine learning algorithms to gain environmental knowledge. To accomplish this goal, an agent coach was created that can monitor the agents and their interactions with the environment. The coach has the ability to send rewards, based on what it sees, back to the learning system using IIOP and an agent's unique identifier. The coach can be watching the environment to see when agents have completed parts of their task, like killing a target, or their entire task. Simulation systems may provide built in mission system capabilities that can be leveraged and extended for to accomplish this task.

REAL-TIME OPERATION

Prior applications of the LCSGP involved non-real-time operations that operated at speeds governed by the time durations needed to complete different functions. In this case, to operate in a real-time simulation or against a human participant, it was necessary to constrain the evaluation of LCSGP rule sets and their interaction with the CAS environment to take place at a real-time rate with repetitive periodic processing. The repetitive time period may be termed a tic, a frame, a

dt, or an epoch. The simulation ran at a full real-time visual rate, while a slower internal update rate for the learning system was used, commensurate with a fast human cognitive response duration.

Movement in the 3D environment is controlled by the agent through move commands. The move command employed in this system takes two arguments, a direction and a speed. This command serves as a function that is issued by the evolved agent rule set. The agent may evolve to issue movement commands every epoch, or only rarely, for scouting or as part of a defense mechanism. In the LCSGP system, it is possible for multiple rules to fire in a single epoch, and in that case, the combined movement commands are averaged together, and any resulting reward is also sent to the rules in proportion as to how well they caused the agent to move toward the goal.

Combat within the environment is accomplished with a ranged shoot command that offers both an offensive and defensive capability. The agents are started with a limited supply of ammo, and will only register a hit within 50 meters. Because of this, and the fitness pressure applied by the jobs for combat, agents may evolve to be more combatant, or more pacifistic. For example, the red agent is rewarded more heavily than blue for shooting his opponent, so he may learn to become more aggressive. However, this kind of behavior could also emerge if he starts with more ammo. Given the nature of the two opposing missions, the amount of ammo each agent starts with could be an important threshold as mentioned previously.

With the use of the external message list, sensed situations are reflected as messages posted on the message list. When a situation message is recognized by an existing rule, then that rule purchases the message and executes. If a new situation is sensed, and a message is posted about it, the agent learning system learns to recognize this message and, through simulation and testing, evolves a response to the new situation.

RESULTS

The initial implementation in the jME environment did not save best rule lists and would only learn subsets of the jobs, and thus the saving of the best rule lists has been one of the prime goals of the retargeting. Even so, the earlier agents generated emergent behavior that showed how the agents evolved terrain crossing rule sets, and showed examples of the adversarial behavior between the blue and red agents. This project retargeting is still in progress; however, a testing environment was developed to simulate a CAS environment for further tests on the learning system. Results show that the agents did develop effective learned behavior, including problems employing temporal delayed rewards and those that test the ability to learn non-Markovian behaviors, both of which are exceedingly difficult for XCS learning machines, but were learned naturally by the message-list-type LCS learning machine. A representative subset of the results follows.

The testing framework consisted of a set of tasks that an agent had to perform in a certain order. The end goal was to have the tasks in the correct order, with no multiple task execution, unless multiple tasks are part of the desired job workflow. This problem is similar to the learning of a state machine, and is of a highly random-access nature, having been developed in conjunction with a drive to have the mobile agents traverse an internet.

The tasks could be compared to the correct COA to achieve a specific goal. The COA test results are shown in the learning graphs of Figure 5a and b. Figure 5a shows the middle line as the average of 500 times through trying to learn the COA job. If the job was not learned after 10000 job runs the learning for that trial was stopped. The dotted lines above and below the central solid line show the first standard deviation of the 500 trials at that number of job runs. The top and bottom solid lines show the maximum and minimum of all 500 trials at that job run. It indicates that some trials learned very quickly while others did not. If a job was learned before the 10000th job run in a trial, the final fitness was extended out to the 10000th entry, since it was completely learned. Figure 5b shows a histogram of the learning. The first bar shows that 500 trials were completed, and the remaining bars show how many trials completed learning within that number of job runs. For instance, approximately 200 of the 500 trials completed within 500 job runs. Approximately 24 of the 500 trials did not complete learning by job cycle 10000.

The COA task performance was recorded in internal memory in the agent, and viewed by the fitness function (the coach) to determine the various successes of the LCSGP. The rewards were immediately given to the rule that just accomplished the portion of the solution (at the end of the LCS epoch), and were given only once in a job. For instance, if the agent did task 1 to 2, a medium sized reward was given. Accomplishing more of the job, as in moving from 1 to 2 to 3 to 4 yielded a much larger reward. Learning segments of the job, such as 3 to 4 to 5, earned small rewards, if shaping rewards were used. In Reinforcement Learning, shaping rewards (Ng, 1999), are rewards that are given for accomplishing facets of

the job that are known ahead of time and can be programmed into the fitness function by a knowledgeable trainer. Of course, these simple goals are easy to program, but more advanced goals have also been achieved that rely on multiple, non-explicit steps. These multiple-step rewards are known as temporal delay rewards, since the agent may have to accomplish many steps before getting rewarded; there is not a constant reward flow. Punishments were also given to the rules that caused errors in the job, such as going backwards from task 4 to 3, or performing task 2 more than once. Parsimony punishments were given at the end of the job, if extra, unneeded, work was performed.

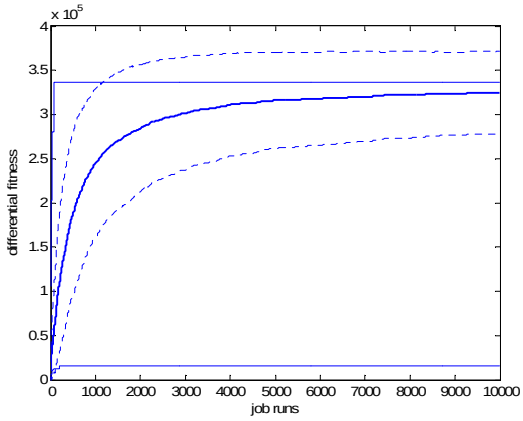
CONCLUSIONS

Creation of a CAS environment that runs in real-time with a multi-agent learning system requires an effort that embodies detailed artificial intelligence techniques and in-depth knowledge of the simulation platform, as well as generation of a suitable architecture to join the two. The paradigm of control in this system is somewhat different than in that of a Semi-Automated Forces system, since the controls and behaviors are evolved instead of preprogrammed. There was a sizeable learning curve required in order to get the system operational in the 3D simulator. Now that the IIOP-based architecture has been developed, it is expected and planned that re-hosting and re-application should be greatly facilitated, and the focus can shift to developing the fitness functions that allow the agents to learn the simulation tasks they need to perform to be a suitable training adversary.

The agents themselves need a training program in order to keep them tuned up for new changes in the environment or to learn new tasks, or just to adapt to new techniques by the human adversaries. In that regard, the latest architecture includes a trainer agent that works to keep the agents learning and improving their best rule lists.

The debugging of such a system is rather consuming in that the behaviors of the agents in the environment can be attributed to multiple sources. The terrain affects the physical motion, as do other aspects of the physics engine; the learning system supplies the commands to the agents, and the commands are developed due to agents learning in the environment and from adaptation due to fitness pressure. The agent learning system log files, which may be optionally enabled, were generally in the order of multi-gigabytes in size and need to be examined to determine a) is the agent learning, b) is the learning system operating correctly, c) are the commands to the environment being issued correctly, and d) is the environment executing the commands faithfully. Furthermore, it is necessary to verify that the situational awareness is being reported correctly to the agent. Due to the complexity required to have an agent automatically learn, careful and exacting examination of every step of the learning process must be reviewed and checked. Although fitness pressure in a learning system always tends to provide improvements that even go around bugs in the system, the learning was more improved and effective as the system became more and more operational.

a.)



b.)

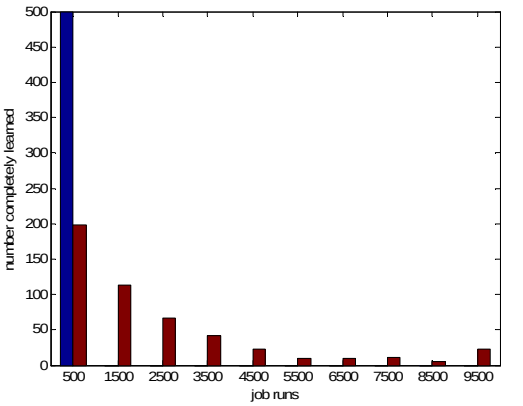


Figure 5a & b. COA shaping/incremental fitness scheme.

The 3D agent systems is also a first step towards improved Course of Action (COA) analysis. For COAs that consist of avenues-of-approach and time-based operations in a 3D battlefield, the interacting agents with their capabilities that are meant to simulate the actual soldiers and forces in the battlefield provide a means to help determine how one side or the other could win. A human could play blue force or red force, thus doubling the utility of the simulation scenario. In a real battle situation it is expected that there would be

real-time situational awareness of terrain, enemy and friendly forces, and other influences such as natural environment situations, including smoke, which would all be included in the CAS simulation. This CAS simulation would be exercised and learning would be conducted in faster than real time in order to help determine the best COA. With the use of the fitness forces driving the COA development, the COAs are generated with respect to battle plans and priorities, as these can factor into the fitness calculations for the individual agents and the plan as a whole.

The system has advanced and has overcome initial problems discovered in the earlier implementation. It is now proceeding into integration, test, and training. The use and response of the system to doctrine and battle plans as incorporated in the fitness functions should result in some interesting behaviors, as seen by testing it in a first-person interactive simulation.

As the system gets incorporated into more simulations, trained to operate, and used for training students, then metrics and paradigms of use will be calculated and emergent. It is expected that the agents that are trained for individual students will be able to follow them around in their training career and be used to enhance their performance by helping them to overcome any weaknesses that were discovered to be exploitable by the agents. These agents, or rather, their learned behaviors can be stored and transported through updates to the Training Management Systems.

The agents are trained in a simulation that allows them to evolve behaviors that work within the simulation to behave in a more optimized manner. It will be important that the agent evolution be controlled to not exceed the capabilities of a human, so that there will be a fair fight capability. That is one of the reasons that the cognition cycle of the agents, the LCS epoch, is constrained to put it more on par with the reaction time of a human.

The agents are expected to perform in a human manner, but with the ability to learn from experience. They will not be given ground truth. They will be provided with the same resources that a human adversary has in the simulation. It should be interesting to see what evolves.

ACKNOWLEDGEMENTS

Thanks to Lockheed Martin Global Training and Logistics for funding the research and development involved in the technology behind this paper.

REFERENCES

Baddeley, A.H. (1997). *Human Memory: Theory and Practice*, Hove, East Sussex: Psychology Press Ltd.

Clauswitz, C. Von (1830). *On War*, New York, Oxford University Press (2007).

Harrison, G.A., & Worden, E.W. (2007). Genetically programmed learning classifier system description and results. *Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, London: ACM.

Holland, J.H. (1995). *Hidden Order: How Adaptation Produces Complexity*, New York: Addison-Wesley.

jME Wiki, (2009). *jME Wiki :: Complete Features List.* Retrieved June 28, 2009, from http://www.jmonkeyengine.com/wiki/doku.php?id=complete_features_list

Koza, J.R. (1994). *Genetic programming II: automatic discovery of reusable programs*. Cambridge, Massachusetts: MIT Press.

Lange, D. (1998). *Programming and Depoying Java™ Mobile Agents with Atlets™*. Upper Saddle River, New Jersey: Addison-Wesley Professional.

lmephysics (2007). *Jmephysics: Project Home Page*. Retreived June 28, 2009, from https://jmephysics.dev.java.net

Miller, J.H. (2007). *Complex Adaptive Systems: An Introduction to Computational Models of Social Life*. Princeton, New Jersey: Princeton University Press.

Ng, A. Y., Harada, D., & Russell, S., (1999) Policy invariance under reward transformations: theory and application to reward shaping. *Machine Learning, Proceedings of the Sixteenth International Conference*, Morgan Kaufmann. pp. 278–287.

Pernin, C.G., Comanor, K., Menthe, L., Moore, L.R, and Anderson, T. (2008). *Allocation of Forces, Fires, and Effects Using Genetic Algorithms*. RAND Corporation. Retrieved June 28, 2009, from http://www.rand.org/pubs/technical_reports/2008/RAND_TR423.sum.pdf.

Russell, S. and Norvig, P. (2002). *Artificial Intelligence: A Modern Approach (2nd Edition)*, New Jersey: Prentice Hall.

Santa Fe Institute (2009). *Santa Fe Institute*, retrieved July 3, 2009 from: http://www.santafe.edu/

Wilson, S.W. (1995). Classifier fitness based on accuracy. Journal of Evolutionary Computation, pp 149-175.

Yang, A. and Shan, Y. (2008). *Intelligent Complex Adaptive Systems*, Hershey Pennsylvania: IGI Publishing.