No More Zombies!
High-Fidelity Character Autonomy for Virtual Small-Unit Training

Brian S. Stensrud, Ph.D.,
Angela Woods,
Samuel Wintermute, Ph.D.
Soar Technology, Inc.
1942 County Road 419 #1060
Oviedo, FL 32766
stensrud@soartech.com

Eugene Ray Purcel
Marine Corps
Warfighting Laboratory
3255 Meyers Ave.
Quantico, VA 22134
eugene.pursel@usmc.mil

Gino Fragomeni, Pat Garrity
U.S. Army Research Laboratory
SFC Paul Ray Smith, Simulation &
Training Technology Center
12423 Research Parkway
Orlando, FL 32826
gino.fragomeni@us.army.mil

ABSTRACT

Virtual practice environments can offer situated, realistic learning experiences if properly implemented. 'Serious Games' delivered within these environments offer visually compelling experiences, but often suffer from a lack of realistic interactions with virtual characters such as teammates, adversaries, and other non-combatants. Artificially intelligent human behavior models – intelligent agents - provide a variety of features not present in legacy computer-generated forces (CGF) systems; including goal-directed dynamic decision making, non-determinism, interactivity, and transparency. Using a cognitive architecture, intelligent agents exhibiting these features can be brought to bear for virtual training environments to support both kinetic and non-kinetic small-unit training exercises.

In partnership with the U.S. Army Research Lab's SFC Paul Ray Smith, Simulation & Training Technology Center (STTC), we have developed a suite of intelligent agents for virtual environments that can realistically engage human players in small-unit training scenarios. The centerpiece of this work is a knowledge-rich OPFOR sniper behavior, capable of detecting and selecting friendly targets of opportunity, communicating with other insurgent support entities such as lookouts, and finding the appropriate escape path to avoid detection and capture. In addition to a sniper entity, we also developed agents to play supporting roles, including autonomous fire team agents and non-combatant townsfolk. In this paper, we describe in detail the challenges encountered during this effort and how intelligent agents can be exploited to improve training within systems such as the Dismounted Soldier Training System (DSTS). In addition, we outline the reusable integration architecture we developed to connect our agents to EDGE, a massively-multiplayer online virtual environment developed at STTC, and describe the design choices made to ensure that the architecture can be reused to connect both these agents and other AI technologies with new virtual environments as they become available.

ABOUT THE AUTHORS

Dr. Brian Stensrud is a lead scientist at Soar Technology specializing in artificial intelligence technology for training simulations and related applications. On staff since 2003, Brian has been principal investigator/project manager for over $3M in research contracts, as well as serving in a technical lead role in the development of several autonomous agent platforms and toolsets. He has been selected for two Army SBIR Achievement Awards, in 2010 and 2012, for work performed by the U.S. Army Aviation, Research, Development and Engineering Center (AMRDEC). Brian received a Ph.D. in Computer Engineering from the University of Central Florida (2005), and BS degrees in Mathematics and Electrical Engineering from the University of Florida (2001). He has over 10 years experience in artificial intelligence, behavior modeling, and simulation.

Mr. Ray Pursel served over 23 years as a Marine in both enlisted and officer roles. His billets ranged from Aviations Operations Clerk to Helicopter Section Leader to Modeling and Simulations Officer. He earned a B.S. in Computer Science and Mathematics Minor from the Pennsylvania State University in 1995 and an M.S. in Modeling, Virtual Environments and Simulation from the Naval Postgraduate School in 2004. Now retired from active duty, he is serving as a Modeling and Simulations Analyst with the Marine Corps Warfighting Laboratory.

Mr. Gino Fragomeni serves as a Science & Technology Manager for Dismounted Soldier Technologies at U.S. Army Research Laboratory, Simulation & Training Technology Center (ARL-STTC). He currently works in Ground Simulation Environments Division conducting R&D in the area of dismounted Soldier training & simulation. His current interests include artificial intelligence and immersive environments centric to dismounted training applications. Gino is also a reservist with the United States Army Special Operations Command-Central (SOCCENT) with over 28 years of military experience. He earned a Master of Science from the University of Central Florida (2002) and has specialized training in Systems Engineering and Simulation.

Mr. Pat Garrity is the Chief Engineer for Dismounted Soldier Training Technologies at the Army Research Laboratory's Simulation and Training Technology Center (ARL STTC). He currently works in the Ground Simulation Environments Branch conducting research and development in the area of dismounted Soldier training and simulation where he was the Army's Science and Technology Objective Manager for the Embedded Training for Dismounted Soldiers Science and Technology Objective. His current interests include Human-In-The-Loop (HITL) networked simulators, virtual and augmented reality, and immersive dismounted training applications. Garrity earned his B.S. in Computer Engineering from the University of South Florida in 1985 and his M.S. in Simulation Systems from the University of Central Florida in 1994.

No More Zombies!
High-Fidelity Character Autonomy for Virtual Small-Unit Training

Brian S. Stensrud, Ph.D.,
Angela Woods,
Samuel Wintermute, Ph.D.
Soar Technology, Inc.
1942 County Road 419 #1060
Oviedo, FL 32766
stensrud@soartech.com

Eugene Ray Purcel
USMC Warfighting Laboratory
3255 Meyers Ave.
Quantico, VA 22134
eugene.pursel@usmc.mil

Gino Fragomeni, Pat Garrity
U.S. Army Research Laboratory
SFC Paul Ray Smith, Simulation & Training Technology Center
12423 Research Parkway
Orlando, FL 32826
gino.fragomeni@us.army.mil

INTRODUCTION

Virtual practice environments can offer situated, realistic learning experiences if properly implemented. 'Serious Games' delivered using modern game engines and virtual environments offer visually compelling experiences, but often suffer from a lack of realistic interactions with virtual Computer-Generated Forces (CGFs) such as teammates, adversaries, and other non-combatants. Better technologies must be brought to bear in these environments so that trainees can practice the wide variety of skills modern warfare requires, both kinetic (e.g., firefights) and non-kinetic (e.g., face to face interaction with locals). The Army is interested in applying next generation Artificial Intelligence (AI) techniques to help address these needs. Artificially intelligent human behavior models – *intelligent agents* – can provide a variety of features not present in typical CGF platforms:

- **Goal-based**. Not dependent on a pre-specified script, intelligent agents use real-time situation awareness and can act in pursuit of domain-specific goals.
- **Non-deterministic**. Intelligent agents are responsive and sensitive to unpredictable changes and unanticipated events in the environment.
- **Dynamic**. Intelligent agents are able to pursue, prioritize and reason about multiple goals or tasks simultaneously, automatically switching goals when necessary.
- **Interactive**. Intelligent agents are able to communicate information, status and commands and interact with human operators and also with other agents in the context of the exercise.
- **Transparent**. Intelligent agents can explain complex reasoning behaviors and their motivations for validation and after-action review purposes.

We describe in this paper an effort to develop and demonstrate a robust, reusable set of human behavior models that can be exploited virtual, small-unit training exercises, as well as a reusable architecture for integrating such behaviors into virtual training and practice environments. Using STTC's EDGE multiplayer online virtual environment (Dwyer et al, 2011), we developed a set of intelligent agents that can execute a variety of goal-directed tasks completely autonomously within a virtual environment. These agents were developed using an open-source, multi-feature cognitive architecture (Laird, 2012) that has been used to develop a wide variety of knowledge-rich agents for military applications (Jones, 1999; Jones, 2002; Jones, 2004; Stensrud, 2006, Taylor, 2007; Wray, 2009; Taylor, 2011). To integrate these agents within EDGE, we also developed an environment-neutral API that uses Program Executive Office for Simulation, Training and Instrumentation Joint Simulation Bus software protocol (Dumanoir, 2008). The development of this API will allow us to quickly port agents into new virtual environments as they become available. In this report, we describe the agents developed during the effort. Additionally, we introduce the architecture used to integrate the agents into EDGE, and provide a list of requirements for virtual environments so that they can properly support them.

A key challenge of this research effort was to identify the specific requirements and challenges associated with developing agents and integrating them into virtual environments. During the effort, we considered the particular tasks required to develop and integrate these behaviors; identifying patterns, hard problems and design challenges that inform future efforts. The key questions that we considered are as follows:

- *What is required to develop an intelligent agent,*
- *What is required to integrate that agent in a game/virtual environment, and*
- *What does the environment require to be able to support so that it can support that agent?*

CONCEPT OF OPERATIONS

As a backdrop for this effort, we first developed a small unit Concept of Operations (CONOPS) and then generated a set of scenario vignettes. Within the context of this CONOPS, we developed a set of intelligent agents to play each of the various defined roles. In this CONOPS, intelligent agents are responsible for each of the characters in the simulation – all OPFOR, friendly, and neutral entities. As such, this CONOPS is not a training scenario but rather a proof-of-concept exemplar to demonstrate the abilities of intelligent entities in a representative small-unit environment that can be translated to virtual training applications.

Our CONOPS focused on an amateur sniper, holed up in an insurgent compound, who is planning an ambush on a dismounted US platoon force (see Figure 1). Our primary behavior development was in support of this sniper element. Additionally, the vignettes we developed included a variety of enemy, friendly and neutral agents that provide complexity to the environment. The combination of decisions made by these agents at run-time result in a variety of different vignette paths that demonstrate their flexibility. Our final prototype was fully populated with intelligent agents – over 50 in all – with no humans-in-the-loop either as role players or operators.

Figure 1. The sniper agent, waiting for a good shot

In our CONOPS, US Army platoon has just arrived in the northwest section of a typical middle-eastern town. After arriving, two of the squadrons provide security around the vehicles and the north exit of town, while a third squadron proceeds south on foot towards the town square to interview a local religious leader. Leaving behind their vehicles, the squad is vulnerable to small-arms fire, however the city is in an area largely considered to be 'safe' with a low probability of enemy contact.

Unbeknownst to the squad, our amateur insurgent has taken a position within a compound near the religious leader's residence. This insurgent is looking specifically for US Army targets to shoot at and, while untrained and unlikely to do much damage from a distance, could be able to take advantage of lapses made by the squad such as poorly executed movement tactics or improper cover and concealment techniques. The squad's two fire teams traverse along the main road towards the compound (see Figure 2), unaware of the imminent threat, with the squad leader attached to one fire team.

Figure 2. Fire team agents move in line formation

AGENT ROLES

The sniper agent exploits a variety of information to make decisions about how and when to engage the squad. First, he has two 'lookout' agents that observe activity in the town and provide timely status information to the sniper. The first lookout, located near the town square, collects input on the squad's intentions and communicates with the sniper over walkie-talkie. The second buddy provides information about the status of the sniper's possible escape routes. Based on this information, the sniper chooses a spot to engage the squad and plans an escape route after taking his shot. From there, the sniper waits for a high-probability opportunity to take a shot, based on the distance and movement techniques of the incoming fire teams – specifically whether they are properly using proper cover and concealment.

To support and highlight the dynamic behavior of the sniper entity and his cohorts, we also developed and integrated friendly and neutral agents whose decisions impact the sniper's strategy. The friendly agents developed are individual combatants in the squad

tasked with patrolling the area. These agents are able to traverse an area in proper formation, react appropriately to contact situations, and engage the sniper target if he is detected. We have encoded variation into these agents, such as improper cover and concealment and security procedures. This variation in turn affects the decision-making of the sniper agent, including where/when he will fire and if/how he will escape the compound after engaging.

Figure 3. Intelligent clutter agents engage in realistic activity around town

Additionally, we also developed and integrated civilian agents (see Figure 3 above) that populate the town and behave in various ways. Neutral civilian agents occupy the streets of the town, navigating from building to building to simulate errands and other daily activities. We have also introduced some civilian agents who are sympathetic to either the insurgent or coalition cause. Civilians sympathetic to the enemy, for instance, serve as lookouts for the sniper agent, and can communicate to the sniper when escape routes are blocked or safe for use. Similarly, civilians working together with US forces can keep an eye out for suspicious activity and point out escaping insurgents to the squad.

SNIPER AGENT DESIGN

As the focus of our CONOPS, the sniper is the most sophisticated of the intelligent agents developed during this effort. We describe here the significant decisions and parameters considered by the sniper agent, whose design is depicted in Figure 4. This diagram depicts the decision-making process of the sniper agent throughout the course of the scenario. It is NOT simply a linear algorithm or script through a particular use case. The sniper agent (as well as the other agents in the CONOPS) is constantly monitoring the state of the environment at a rate of over 20 times per second, and using that situation awareness to determine which goals or actions to pursue. This persistent situation awareness allows the agent to pursue, prioritize and reason about multiple goals or tasks simultaneously, automatically switching goals when necessary.

The sniper enters the world in the back of a compound on the east side of town, ostensibly engaged in conversation with a lookout agent standing nearby (Figure 5). At some point, a radio message might be received from a lookout agent, who is walking in the area of the main square. This message causes the snipe goal to be invoked. To achieve this goal, the sniper must first move to a position from which to shoot. He knows of two good locations in his compound, one with a view of the main road, and the other with a view of the back entrance to the elder's compound. Depending on whether or not the lookout informed the sniper of the purpose of the Soldier's visit, either destination might be used.

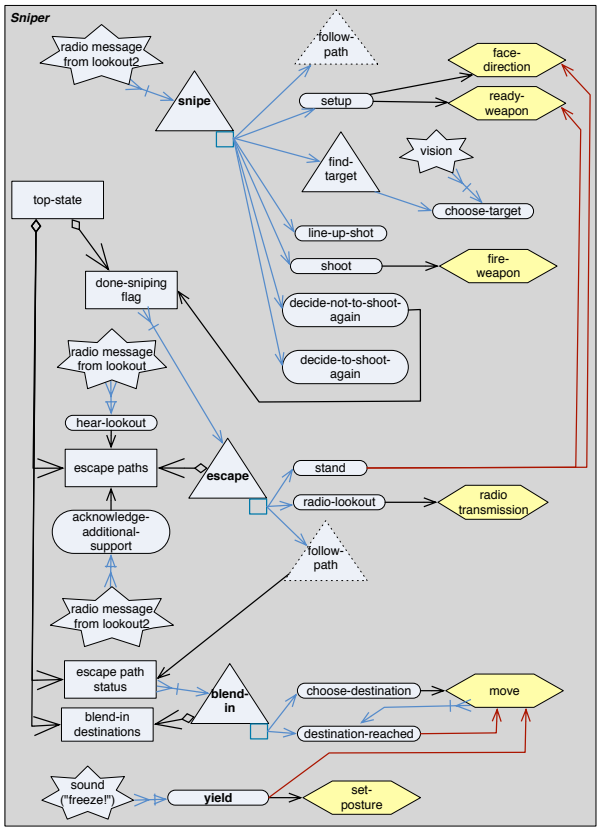

Figure 4. Sniper Agent Decision-Making Diagram

A follow-path goal is created in order to reach the sniping position. Once this position is reached, the sniper takes out his weapon and generates a goal to find a target. In this goal, when a Soldier comes into view, an action is proposed to shoot at that Soldier. If multiple Soldiers are present, this provides an opportunity for the sniper to reason about which makes

a better target—as implemented, there is a preference to shoot at targets who are already injured.

After selecting a target, there is optionally an action selected for the sniper to carefully line up the shot. Ideally, this would allow the sniper to execute a speed/accuracy tradeoff, having a higher probability of hitting the target if he takes more time to aim. However, as the current weapon simulation is not sophisticated, this simply delays the shot. This action is used when the sniper is aiming at the back entrance, since he knows that the Soldiers are waiting there, and won't quickly pass out of view. The sniper next takes the shot. Once the shot is executed, the sniper decides whether or not to shoot again. This choice depends on an aggression parameter; a more aggressive sniper will take more shots even as his probability of successful escape may go down.

Figure 5. The sniper and lookout agents await information about the squad near the east exit

Once the sniper has decided not to shoot again, a goal to escape is invoked. The sniper maintains representations of two potential means of escape: quickly walking to a hideout on the other side of the town, via the back exit of his compound, or casually walking into the crowd of civilians near the front entrance of his compound. Communications from the lookout agents will cause the sniper to weigh the value of these options differently. The lookout near the back of the compound might sight the goat herder nearby, who is known to be friendly with the Soldiers, and alert the sniper. The lookout near the town square may also learn that an additional squad of Soldiers (Squad C) is moving near the crowd, and radio the sniper.

If both of these communications occur before the sniper takes a shot, the goal to snipe will retract, and he will stand down and remain in the compound. Otherwise, the information is used to achieve the escape goal. The back entrance is preferred over the front if neither (or both) of the lookouts has provided information. Before escaping, the sniper first radios the lookout in the compound, letting him know which route he is taking; the lookout will leave via the same route. The follow-path goal is again used to execute the escape movement.

If the front escape was used, a new goal is invoked once the sniper gets to the crowd. The sniper attempts to blend in with the crowd, wandering aimlessly between several waypoints in the courtyards and road. This is interrupted, though, if a Soldier finds and arrests the sniper (by yelling "freeze!" nearby). The sniper complies with the order, stopping and lying prone on the ground.

Detailed design descriptions for the squad, lookout and neutral crowd behaviors have been omitted for this paper.

A GENERIC AGENT INTEGRATION ARCHITECTURE FOR VIRTUAL ENVIRONMENTS

Integrating complex, realistic agents into a game architecture successfully requires an integration strategy that addresses the specialized needs that agents impose on a game environment. However, creating a functional agent is not the only goal. One of the primary advantages that agent behaviors hold over scripting is that they are not brittle. Agents can adapt and function in dynamic environments. Scripted behavior can seem very realistic, but as soon as any of the parameters of the environment change, the script is rendered obsolete. Agent behaviors, in contrast, can successfully reason over changing parameters in the game environment and select reasonable behavior to exhibit in each situation. One of the primary goals of the agent integration effort was to design the integration framework to take advantage of the agents' ability to reason successfully over dynamic activity within the game world.

In addition, particular attention was given to re-usability. Not only is it useful for agents to be developed in a way that allows the same agent to function in multiple scenarios in the same game engine, but also the agent can (ideally) be developed and integrated in a way that allows the game engine itself to be changed to a different game engine in the future. Doing so allows agent behaviors to be developed and composed into re-usable libraries, which can be deployed in multiple scenarios and in multiple game engines.

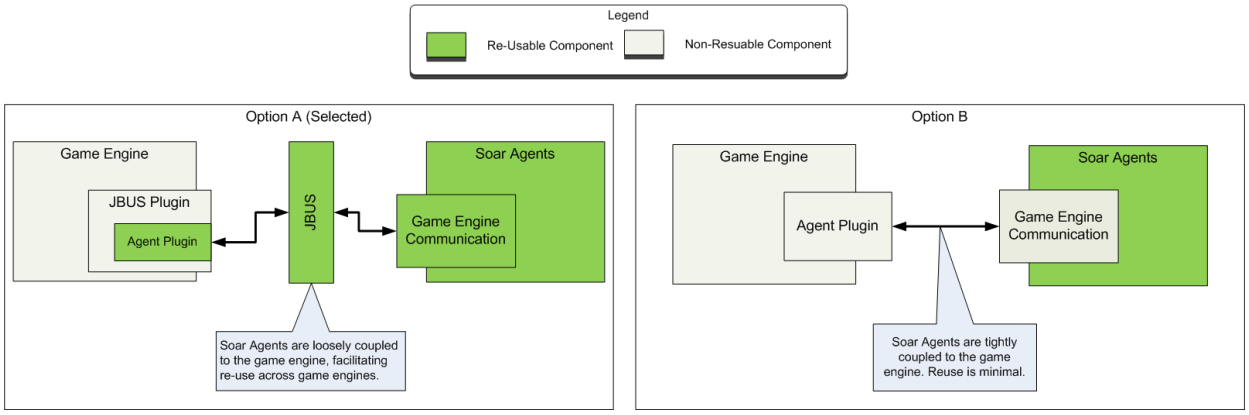

Figure 6. Reusability of intelligent agents in virtual environments is heavily dependent on a loosely coupled integration architecture

After selecting the EDGE platform as the game engine, the next important decision was determining how tightly to couple the intelligent agents to the game environment. If the agents are tightly coupled to the environment, they can leverage every nuance of the particular game environment. However, doing so reduces the ability to re-use the integration effort, and in some cases the agent behavior in the event the game engine must be changed. Since game engine vendors are fiercely competitive, and new engines routinely emerge, designing the integration architecture to allow the game engine to be changed was given a high priority. By leveraging existing software protocols – in this case PEO STRI's JBUS - we were able to develop a game-neutral architecture that provides loose coupling between intelligent agent technologies and a game environment, which results in maximum reuse of those agents if and when the game environment changes.

The concept of loosely- versus tightly-coupled agent integration with a game engine is illustrated in Figure 6. In the figure, Option B depicts a direct, tight connection to the game engine. In the event that the game engine must be changed, minimal re-use would be possible because the integration code is all very specific to the particular game engine for which it was written. Instead, we selected Option A, using JBUS. JBUS provides an interpretation layer between the game engine and an external component such as an intelligent agent. JBUS provides a messaging protocol for communication between software components. STTC's EDGE developers have developed a set of game engine neutral messages that can be sent over the JBUS pipe. They have also authored the JBUS Plugin to translate the game-engine neutral messages to/from terms specific to the game engine. As the figure shows, this greatly enhances re-usability. By writing the intelligent agents to communicate with JBUS, they are fully re-usable, shielded when the game engine is changed. Only the JBUS Plugin needs to be updated when the game engine is changed.

After selecting the basic integration strategy, a detailed design was developed that met our needs. STTC's EDGE developers were consulted, and advised us on refinements to our design that ensured that it was aligned with their long-term architectural vision for the EDGE and JBUS platform. The resulting design is shown in Figure 7.

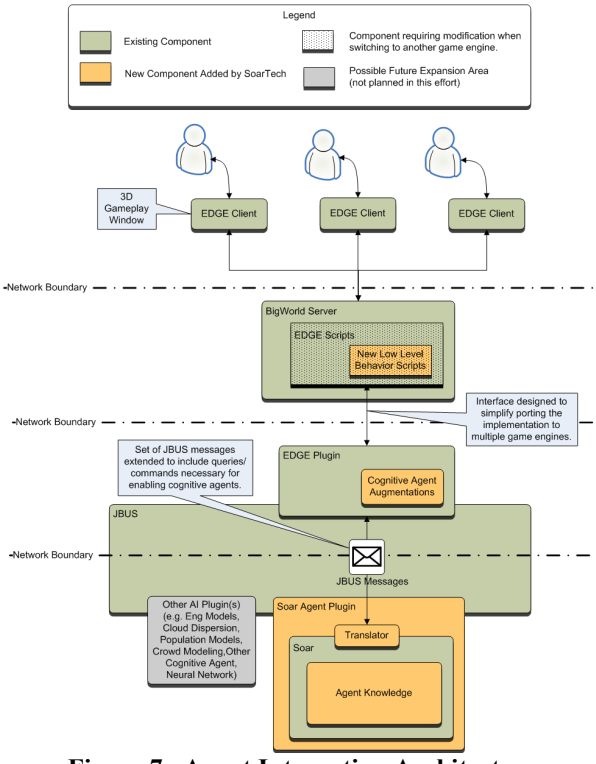

Figure 7. Agent Integration Architecture

The agents are interfaced to the game engine through JBUS. To use JBUS for communication, an agent plugin was developed whose responsibilities are:

- Converting agent commands for the game into JBUS messages.
- Converting JBUS messages from the game into agent knowledge.

The location of the Agent Plugin in the architecture was designed so that the same strategy could be used for integrating other types of AI, such as other cognitive architectures or even sub-symbolic techniques such as Bayesian or neural networks. While only a particular type of agent was integrated in this effort, similar plugin techniques could be used if other types of behavioral or cognitive models are desired.

After an assessment of the existing capabilities of the EDGE Plugin and set of available messages, it was determined that a few augmentations would be required. The existing capability set was targeted at replicating the type of communication that is typical over HLA (High Level Architecture) or DIS (Distributed Interactive Simulation), such as pushing out platform updates for entity state. While this is useful, it is not sufficient for enabling an agent to produce realistic behavior. Most agents require the ability to issue complex commands to the game environment as well as extract additional information from the game. Augmenting the existing communication system to accommodate agent needs required changes in three places:

- Adding new types of JBUS messages.
- Augmenting the EDGE Plugin to process the new JBUS messages.
- Augmenting the game engine code that translates the new message types into engine-specific terms.

Special consideration was given to developing the new set of messages such that the messages would be game engine neutral. For example, queries such as "Is Object A Visible to Object B?" are game engine neutral. Exactly how the game engine decides visibility, whether it is by ray-casting or some other technique is not important. What is important is that it can answer the question. Similarly, commands such as "Move Object A to Point B" should be carried out in a sensible way such that Object A avoids colliding with obstacles on its way to Point B. But again, the specific path planning technique (such as A*) that the game engine uses to carry out the command is not specified in the game engine neutral message.

Another key consideration in a multiplayer, distributed game is balancing the computing load across the machines that are dispersed across the network and designing a scalable solution. Figure 7 shows the network boundaries in our approach. In a typical client/server game, individual trainees sit at a computer and interact with a game client, while a game server hosts the game simulation. This is true of the integration architecture we have chosen. However, utilizing JBUS offers us the opportunity of introducing a second and third network boundary - one that separates JBUS from the game server, and another that separates other back-end processing applications (such as agents) that use JBUS for communication.

The ability to easily introduce these additional network boundaries is a key advantage of selecting JBUS. Agents are computationally expensive, relative to other types of lightweight behaviors. The game clients that the trainees interact with are already computationally expensive due to rendering high quality graphics. We do not want to add the cost of agent computation to the clients. The game server can be either a single machine, or a cluster of machines. While adding the agents to these machines is feasible, integrating them into a specific game engine's server architecture is a poor choice because it is tightly coupled, and is not a re-usable approach. Using a protocol to enable the agents to reside on their own hardware that is separate from both the game server and the game client is ideal. It is loosely coupled, so a change of game engine will not impact the design. And it is scalable. As the number of agents increases, additional machines can be added to the cluster running the Agent plugin. Our current demonstration shows that approximately 50 agents can be run on a single machine. Since additional machines for agent processing can be added, scaling up the number of agents is not dependent on the computational power required for agent reasoning.

GAME ENGINE REQUIREMENTS

Modern game engines have made great advances in the area of 3D visuals. While good graphics are an integral part of an immersive training experience, they are not the whole solution. In addition to good graphics, non-player characters must display realistic behavior. Using intelligent agents is a way to meet this need. Game engines are just starting to consider the needs of intelligent agents, and to achieve realistic intelligent agent behavior inside a game environment; the game environment has to meet a minimum set of requirements. What is needed for really good AI? What requirements must the game engine meet to provide a sufficient set of capabilities to allow the intelligent

agent to do its work effectively? This effort has placed focus on identifying the capabilities a game engine must have to allow realistic, intelligent agents to be inserted into the game environment.

While many game engines have a robust set of capabilities, most have limited their programming APIs to providing simple status information or issuing basic commands. However, to create an intelligent agent that is inserted into a game environment, a much broader set of controls is necessary. Not only is more robust status information required, but also an adequate set of hooks that an agent can use to control elements in the world at a much finer granularity than legacy models or simple behavior scripts.

We identified the following requirements as necessary to provide an effective environment for intelligent agent to provide value:

- Source code access
- Scalable infrastructure
- Control of realistic character avatars
- Character attribute access
- Robust ray-casting (visibility) system
- Robust navigation mesh & path planning system
- Robust collision detection system
- Support for weaponry, projectiles and damage
- Sophisticated character/object interaction
- Realistic vehicles capable of movement and character interaction
- Physics support
- Real world terrain correlation
- Extensible level editor
- Extensible game UI (including key bindings)
- Robust performance analysis and debugging tools
- Flexible camera controls
- Weather and time representation

SUMMARY AND LESSONS LEARNED

During this effort, we successfully developed a set of intelligent agents capable of autonomously performing various small-unit tasks and behaviors within a game environment. These agents include a sniper, a set of spies/lookouts, neutral townsfolk, and a dismounted US Army platoon. Each agent interacts dynamically within STTC's EDGE multiplayer online virtual environment. They have predefined goals that influence their decision making, but react to real-time stimuli in the environment and maintain constant situation awareness in the world. Each agent is flexible to support a variety of different conditions and parameters, meaning that it can behave properly within a multitude of different situations without having to be re-encoded, re-scripted, or otherwise controlled by a human-in-the-loop. As a result, these agents can be integrated into virtual training environments, such as the Dismounted Soldier Training System (DSTS), where they can serve as replacements for live role players in a variety of different small-unit training scenarios.

While developing and integrating these agents, we also discovered and compiled a reusable agent integration architecture and API, as well as set of requirements that future decision makers will be able to refer to, when choosing a game engine for their virtual training system, that their system will be able to support robust, integrated behavior models. This architecture, which connects to PEO STRI's JBUS network translator interface, allows for intelligent agents built using a variety of methodologies to interact with a game engine/virtual environment without having to touch the specifics of that environment. Using this architecture, agents can be re-integrated into new virtual environments, without having to make any changes to or re-implementations of their behaviors provided the new environment supports JBUS and the interactions defined in its API.

Lessons Learned: What went right?

1. EDGE. Selecting the EDGE platform as the game development environment for this effort had many benefits. It satisfied most of the necessary game engine requirements, allowing a rich toolset to use for prototyping. In addition, the existing scenario provided a large number of art assets that we leveraged with only minimal modification. Re-use of the stock EDGE scenario allowed us to bypass art asset creation and focus most of our effort on the key development aspects: creating sophisticated intelligent agents, and developing a re-usable integration framework for connecting them to game environments.

2. JBUS. Selecting JBUS as our network communication protocol was an overwhelmingly positive experience. Many integration frameworks are cumbersome and difficult to use, and it was unclear in the beginning if JBUS would provide enough benefit to be worth the effort to configure it. However, with only a small level of effort we were able to configure JBUS for use and get a working prototype in place. JBUS was powerful enough to be the single communication channel, allowing all of our messaging between the agents and game environment to be uniform. In

addition, by using it to transport game neutral messages, it has facilitated creation of a re-usable integration layer, which will enable plugging in different game engines in the future without needing to change the agents.

3. Rapid-Prototyping Message Framework. Development of a re-usable, game engine neutral API was a core focus of the development effort. The final API is the result of analysis of experimental designs and refinements over the course of the project. Since the API was expected to evolve over the course of the project, a strategy for prototype development while it solidified was necessary. To address this challenge, we created a rapid prototyping API. The API contains multi-purpose, powerful messages that could be overloaded for a variety of needs. The prototype messages facilitated rapid prototyping, and shielded prototype development from constant API changes.

4. Reusable, Loosely Coupled Intelligent Agents. Using a client/server architecture for intelligent agent development is ideal. Intelligent agents can be computationally expensive, which is prohibitive if they are required to run on the end user's client hardware. We addressed this problem by using JBUS to provide communication between the intelligent agents and the game environment. The intelligent agents can run on separate hardware outside of the main game loop, communicating over the network with the game client. This solution is highly scalable; adding more agents to the environment can easily be accomplished by adding additional computing power or adding machines to the training infrastructure. Additionally, this architecture allows agents to be reused in other training environments, even those using different game engines.

Note that keeping the intelligent agents loosely coupled necessarily means that the integration architecture, as described, contains a client-side plugin that is tightly coupled to the game engine source code. Otherwise, the agents will not have the appropriate functions available to operate avatars at an acceptable level of granularity. Examples of this point are provided below.

Lessons Learned: What went wrong?

1. Access to Source Code. The game engine the EDGE platform currently uses does not include the source code or a complete algorithm description for its path planning code. As the agent development proceeded, we required complex path planning abilities to be able give agent's realistic movement such as moving in the wedge formation. Unfortunately the game engine's path planning sometimes returns unexpected and confusing results. For example, some points are reported by the path planning system to be "unreachable" although there do not seem to be any obvious obstacles. Obviously, this problem is magnified in environments with nontrivial terrain. Clearly there are some low level details regarding the interaction of the navigation mesh and the path-planning algorithm that are in play. However, without access to the source code, gaining a better understanding of the issue is not possible. Luckily we were able to use the raycasting API to work around some of the worst cases. However, for future game engine selection, better results will be possible if source code access to the path planning code is available.

2. Low-level Access To Character Animations and Postures. The game engine currently used for EDGE does not provide the ability for game characters to move their upper body independently from their lower body. While the visualization does not impact the authoring of the intelligent agent driving the avatar, it does impact the realism of the overall effect. For example, a squad member using proper movement techniques should be guarding their assigned sector. To do this realistically, they need to be able to walk in one direction while looking or aiming their weapon in another direction. We have authored the squad agents to do this. However, since the game engine does not support independent motion, the visualization shows up as a character turning unexpectedly (see Figure 8).

Figure 8. Fire team agents have to fully turn around to scan behind them due to the lack of low-level animation control

For example, if a squad member should be moving north, but guarding the rear of the squad to the south, the desired effect is for the lower body to face north and for the upper body and head to turn to periodically to look behind the Soldier. However, the rendering seen in the game engine is the whole Soldier avatar spinning around 180 degrees, and then back when a head turn or upper body turn would be correct.

While the agent behavior is appropriate, the fidelity of

the visual rendering is low enough that to an uninformed audience, the behavior will sometimes look 'buggy'. The way to resolve this issue is to select a game engine that allows turning the head and upper body independently from the lower body.

FUTURE RESEARCH

In the coming months, we anticipate extending the small-unit behaviors developed on this effort, adding additional features to further demonstrate the utility of intelligent agents for virtual small unit training. These features include, but are not limited to:

1. Agent behavior authoring. This feature will allow for a non-programmer scenario designer to tweak the behaviors of the agents, setting specific parameters such as points of interest and tuning any agent biases (such as aggressiveness and rifle skill) by extending and using the game engine's existing level authoring tool – and without having to edit any code.

2. Human-in-the-loop support. Our current prototype runs as a series of vignettes with the human out of the loop, allowing us to demonstrate the features of the agents in a controlled setting. We anticipate converting this into a more interactive environment, where a human trainee can execute one of the roles in the CONOPS (initially the squad leader). This will require additional agent support, as we will need to integrate mechanisms by which the squad leader can interact with his team and other agents, e.g. through speech recognition and synthesis.

3. Trainee monitoring and dynamic scenario tailoring. In addition to developing intelligent agents that drive entity behaviors of physical, we have also developed 'behind-the-scenes' agents that can detect actions in a simulated environment and react by cueing responses in that environment – either by directing other characters or modifying the environment itself (e.g., spawning more characters to increase clutter in the marketplace, causing an IED explosion). Acting as somewhat of a scenario battle-master, these agents can monitor the trainee's behavior, responding by tailoring the environment when the trainee requires in-game guidance or feedback.

ACKNOWLEDGEMENTS

The work described in this paper was funded jointly by the Marine Corps Warfighting Laboratory and the U.S. Army Research Lab's SFC Paul Ray Smith, Simulation & Training Technology Center (STTC), under contract #W91CRB-11-C-0100.

REFERENCES

Dwyer, T., Griffith, T. and Maxwell, D. (2011), *Rapid Simulation Development Using a Game Engine – Enhanced Dynamic Geo-Social Environment (EDGE)*, Proceedings of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC), Orlando, FL. 2001.

Laird, J.E.: The Soar Cognitive Architecture, 2012, MIT Press.

Jones, R. M., Laird, J. E., Nielsen, P. E., Coulter, K. J., Kenny, P., & Koss, F. V. (1999, Spring 1999). Automated Intelligent Pilots for Combat Flight Simulation. AI Magazine, 15.

Jones, R., Laird, J., Nuxoll, A., & Wray, R. (2002). Intelligent opponents for virtual reality trainers. Paper presented at the Proceedings of the Interservice/Industry Training, Simulation and Education Conference (I/ITSEC), Orlando, FL.

Jones, R. M., Wallace, A. J., & Wessling, J. (2004). An Intelligent Synthetic Wingman for Army Rotary Wing Aircraft. Paper presented at the IITSEC, Orlando, FL.

Stensrud, B., Taylor, G., & Crossman, J. (2006). IF-Soar: A Virtual, Speech-Enabled Agent for Indirect Fire Training. Paper presented at the 25th Army Science Conference, Orlando, FL.

Taylor, G., Stensrud, B., Eitelman, S., Durham, C., & Harger, E. (2007). Toward Automating Airspace Management. Paper presented at the Computational Intelligence for Security and Defense Applications (CISDA), Honolulu, HI.

Stensrud, B., Taylor, G., Schricker, B., Montefusco, J., & Maddox, J. (2008). An Intelligent User Interface for Enchancing Computer Generated Forces. Paper presented at the Simulation Interoperability Workshop, Orlando, FL.

Wray, R., Lane, H.C., Stensrud, B., Core, M., Hamel, L. and Forbell, E. (2009), "Pedagogical Experience Manipulation for Cultural Learning," Proceedings of the 2nd Workshop on Culturally Aware Tutoring Systems, Brighton, UK, July 6-7, 2009.

Taylor, G., Stensrud, B., Maddox, J., & Aycock, H. (2011). Formative Evaluation of an IUI for Supervisory Control of CGFs. Paper presented at the Behavior Representation in Modeling and Simulation (BRIMS), Sundance, UT.

Dumanoir, P., Clinger, B. and Rivera, J. (2008), Evolving Standards in US Army Live Simulations, Proceedings of the *2008 Simulation Interoperability Workshop*, Orlando, FL.