

Simulation Interoperability with a Commercial Game Engine

Mark Ryan, Doug Hill, and Dennis McGrath
Institute for Security Technology Studies
Dartmouth College
Hanover, NH 03755
United States of America
e-mail: {mryan, dhill, dmcgrath}@ists.dartmouth.edu

Keywords:

Simulation interoperability, game engine, medical, mass casualty.

ABSTRACT: *The interest in game engines as platforms for serious simulation has increased dramatically over the past few years. Game engines have made great advances in user interaction and visualization at low cost that have exceeded advances within the simulation community. With few exceptions, though, the early efforts to use entertainment-based games for non-entertainment applications have been focused on creating "serious games." Open standards for distributed simulations like HLA and DIS have been largely ignored by the game community. If game engines are to be genuinely useful to the simulation community, they must be interoperable with existing simulations. The Gamebots interface, originally developed for AI researchers to experiment with game environments, can also be used to facilitate simulation interoperability. We have modified the Gamebots interface to allow external simulations to control the state of internal game objects. The context of our first usage is a mass casualty simulation called Unreal Triage, in which physiological vital signs, such as pulse and respiration, of disaster victims being monitored with wireless physiological sensors are driven by an external biomedical simulation.*

1. Introduction

The Synthetic Environments for Emergency Response Simulation (SEERS)¹ project at the Institute for Security Technology Studies at Dartmouth College seeks to develop training and analysis tools for emergency responders. Military synthetic battlespace development has produced simulation frameworks, standards, and products for warfighters at many echelons. The civilian emergency response community has similar needs for mission rehearsal and training, but has not yet made wide use of the available technology for simulation. The reason for this shortfall is that two major concerns in the emergency response community – cost and ease of use – are not addressed by conventional simulation technology.

The gaming community, on the other hand, has made great advances in visualization, user

interfaces, and distributed, interactive game play at remarkably low cost to the end user. These advances have fueled a surge in interest in commercial game engines as platforms for serious simulation in recent years. Unfortunately, the gaming world has not adopted open standards for networking and interoperability, so that each new game is very much a universe unto itself.

The Unreal Engine provides a realistic visualization environment, simple user interface, and support for ballistic physics and collisions for very little upfront investment. Its scripting language, however, is a suboptimal environment for running robust algorithms and simulating complex phenomena such as human physiology or chemical plumes. Since validated models and simulations already exist, game engines would ideally be able to re-use these models. This paper describes a framework whereby data from an existing simulation can be piped into the Unreal engine and used to manipulate the behavior and state of in-game objects without modifying any game engine source code.

2. Approach

¹ This project was supported under Award No. 2000-DT-CX-K001 from the Office for Domestic Preparedness, U.S. Department of Homeland Security. Points of view in this document are those of the author(s) and do not necessarily represent the official position of the U.S. Department of Homeland Security.

UnrealTriage is a modification (or “mod”) to Unreal Tournament 2004, a “first person shooter” from Epic Games, intended to simulate emergency responders’ experiences at the scene of a disaster[2]. The specific scenario, based on a live mass casualty exercise conducted in May 2004, involves a small airplane crash with 30 casualties. Players encounter victims and must perform basic triage based on the severity of their injuries.

Most behavior in UnrealTriage is implemented via UnrealScript, the Java-like language used by much of the Unreal Tournament game and bundled with the game to all end-users. But UnrealScript, while providing enormous power to the programmer, is not sufficient for all forms of simulation needs.

The physiological attributes of the casualties, including heart rate, respiration rate, and blood oxygen saturation, must vary over time, as the health of some victims will inevitably deteriorate. Implementing this behavior with UnrealScript inside the game engine would require creating an entirely new simulation of human physiology, in effect “reinventing the wheel”.

Anesoft has a series of simulators [3] that simulate patient vital signs over time based on different pathologies or injuries. While the simulation is not compliant with any simulation interoperability standard, output from the simulation can be dumped to comma-delimited text, ripe for parsing and reuse. This data is well-suited for quantifying the state of casualty characters, but a window was needed to pass the data into the Unreal Triage simulation.

Gamebots is a package written for the original Unreal Tournament engine by researchers at the University of Southern California’s Information Sciences Institute with the goal of making the game a suitable test environment for artificial intelligence software[4]. It provides two interfaces – one for controlling characters inside the game (“BotConnection”) and one for piping information out of the game to an external visualization tool (“VizConnection”). What Gamebots lacks is a means to send data other than character behavior into the game.

JavaBots is an example Gamebots client. The package provides client classes for Gamebots that provide basic AI inside Unreal, but are also

easily extensible to facilitate all types of communication using the Gamebots protocol without requiring the programmer to implement any socket code or parse the protocol.

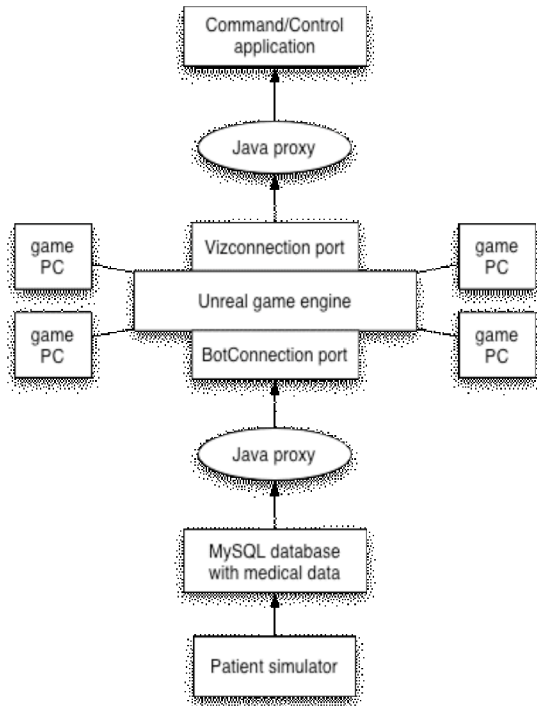
3. Results

The first act of most Gamebots clients upon creating a BotConnection is to send an “INIT” command to the Gamebots server, which spawns a new player in the game controlled by the Gamebots package. If this command is never sent, however, the server will happily accept others. We added a command to the Gamebots API called “SEER” to encompass all the additional functions we wished to add.

One of these functions, “CasualtyEvent”, takes as parameters a numeric identifier of a casualty and the casualty’s pulse, respiration rate, and blood oxygenation. When a CasualtyEvent command is executed, the physiological data maintained within the game for the target casualty is modified. A Java client class named “PhysioClient” executes CasualtyEvent commands transparently, without the programmer’s having any knowledge of Unreal Tournament or Gamebots.

The Java interface is now a method for inserting externally generated data into the Unreal engine, able to accept a stream of live or recorded physiological data. Anesoft does not publish its data as a continuous stream, so we use recorded data used to create the stream. The delimited text dump from Anesoft, which includes the status of the 30 casualties over time, is stored in a MySQL database. A Java program called “CasualtyEventController” queries the database every 30 seconds for updates on each casualty, and uses the PhysioClient interface to insert the figures into Unreal Tournament.

The VizConnection interface is useful for exporting game data to other simulations or applications. We extracted scenario characters’ physiological and location data using this interface and passed it to a remote triage casualty management system [5] as if the data were coming from real biomedical and GPS sensors. With this data, the application could be tested and improved without the need for expensive tests in the field.



4. Conclusions and Future work

Now that the work of defining an interface by which to insert external data into the Unreal engine is complete, we hope to integrate more simulation technologies, particularly models for the dispersion of toxic plumes after a chemical or radiological incident. Crowd control being an integral part of disaster response, we also plan to use external crowd simulations in our scenarios.

Unreal Tournament has proven a quality environment for basic game design and modification. But if game engines are to serve as serious platforms for simulation, then simulation interoperability must be addressed as a fundamental requirement. Game-based exercises, embedded training systems, and virtual prototypes of C4ISR systems will require games that ‘talk’ to each other and to other external applications.

References

- [1] McDowell, P. and Darken, R. "Using Open Source Game Engines to Build Compelling Training Simulations," Proceedings of The Interservice/Industry Training, Simulation and Education Conference (I/ITSEC), Orlando, Florida, December 2004.
- [2] Dennis McGrath and Doug Hill. "UnrealTriage: A Game-Based Simulation for Emergency Response," Proceedings of the 2004 Huntsville Simulation Conference, October 2004
- [3] Schwid, H. A., G. A. Rooke, P. Michalowski, and B. K. Ross, "Screen-Based Anesthesia Simulation with Debriefing Improves Performance in a Mannequin-Based Anesthesia Simulator," Teaching and Learning in Medicine 13 (spring 2001): 92-6
- [4] Rogelio Adobbati, Andrew N. Marshall, Andrew Scholer, Sheila Tejada Gal Kaminka, Steven Schaffer, Chris Sollitto. "Gamebots: A 3D Virtual World Test-Bed For Multi-Agent Research," Proceedings of the International Conference on Autonomous Agents (Agents-2001) Workshop on Infrastructure for Agents, MAS, and Scalable MAS, Montreal, Canada, 2001.
- [5] Suzanne Wendelken, Susan McGrath, George Blike. "A Medical Assessment Algorithm for Automated Remote Triage", Proceedings of the 25th Annual International Conference of the IEEE and Engineering in Medicine and Biology Society, Cancun, Mexico, Sept. 2003.

Author Biographies

MARK RYAN is a research associate at Dartmouth College's Institute for Security Technology Studies. Since arriving at ISTS in June of 2002, Mark has created an XML-based representation of BGP traffic levels, which is now used to generate the Global Instability Index, a measure of Internet health, investigated forensic methods of comparing Linux honeypot images from before and after attacks take place, developed a network simulation to facilitate the TOPOFF and Livewire Cyber Exercises held in 2003 and 2005, and written code to facilitate simulation interoperability with the Unreal Tournament engine. Mark received his B.S.

degree in Computer Science and Classical Humanities from Rutgers University.

DENNIS MGRATH is a Senior Research Engineer at the Institute for Security Technology Studies at Dartmouth College, where his research focuses on simulation for homeland defense applications. His earlier work at Lockheed Martin Naval Electronics and Surveillance Systems included distributed simulation and virtual reality. He also worked for the Naval Air Warfare Center, and is a graduate of Rutgers University (B.S. 1989, M.A. 1998).

DOUG HILL is a Research Engineer at the Institute for Security Technology Studies at Dartmouth College. He received his SB in Physics from MIT in 1979. Doug developed embedded systems, motion control, networking, and web applications before joining the Institute.