Integrating Game Technology and Discrete Event Simulation to Analyze Mass Casualty Scenarios

Jason Loveland

Old Dominion University

Follow this and additional works at: https://digitalcommons.odu.edu/msve_etds

Part of the Computational Engineering Commons, Emergency and Disaster Management Commons, Graphics and Human Computer Interfaces Commons, OS and Networks Commons, and the Systems Engineering Commons

Recommended Citation

Loveland, Jason. "Integrating Game Technology and Discrete Event Simulation to Analyze Mass Casualty Scenarios" (2007). Master of Science (MS), Thesis, Computational Modeling & Simulation Engineering, Old Dominion University, DOI: 10.25777/f83m-1653

https://digitalcommons.odu.edu/msve_etds/110

This Thesis is brought to you for free and open access by the Computational Modeling & Simulation Engineering at ODU Digital Commons. It has been accepted for inclusion in Computational Modeling & Simulation Engineering Theses & Dissertations by an authorized administrator of ODU Digital Commons. For more information, please contact digitalcommons@odu.edu.
INTEGRATING GAME TECHNOLOGY AND DISCRETE EVENT
SIMULATION TO ANALYZE MASS CASUALTY SCENARIOS

by

Jason Loveland
B.S. Computer Engineering, May 2003, Villanova University

A Thesis Submitted to the Faculty of
Old Dominion University in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE
MODELING AND SIMULATION

OLD DOMINION UNIVERSITY
December 2007

Approved by:

Frederic D. McKenzie (Director)

John Sokolowski (Member)

Lee Before (Member)
ABSTRACT

INTEGRATING GAME TECHNOLOGY AND DISCRETE EVENT SIMULATION TO ANALYZE MASS CASUALTY SCENARIOS

Jason Loveland
Old Dominion University, 2007
Director: Dr. Frederic D. McKenzie

In the last 10 years, video games have become complex simulation environments with high resolution 3D graphics enabled by powerhouse rendering engines, multi-player client server networks, user friendly displays and graphical user interfaces, while remaining relatively inexpensive. There is a critical need for systems engineering analysis and rapid trade studies due to changes in operations caused by current events such as terrorist attacks, asymmetric threats, natural disasters, etc. Modern games provide a unique way to visualize and interact with these complex environments, scenarios, missions, and operations. A discrete event simulator (DES) provides an environment to model system architecture behavior, organizational process flow, operational activities, and system performance. The integration of a gaming engine and a DES will allow for the evaluation of concepts of operations in complex, real world scenarios.

The goal of this research is to leverage the advantages of both game engines and DESs to conduct systems engineering analysis in an efficient, feasible, cost effective way. Various game engines, DESs, and applicable architectures for integrating these simulation tools are evaluated and an appropriate framework developed. The resulting integrated simulator is used to evaluate the effectiveness of this hybrid of gaming
technology and DES. This is evaluated in a mass casualty scenario using first responders’ information exchange with hospital emergency rooms.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>MOTIVATION</td>
<td>1</td>
</tr>
<tr>
<td>PROBLEM DEFINITION</td>
<td>1</td>
</tr>
<tr>
<td>APPROACH</td>
<td>2</td>
</tr>
<tr>
<td>OBJECTIVES AND RESEARCH QUESTIONS</td>
<td>3</td>
</tr>
<tr>
<td>THESIS ORGANIZATION</td>
<td>4</td>
</tr>
<tr>
<td>2. VIDEO GAMES AND DISCRETE EVENT SIMULATION</td>
<td>6</td>
</tr>
<tr>
<td>WHY VIDEO GAMES?</td>
<td>6</td>
</tr>
<tr>
<td>ANATOMY OF VIDEO GAME TECHNOLOGY</td>
<td>8</td>
</tr>
<tr>
<td>CURRENT VIDEO GAME ENGINES</td>
<td>16</td>
</tr>
<tr>
<td>SERIOUS GAMES</td>
<td>24</td>
</tr>
<tr>
<td>WHY DISCRETE EVENT SIMULATION</td>
<td>25</td>
</tr>
<tr>
<td>DEFINITION OF DISCRETE EVENT SIMULATION</td>
<td>27</td>
</tr>
<tr>
<td>ADVANTAGES OF DISCRETE EVENT SIMULATION</td>
<td>28</td>
</tr>
<tr>
<td>DISADVANTAGES OF DISCRETE EVENT SIMULATION</td>
<td>29</td>
</tr>
<tr>
<td>CHOOSING A DES</td>
<td>30</td>
</tr>
<tr>
<td>HOW COULD I GET THE DES INTEGRATED WITH A GAME ENGINE?</td>
<td>33</td>
</tr>
<tr>
<td>3. RELATED WORK: MODELING AND SIMULATION</td>
<td>38</td>
</tr>
<tr>
<td>IN EMERGENCY MANAGEMENT AND RESPONSE</td>
<td>38</td>
</tr>
<tr>
<td>UNREAL TRIAGE</td>
<td>38</td>
</tr>
<tr>
<td>PLAY2TRAIN</td>
<td>41</td>
</tr>
<tr>
<td>ARENA EMERGENCY OPERATIONS CENTER PROCESS MODEL</td>
<td>43</td>
</tr>
<tr>
<td>SIMULATION SOLUTIONS AND SYSTEMS ENGINEERING</td>
<td>44</td>
</tr>
<tr>
<td>A CONCEPT PROTOTYPE FOR INTEGRATED GAMING AND SIMULATION FOR INCIDENT MANAGEMENT</td>
<td>45</td>
</tr>
<tr>
<td>THE CLOSEST TOOLS THAT REPRESENT THE INTEGRATION OF A DES TO A GAME ENGINE</td>
<td>47</td>
</tr>
<tr>
<td>4. SIMULATION ARCHITECTURE DESIGN AND IMPLEMENTATION</td>
<td>51</td>
</tr>
<tr>
<td>SERIOUS GAMES-BASED SIMULATOR DESIGN</td>
<td>51</td>
</tr>
<tr>
<td>TECHNICAL INTEGRATION IMPLEMENTATION</td>
<td>52</td>
</tr>
<tr>
<td>SIMULATION PROCESS ARCHITECTURE</td>
<td>53</td>
</tr>
<tr>
<td>SIMULATION SYSTEM ARCHITECTURE</td>
<td>55</td>
</tr>
<tr>
<td>MODIFICATION OF UNREAL TRIAGE</td>
<td></td>
</tr>
</tbody>
</table>
### Contents

- AND THE USE OF GAMEBOTS MESSAGING ........................................... 57
- UNREAL TRIAGE MODDING/EDITING .................................................. 62
- UT-ARENA INTERFACE APPLICATION .................................................. 65
- PTHREADS-W32 ....................................................................................... 66
- GAPI ......................................................................................................... 67
- ARAPI ....................................................................................................... 68
- FATEMAH ALDOULI'S HOSPITAL MODEL ............................................ 71
- MODIFICATIONS MADE TO THE HOSPITAL MODEL ............................ 84
- VERIFICATION AND VALIDATION OF THE SERIOUS GAMES-BASED SIMULATOR .............................................................. 96

5. EXPERIMENTATION AND ANALYSIS .................................................. 100
- THE PROPOSED SCENARIO .................................................................... 100
- ASSUMPTIONS FOR THE EXPERIMENT ............................................. 103
- EXPERIMENTAL DESIGN AND OUTPUT ANALYSIS ......................... 104
- RESULTS ................................................................................................. 107

6. CONCLUSION ......................................................................................... 111
- CONCLUSION DISCUSSION AND CONTRIBUTIONS ............................ 111
- FUTURE WORK ......................................................................................... 112

REFERENCES ............................................................................................ 115

APPENDICES

- A. ARENA TASKS BLOCK ................................................................. 119
- B. ARENA DELAY BLOCK ................................................................. 120
- C. ARENA RECEIVING MESSAGES .................................................. 121
- D. ARENA EMERGENCY OPERATIONS CENTER .............................. 123

VITA ........................................................................................................... 125
# LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Defense Simulations vs. Video Games</td>
<td>7</td>
</tr>
<tr>
<td>2. Video Game Engine Comparison</td>
<td>18</td>
</tr>
<tr>
<td>3. Probability Distributions Obtained Using Arena Input Analyzer</td>
<td>75</td>
</tr>
<tr>
<td>4. Average Patient Total Time for Thirty Independent Replications</td>
<td>109</td>
</tr>
<tr>
<td>5. 90% Confidence Interval for Average Patient Total Time</td>
<td>109</td>
</tr>
<tr>
<td>6. 95% Confidence Interval for Average Patient Total Time</td>
<td>110</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>1. Modular Game Architecture</td>
<td>10</td>
</tr>
<tr>
<td>2. Peer to Peer schema</td>
<td>14</td>
</tr>
<tr>
<td>3. Client / Server schema</td>
<td>15</td>
</tr>
<tr>
<td>4. The Gamebots System</td>
<td>35</td>
</tr>
<tr>
<td>5. User View of Unreal Triage</td>
<td>40</td>
</tr>
<tr>
<td>6. Example EOC Model</td>
<td>43</td>
</tr>
<tr>
<td>7. Architecture Concept for Simulation and Gaming</td>
<td>46</td>
</tr>
<tr>
<td>8. Micro Saint Sharp</td>
<td>49</td>
</tr>
<tr>
<td>9. Extend v7</td>
<td>50</td>
</tr>
<tr>
<td>10. Unreal Triage</td>
<td>53</td>
</tr>
<tr>
<td>11. Simulation Process Architecture</td>
<td>54</td>
</tr>
<tr>
<td>12. Simulation System Architecture</td>
<td>56</td>
</tr>
<tr>
<td>13. Gamebots Client</td>
<td>58</td>
</tr>
<tr>
<td>14. Hospital Status message received in Unreal Triage</td>
<td>70</td>
</tr>
<tr>
<td>15. Patient Flow through the Hospital Model ER</td>
<td>72</td>
</tr>
<tr>
<td>16. Patient Flows in the ER</td>
<td>74</td>
</tr>
<tr>
<td>17. Non-stationary Poisson Distribution for the average of walk-in patient arrival rate between Tuesday and Thursday</td>
<td>76</td>
</tr>
<tr>
<td>18. Non-stationary Poisson Distribution for the average of walk-in patient arrival rate between Friday and Monday</td>
<td>77</td>
</tr>
<tr>
<td>19. Patient Flow in the ER</td>
<td>78</td>
</tr>
<tr>
<td>20. ARENA Model Window Flowchart View</td>
<td>79</td>
</tr>
<tr>
<td>21. Before Doctor Activities Submodel</td>
<td>80</td>
</tr>
<tr>
<td>22. “Determine A Room” Submodel</td>
<td>81</td>
</tr>
<tr>
<td>23. The MTA Submodel</td>
<td>82</td>
</tr>
<tr>
<td>24. Non-Stationary Poisson Distribution for the Average of Walk-In Patient Arrival Rate During Heavy Load Times</td>
<td>84</td>
</tr>
<tr>
<td>25. Arena Hospital Models</td>
<td>88</td>
</tr>
<tr>
<td>26. Message Receiver Submodel</td>
<td>89</td>
</tr>
<tr>
<td>27. UT Patient Arrival Submodel</td>
<td>90</td>
</tr>
<tr>
<td>28. Get Statistics Submodel</td>
<td>90</td>
</tr>
<tr>
<td>29. Message Sender Submodel</td>
<td>91</td>
</tr>
<tr>
<td>30. RTConsole.exe</td>
<td>97</td>
</tr>
<tr>
<td>31. JavaBots</td>
<td>98</td>
</tr>
<tr>
<td>32. Scenario Map</td>
<td>100</td>
</tr>
<tr>
<td>33. Burning Plane and Victims</td>
<td>101</td>
</tr>
<tr>
<td>34. Plane Crash Victims</td>
<td>102</td>
</tr>
<tr>
<td>35. Scenario 2 Smart Agent</td>
<td>103</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

1.1 MOTIVATION

The age of unprecedented change in the way organizations operate is here due to an increase in unexpected events associated with terrorism, natural disasters, and war. These dynamic events create a need for modeling and simulation to explore and analyze the new environment. Adaptations and enhancements of operational process flows and system architectures need to take place continuously. This requires advanced virtual experimentation and simulation because experimentation with the real system would be too disruptive and costly and often is not possible at all [19].

1.2 PROBLEM DEFINITION

In particular, there is a need to apply systems engineering best practices, modeling and simulation, and commercial gaming technologies to solve new, real world problems. We must analyze concepts of operations and scenarios to help keep peace and help prevent, defend against, and respond to terrorist activities, natural disasters, and asymmetric events. There is a need to look at the ways systems are designed, policies are implemented, and operations are executed within every organization committed to keeping up with changing times. This involves leveraging the rigor of systems engineering tools, validated architectures, and a three-dimensional, realistic perspective. Also, we must develop simulations that allow researchers to test some aspects of their solutions without requiring complete implementation or access to a physical arena.

This thesis follows the style of IEEE Transactions on Aerospace and Electronic Systems
More specifically, this thesis evaluates and analyzes a hypothetical mass casualty triage scenario in Hampton Roads, Virginia using systems engineering and modeling and simulation tools. Due to lack of reliable technology required to adequately share real-time information, triage is administered by the first responder at the disaster site without knowing the status of hospitals and other health service facilities in the region [42]. There are many scientists, engineers, and researchers who are using systems engineering tools such as modeling and simulation to architect and evaluate alternative solutions for these complex information technology challenges [McLean, Gunal, Pollak, McGrath]. Using modeling and simulation, I will evaluate alternative technologies and architectural solutions for mass casualty triage through the integration of a discrete event simulation tool and a commercial game engine.

1.3 APPROACH

DESs provide a modeling and simulation environment that enables an engineer to perform analytical trade studies on organizational processes and system architecture alternatives. One can evaluate the organizations involved in a mission and their processes for handling events. It is also possible to model the operational activities and system functions. With this technology, an engineer can truly understand the architecture behavior, critical time lines, and bottlenecks associated with executing a scenario. Almost all viewpoints of the respective architectures can be analyzed in a DES; however, the challenge will be to view them through a real world perspective. Few, if any, DES solutions provide a truly interactive, three-dimensional virtual interface to experience the simulation from a first-person perspective.

Commercially available game engines provide a physics-based environment to
allow operators, engineers, and analysts to interact in a three dimensional, realistic world. Multiple players and actors can literally walk through the scenario, mission, or vignette because of a client-server network enabled architecture. Each perspective can be tailored to the users with modifiable heads up displays (HUDs) and operator interfaces. However, current video game engines do not leverage high fidelity mathematical representation of real world operational and system architectures. They also do not provide a realistic, time ordered execution of events [18].

I propose that the integration of these modeling and simulation capabilities will enable an environment that can allow engineers to analyze how current and future systems perform in the complex scenarios of today. This paper will discuss the process associated with creating this modeling and simulation environment called the serious games-based simulator. It will also discuss an example implementation that shows the value of the simulator's inception.

1.4 OBJECTIVES AND RESEARCH QUESTIONS

The overall objective of this thesis is to use experimentation to explore the utility and show the usefulness of the serious games-based simulator for evaluating first responders' needs during a mass casualty event. More specifically, the serious games-based simulator will be used to compare a mass casualty scenario where a first responder does not have information about hospital statistics and a scenario where the first responder has a hand held mobile communications device, or PDA, that provides real-time hospital status information. The goal of this experiment is to evaluate the average patient wait time resulting from the first responder with and without using updated hospital status information.
This thesis should prove that I can analyze and optimize real-time decisions of a first responder to a mass casualty event with the modeling and simulation environment. I should be able to integrate DES and game technology to evaluate the system/operational architecture and first-person decision making respectively. Also, the integration of these two technologies should provide analysts with enough data to evaluate scenarios from many perspectives including systems engineering, data engineering, and human factors. All of the research questions below will be addressed during this study and will be evaluated using the serious games-based simulator:

- Can I analyze and optimize real-time decisions of a first responder to a mass casualty event with modeling and simulation? If so, what tools, methodologies, and processes are required?
- How can video games be useful and interesting tools for real-time training and analysis of first responders in a mass casualty event?
- What are the advantages and disadvantages of DES for real-time execution and analysis of process flows, system event traces, and messages?
- Can a DES model of Regional Hospital's Emergency Rooms operations and architecture provide enough information for a first responder in a video game environment to have increased situational awareness, thus yielding better decision making and increasing the number of lives saved?
- Will a first responder find more information useful? What kind of information would a first responder need when responding to a plane crash with mass casualties?

1.5 THESIS ORGANIZATION

Chapter two presents a background of modeling and simulation specific to video
games and DES. Chapter three discusses current modeling and simulation relative to emergency management and response, relevant research, and introduces the concept of the serious games-based simulator. Chapter four begins with a general description of the study and the serious games-based simulator. It also includes a description of implementing serious games-based simulator with both the simulation systems architecture and process architecture. Chapter five discusses the experimentation and the results of modeling and simulating a mass casualty event with the serious games-based simulator. Finally, chapter six contains the conclusion, a general evaluation of the study, contributions, and future work.
CHAPTER II

VIDEO GAMES AND DISCRETE EVENT SIMULATION

2.1 WHY VIDEO GAMES?

Visualization is a crucial piece of a modeling and simulation environment needed for engineering analysis of complex systems because it allows you to walk through a multi-perspective view of the concept of operations. Commanders and decision makers can enter the virtual battlespace and experience an environment close to reality. It can also help in understanding how future systems will be implemented and drive requirements. Visualization also allows both the operator and the engineer to understand how the system is designed and how it is represented in the real world. This can significantly reduce the amount of time, money, and effort required to design a system. Hence, visualization can be used in systems engineering analysis.

The military has invested large sums of money to create modeling and simulation environments that use expensive high-end work stations and proprietary visualization engines. This investment started in the 1970s with computer generated imagery (CGI). CGI replaced physical model boards used to represent terrain and objects in first-person visual scenes, in military simulators. The drawbacks of these systems were quickly identified: "While much more flexible and typically lower in life cycle costs, CGI systems lacked the detailed realism of the model boards they replaced [1]." Over the years, PC and COTS (commercial off the shelf) based technologies replaced CGI systems to provide more resolution and fidelity for the military. Recently, advancements in the commercial gaming sector have made it possible to produce equivalent visual scenes on
relatively inexpensive laptops. The shift from legacy simulation systems to the latest progression in video game technology has more become evident.

First of all, video games are less expensive to purchase than legacy simulation systems. As seen in Table 2.1, video games cost between $12 and $50 to buy off the shelf. Video games are created for personal computers or proprietary game consoles. PC games can be played on a low-end desktop or average quality laptop. If a PC is unavailable, a game console such as the Microsoft X-Box or Sony Playstation 2 can be purchased for less than $200 [4]. The game consoles’ CPUs are as powerful as the average PC and are architected for maximum performance and ease of play with built-in memory, network connectivity, and user controllers.

Table 2.1 Defense Simulations vs. Video Games [4]

<table>
<thead>
<tr>
<th></th>
<th>Defense Industry</th>
<th>Gaming</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operator Requirements</strong></td>
<td>Many Degreed Engineers</td>
<td>Any 15 Year Old</td>
</tr>
<tr>
<td><strong>Training</strong></td>
<td>Specialized Skills Development</td>
<td>None</td>
</tr>
<tr>
<td><strong>Computer Costs</strong></td>
<td>$5,000 to $300,000</td>
<td>$300 to $1500</td>
</tr>
<tr>
<td><strong>Software Costs</strong></td>
<td>High (10s or 100s of K$s)</td>
<td>$9 to $49</td>
</tr>
<tr>
<td><strong>Software Complexity</strong></td>
<td>Millions of Lines of Custom Code</td>
<td>Reusable Modular Code/OOP</td>
</tr>
<tr>
<td><strong>Network Setup</strong></td>
<td>Extensive</td>
<td>On the Fly/On Demand</td>
</tr>
<tr>
<td><strong>User Interface</strong></td>
<td>Cumbersome</td>
<td>Intuitive</td>
</tr>
<tr>
<td><strong># of Sim Entities</strong></td>
<td>10s to 100s</td>
<td>1000s</td>
</tr>
<tr>
<td><strong>Representations</strong></td>
<td>2D and Limited 3D</td>
<td>3D</td>
</tr>
<tr>
<td><strong>Human Simulation</strong></td>
<td>Limited if Any</td>
<td>Interactive 3D w/ Realistic Motion and AI</td>
</tr>
<tr>
<td><strong>Terrain and Scenario Generators</strong></td>
<td>Extra/Geo Referenced</td>
<td>Included/Non-specific</td>
</tr>
</tbody>
</table>

Also, there is a significant amount of investment in the development of video games. The DOD has realized that budgets for the development of commercial games far surpass the Army’s science and technology budget. For example, Microsoft has invested more than $2 billion in the development of the X-Box gaming system compared to the
$1.6 billion for the Army's R&D [2]. The military, along with the rest of the world is taking a serious look at how to leverage commercial video game technology for serious applications.

Video games have become sophisticated, realistic, and complex simulation systems. In the past 10 years, there has been a shift from the classic flight and driving simulation to a first-person combat simulation that provides players the ability to interact with other gamers, artificially intelligent bots, and simulation systems - all connected over robust, distributed networks. Users can also use the networking capabilities to connect to distributed simulations or high fidelity algorithms to represent the virtual world as realistically as possible. To increase adoption, new games can run on a wide variety of platforms including personal computers, game consoles, handheld game players, palm computers, and, more recently, cell phones.

The technology provided by video games can help us analyze our current and future concept of operations for systems engineering purposes. Engineers can do trade studies on operational architectures and policies that support very complex operations. Operation Flashpoint is an example of a game that enables users to dynamically select policies that modify scenarios and analyze the effects on operations:

“For example, in Operation Flashpoint – a game involving synthetic soldiers operating in military units, an author can pick a policy which is the behavioral goal for a whole unit. When creating a new scenario, the author picks an overall policy which is the behavioral goal for the unit (such as engage at will, seek and destroy, or hold position) as well as a manner (such as cautious, aggressive, or reckless) [6].”

In order to create the serious games-based simulator, we must understand the anatomy of video game engines.
2.2 ANATOMY OF VIDEO GAME TECHNOLOGY

For the purposes of this paper, I will focus on PC video games for integration with engineering analysis tools. Video games consist of the following features that make their use extremely appealing for systems engineering and analysis: Rendering, World Building, Character Modeling, Animation, Physics, AI, Sound, Special Effects, User Control, Navigation/Camera, Graphical User Interfaces (GUI), Heads Up Display (HUD), Networking, Scripting, and Development Tools/SDK. Below is a discussion of the major features that are relevant to selecting a video game for integration into the serious games-based simulator.

2.2.1 THE GAME ENGINE

For the serious games-based simulator, I will try to find a game engine to support the visualization. The game engine is the “umpire” of the entire video game environment:

“In today’s modularly constructed games, the game’s engine refers to that collection of modules of simulation code that do not directly specify the game’s behavior (game logic) or game’s environment (level data). The engine includes modules handling input, output (3D rendering, 2D drawing, sound), and generic physics/dynamics for game worlds [4].”

Fig. 2.1 is a diagram of a modular video game architecture; it shows that the game engine interfaces with the other building blocks of the game.
Fig. 2.1 Modular Game Architecture

The top level represents the three dimensional virtual world that allows the user to experience the missions and scenarios played in the game. Later, I will discuss how editors can allow a user to modify the virtual worlds and edit the levels of the game to suit their needs. The game code handles the high level physics, artificial intelligence or player behaviors, networking, and other mechanics of the game. The rendering engine is the proprietary code given to us by the game developers and is not open for the community to modify. It incorporates all of the complicated code needed to efficiently identify and render the player's view from a complex 3D model of the environment [4]. The networking layer enables players to connect with each other and interact in the same environment. Currently, most video games operate with a client/server architecture, where the game is hosted on a server. Later, I will discuss this in greater detail and introduce other options such as High Level Architecture (HLA) or Distributed Integrated Simulation (DIS). The graphics drivers facilitate communication between the operating system of the client computer and the game engine. They also translate generic requests from the rendering engine to the underlying graphics library, using APIs such as DirectX, OpenGL, and others. The graphics drivers interface with extremely powerful hardware
discussed below that allows video games to operate at rapid speeds.

A crucial feature of the game engines is that almost all code, except the proprietary graphics engine, can be open source. Under its GNU license, this code may be freely distributed, copied, and modified. The power is in the ability to allow users to do most of the leg work involved in developing and testing capabilities.

2.2.2 THREE-DIMENSIONAL VISUALIZATION ENVIRONMENT

The most popular video games developed within the last 10 years have three dimensional visualization environments rather than the two-dimensional graphics seen in the past. We live in a world of three-dimensional objects, and there is a human desire to represent and visualize spatial relationships among these objects in a systematized way. The conceptually simple idea of measuring the size of an object and distance between objects yielded the development of various branches of mathematics such as geometry and trigonometry. The link between physics and mathematics of image formation can be exploited in the development of computer image formation [39]. These mathematics are used heavily when computing the three dimensional representation of the world in a video game. The next section will discuss how software and hardware enable the creation of realistic images of computer-generated, three-dimensional mathematical objects.

2.2.3 RENDERING/CPU POWER

Video cards have enabled video games to become very fast and efficient simulation engines that are able to run on low cost machines with average performance. The video card, sometimes referred to as a Graphics Processing Unit (GPU), is a dedicated rendering device located on personal computers and game consoles. Recently developed GPU hardware is achieving supercomputer-level performance for the average
consumer at an affordable price. The GPU is similar to a CPU; however, it executes an architecture specific for maximizing computer graphics output. Increasing growth in GPU processing power relieves the CPU processing required for visualization in the past: "While CPUs technology has sustained exponential growth in speed (Moore’s Law), specialized GPUs have been improving even faster. Visualization used to be handled by tapping into spare CPU cycles, and eventually to specially tuned graphics processors [5].” Now, the GPU handles the labor intensive processing involved in rendering the complex 3D graphical visualization. This allows the CPU to handle other tasks associated with running a video game. This technology has taken video game graphics to a new level, able to run seamlessly on a laptop. For instance, my Dell® Inspiron laptop has a built-in NVIDIA® GeForce® 7800 graphics processing unit that delivers over 30 million polygons per second, 13.2 billion pixels per second fill rate, and has 256 MB of ram. This is unparalleled horsepower and enough ram to out perform a high end desktop from just 6 years ago. What does this mean? The advancements in GPU technology have allowed video games to reach unprecedented levels of visual complexity and high definition output while allowing the CPU to handle higher fidelity physics and better artificial intelligence.

2.2.4 EDITORS/MODS

Another attractive feature the commercial game industry offers is the ability for a user to edit content, build a new world, and create a character without much difficulty. The act of modifying a game is called modding, where players often use editors to create their own virtual worlds, (scenarios, maps, levels, or boards) modify existing objects or create new objects, and alter the way the game behaves or rules of the game. Content
authoring is performed using a vendor supplied tool that is sold with the game. Non-programmers can now build their own models, modify physics, and add new behavior because of the intuitive interface and available tutorials. During recent years, video games such as Unreal Tournament and Quake have come with editors that have increased their popularity both in the gamer community and in the research community. Also, the Internet has become the perfect medium for developers and gamers to freely distribute mods and tutorials. Users can visit forums and support sites to gain access to the latest updates. This activity increases the popularity and creativity of game releases to the point where the original developers of the game purchase the mods from users. These mods turn into new releases, such as “Counter Strike”, that gain revenue for the developers and spark new interest in the game.

2.2.5 SCRIPTING

Scripting is a major module built around a game engine to allow non-programmers to create character behavior and synthetic entities, modify/tweak physics parameters, and dynamically add new objects to the game world. Scripting is considered a high level language that translates into low level executable code for the game engine to understand. The use of high-level scripting languages allows designers to interface with game engines without knowing detailed programming languages or how to program. This allows inexperienced programmers to rapidly modify the way the game plays. Programmers can also modify a game’s behavior by interfacing through APIs that allow them to create complex models external to the game. They could use a programming language like C to write high fidelity models of human or system behavior. They can attach this code to the game engine using an API that executes commands with game
specific scripting language. Scripting languages bridge the gap between system experts and programmers of video games. Scripting languages such as Python and variants on C are used to interface with a variety of game engines. Python was adopted by the video game development community a decade ago because it was an open source, dynamic, high level, object-oriented programming language. A variety of current commercial video games, such as Battlefield 2, allow the use of Python; however, many use their home grown proprietary language supported by their engine.

Proprietary languages are provided by video game developers and accompanied by comprehensive tutorials and documentation. These languages are created by the developer for various reasons including customization and optimization for the game engine. Examples of proprietary scripting languages include UnrealScript for Unreal Tournament and QuakeC for Quake.

2.2.6 NETWORKING

Using the same architecture implemented by DIS and HLA, the first video games provided networking play using a peer-to-peer schema that only allowed a limited number of players to interact in the video game. Specifically, this becomes an $n \times n$ problem that restricts scalability for player connections. (See Fig. 2.2)
Peer-to-peer schema, like HLA RTIs, have been considered unsuitable by game developers because they need real-time performance [16]. Other issues with peer-to-peer communication are lack of support for websites, player authentication, and a central database in which to store data.

For security, bandwidth, and other reasons, game developers realized that they needed a client/server architecture to support a multi-player game. In this type of architecture, only \( n \) clients generate \( n \) communication links. (See Fig. 2.3)

![Client / Server schema](image)

**Fig. 2.3 Client / Server schema**

The remote game server (green) is a separate process, usually on a different machine, that maintains information about the game state on whichever virtual world it is supporting at the time. It communicates with the game clients used by the players to maintain global information about shared environments, player interactions (such as damage) and synchronization information. The remote game server provides many benefits: “The remote game server provides game logic and game security (by removing hacker access to central game logic), and can manage overall game messaging traffic from a central point. This makes the communications order \( n \) and can scale communications linearly with the number of players that connect [11].” The client (red) preserves relative performance even as clients join the game because it also performs complex graphics
computations, using its own GPU, needed to display individual views. This allows the server to handle the laborious processes associated with the game state.

The most common mechanism used for communications between clients and servers in multi-player games are sockets. A socket is an endpoint of a two-way communication link between two programs running on a communications network. A socket is bound to a particular port number on a networked computer. The client and server programs write message packets to the socket for delivery to each other. Sockets guarantee the delivery of game data packets across the Internet. With socket-based interfaces and the addition of APIs, user coded applications can be leveraged to connect simulated robots (bots), high fidelity behavior algorithms, and simulated systems to the video game environment. I will discuss a technology later called Gamebots which connects to Unreal Tournament to allow engineers to test, play, and evaluate robotics.

2.3 CURRENT VIDEO GAME ENGINES

There are multiple video games on the market with game engine modularity that would be applicable to integrate into the serious games-based simulator. There are some open source game engines that enable video games to be developed with and sold for a licensing fee. Open source engines are typically developed as projects by students or a loose, diverse community of programmers and game enthusiasts. The business model is similar to that of the original Linux community – to make useful software freely available. Examples of these engines are Crystal Space, NeoEngine, FLY3D, and the original Quake engine. There are other engines that require the user to purchase a license because they may contain more functionality and support. Below is a discussion of the notable video games and or engines. I will explain what value they bring and whether or
not they fit the criteria for my serious games-based simulator study. Table 2.2 shows a comparison of the video game engines that I evaluated for this simulation study.
<table>
<thead>
<tr>
<th>Name</th>
<th>Vendor</th>
<th>Released</th>
<th>Type</th>
<th>Cost</th>
<th>Editor</th>
<th>Scripting Language/API</th>
<th>Modularity</th>
<th>Multi-player Online Play</th>
<th>Chosen: Reason</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battlefield 2</td>
<td>EA Games</td>
<td>2006</td>
<td>First-person military simulation</td>
<td>40</td>
<td>Yes</td>
<td>Python</td>
<td>unknown</td>
<td>Yes: 64 people</td>
<td>No: Too New</td>
</tr>
<tr>
<td>Delta 3D</td>
<td>NA</td>
<td>2004</td>
<td>Multi-purpose</td>
<td>Free</td>
<td>Yes</td>
<td>C++/OpenGL</td>
<td>Yes</td>
<td>Yes</td>
<td>No: Not enough time to learn</td>
</tr>
<tr>
<td>Torque</td>
<td>Garage Games</td>
<td>2001</td>
<td>Multi-purpose</td>
<td>$150.00-$749.00</td>
<td>Yes</td>
<td>C++ and Torque Script/OpenGL and DirectX</td>
<td>Yes</td>
<td>Yes</td>
<td>No: Not enough time to learn</td>
</tr>
<tr>
<td>Quake</td>
<td>Id Software</td>
<td>1996</td>
<td>First-Person Shooter</td>
<td>40</td>
<td>Yes</td>
<td>C/OpenGL</td>
<td>Yes</td>
<td>Yes</td>
<td>No: Coded in C</td>
</tr>
<tr>
<td>Unreal Tournament 2004</td>
<td>Epic</td>
<td>2004</td>
<td>First-Person Shooter</td>
<td>50</td>
<td>Yes</td>
<td>Unreal Script/OpenGL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes: Knowledge, rapid prototyping, academia use</td>
</tr>
<tr>
<td>Americas Army</td>
<td>Ubisoft and U.S. Army</td>
<td>2002</td>
<td>First-Person Shooter</td>
<td>Free</td>
<td>No</td>
<td>Unknown</td>
<td>Unknown</td>
<td>Yes</td>
<td>No: Unknown Support for modification</td>
</tr>
</tbody>
</table>
2.3.1 BATTLEFIELD 2 BY EA GAMES

Battlefield 2, or BF2, is a modern warfare video game that was released in 2006. It is very much a first-person military simulation that gives the user the ability to command tanks, fighter jets, helicopters, cars, trucks, etc. In Battlefield 2, players will choose to fight for one of three military superpowers: the United States, China, or the newly formed Middle East Coalition. Players are armed with the latest weaponry and can operate more than 30 vehicles that are true to real world physics. The necessity of communication amongst players and realistic strategy makes this game popular. Players assume roles and have specific orders that make their unit operational. The teams that follow military tactics, techniques, and procedures (TTPs) will ultimately perform better in the game. Along with state of the art artificial intelligence (AI) and high resolution graphics, BF2 supports up to 64 people playing within a single online game. This is truly one of the most advanced video games ever released. BF2 encourages modding and offers a free editor on their website. It also supports python scripting for dynamic access and collection of information, user administration, and tweaks to gameplay. Because Battlefield 2 is such a new release, there have been very few support sites, forums, tutorials, and documentation specifically addressing modding and scripting. Thus, I have not selected BF2 for this study despite the potential for BF2 to become a very powerful research engine, I have not selected BF2 for this study due to such implications.

2.3.2 DELTA3D

A well-supported open source project, Delta3D, is a full-function game engine appropriate for a wide variety of modeling & simulation applications including training, education, visualizations, and entertainment. Delta3D users consist of the U.S. Military,
numerous research universities, hobbyists, and commercial companies. It is an open
source engine and its modular design integrates other well-known open source projects
such as Open Scene Graph, Open Dynamics Engine, Character Animation Library, and
OpenAL. Delta3D comes with an API that allows users to access the important
underlying components and optional, low-level functionality. It renders using OpenGL
and imports a whole list of diverse file formats (.flt, .3ds, .obj, etc.). Currently, Delta3D is
developed and tested on Windows XP using Microsoft Visual Studio .NET (7.1) and
Fedora Core 4 using gcc 4.0.0. It supports HLA and game-style client/server networking
for multi-player interaction or distributed simulation integration. It includes the STAGE
editor and the ability to load DTEDs for terrain. It also has a support web site with
tutorials and forums.

Delta3D is a powerful game engine that has the potential to gain lots of
momentum in the research community due to previously stated features. However, I did
not choose to use Delta3D in this project due to lack of knowledge of its capabilities and
scarcity of documented use in research and white papers.

2.3.4 TORQUE, BY GARAGE GAMES

The Torque Game Engine (TGE) is a modified version of a 3D computer game
engine originally developed by Dynamix for the 2001 FPS Tribes 2. The Torque engine
has since been available for license from GarageGames to independent and professional
game developers. The TGE gives you access to its C++ source code, built in tools, and its
own scripting language called TorqueScript. It supports many industry standard content
creation tools for 3D modeling and animation, and comes standard with exporters for 3D
Studio Max on the high end and Milkshape on the low end. It includes an editor for
terrain generation and object placement. Torque has two major drawbacks; its poor
documentation and level of support are incompatible with the tight development time
frame [10].

2.3.5 QUAKE

Quake is a first-person shooter computer game that was released by id Software
on July 22, 1996. At the time, Quake provided a unique experience because it offered
many innovative features: “It can be said that the original Quake game pushed most PC
hardware to its limits, due to the never-before-seen features it offered: complex textured
3D environments, polygon-modeled enemies with certain intelligence, and the like [15].”
Quake was one of the first games to provide the client/server architecture for networking.
Quake also extended its programmability to provide the first game-independent game
engine providing both a level editor for changing layouts and QuakeC, a byte-compiled
scripting language, for changing behavior in the simulation. Quake was not chosen
because, since it uses the C language, it is not object-oriented. As discussed in the next
section, object-oriented languages are preferred because they offer greater flexibility and
maintainability in programming.

2.3.6 UNREAL ENGINE 2 / UNREAL TOURNAMENT BY EPIC

Unreal is a first-person shooter computer game developed by Epic Games and
published by GT Interactive on May 22, 1998. For the price of $40, you can purchase
Unreal Tournament 2004 Editors’ Edition (UT2004), which includes the complete game
and bonus discs with the Unreal Editor, Server, and hours of video tutorials. The game
allows for 10 modes of gameplay where you navigate a futuristic combat environment
with the objective being to “kill or be killed”. You have the choice to play single player
or multi-player in an easy to use networked environment. Single player allows you to compete against bots (artificially intelligent robots in the game that play with you or against you) while multi-player allows you to configure a game server and host your own game on a local area network. There is also server administration so you can set server rules and grant certain users access/authentication. Communications amongst players and bots include text chat and voice chat. In most games, communication between players and bots is quite limited, with the dialog consisting of a few scripted phrases, but with UT2004, speech recognition capabilities and software development kits (SDKs) allow players to communicate more dynamically.

The Unreal Tournament Game Engine has been well adopted by the research community as an easy to learn development and test environment for a variety of applications [USAR and UTSAF: University of Pittsburgh, 2003] [ACT-R: Carnegie Mellon University, 2003]. Also, its broad range of tutorials and community forums puts information about modding or user support at your fingertips. The Unreal Editor (UnrealEd) is a sophisticated graphical development environment that provides a very user friendly interface for creating characters, building models, modifying actor behavior, and building 3D virtual worlds. It also provides a variety of specialized tools including the Karma physics engine and a skeletal animation system that simplify the detailed tasks of modeling physical processes [17]. If you do not care for this editor, there are a variety to choose from. Secret Level, Alias, and Epic Games, Inc. have created Maya Complete / Maya Unlimited versions of the user-friendly UT2004 Maya-to-UnrealEd exporter to facilitate export of static meshes, skeletal meshes and animations directly into the UT2004 editor. This allows modelers familiar with Maya to work easily with the Unreal
Tournament Game Engine.

Unlike Quake, which has descended from a succession of C language implementations, the Unreal engine is strictly object-oriented. This pays big dividends through mutators and other programming constructs that allow a programmer to make robust changes in game behavior without requiring detailed knowledge of the involved code [7]. Some have tried to modify the Unreal game play by working with the core C++ code, but quickly found that they could accomplish most of their modifications quicker using UnrealScript. The scripting language is easy to use and well documented much like the well-designed UnrealEd development environment. Developers can extend the existing game types, or implement new ones using UnrealScript. UnrealScript is a C++ based scripting language that handles all game logic and object interaction under Unreal Tournament while the main game engine handles the hardcore work like rendering scenes and simulating physics. UnrealScript can also be used to build small mutator scripts that implement small tweaks, such as adding or replacing world items, adjusting physics parameters, or changing item effects. Specifically, the Unreal Engine is an excellent platform for rapid prototyping because of the well documented scripting language and editing tools that come with the game. Later in this paper, I will discuss Unreal Triage as an extremely good example of how UT2004 can be used in research, development, and experimentation.

2.3.7 AMERICA'S ARMY

The Unreal Engine is also used in over 20 current games including America's Army. America's Army (AA) was developed as a promotional and recruitment tool for the U.S. Army. Since its release on July 4th, 2002, it has become one of the most
successful PC games currently on the net, and it's free. AA has seen numerous updates and offers gamers the most realistic squad tactics experience yet. Currently, AA has over 5.5 million registered users.

In addition to the huge commercial success of AA, many organizations have expressed interest in extending applications of AA in a variety of ways: education, values & cultural awareness, materiel prototyping, and operational support - to name a few. As a result, AA has created development teams that allow extensions to the game engine for scientific research and experimentation. The America's Army Future Applications Team was formed in cooperation with the Armaments Research and Development Center (ARDEC) in Picatinny, New Jersey. This highly experienced simulation team is using and promoting AA in research, development, and training. Their emphasis is focused on new and developmental weapon systems. Using the existing high fidelity models of AA, they would like to place virtual weapons in the hands of soldiers and engineers for evaluation in a virtual battlespace before real-world production starts.

I chose the Unreal Tournament 2004 Engine for use in the serious games-based simulator for the following features:

- modularity of the game engine;
- networking and APIs;
- easy to use editor and scripting language with available tutorials;
- the wide variety of applications and proof of concepts in the research; development, and experimentation community.

2.4 SERIOUS GAMES

Recently, a number of organizations and conferences have sprung up that are focused on educational and training applications of video game technology. Some examples include the Serious Games Summit [Vargas 2004 and Serious 2005], the MIT

The Serious Games Summit has recognized the importance of video games and their application to real world problems. The focus of the Serious Games Summits is on game technology within non-entertainment sectors: training, engineering analysis, experimentation, etc. The objective is to assess the effectiveness of serious games in achieving goals, discovering innovative and technological breakthroughs. The results are cost-effective development and regulations for serious games, and future prospects and market potential of game development.

These types of conferences and summits bring together game developers, buyers, and industry professionals in order to exchange ideas and advance the state of the art of serious games development. This is very important because it shows how the gaming community is sharing information and ideas about technology. There are many unique and creative ideas regarding video game technology to solve real world problems; however, there does not seem to be solutions that involve DESs that ground the simulation in real-world analytical architectures. An analytical architecture backbone will allow a simulation environment to analyze architecture behavior, system functionality and operational activity flow as the scenario of the video game plays out.

2.5 WHY DISCRETE EVENT SIMULATION?

Gaming technology provides user interfaces, interaction mechanisms, and human modeling capabilities that support the creation of immersive experiences in virtual worlds. Simulation technology can complement game technology by offering higher
fidelity, validated models: “Simulation technology enables the construction of technically correct, dynamic models of organizations, systems, and processes. The models, once validated, can be used for supporting decisions for design and operation of the systems to achieve desired performance [3].” Also, validated models and simulations already exist and game engines would ideally be able to re-use them.

There is a need to apply some systems engineering rigor and higher fidelity modeling to the video game environment if I intend to use it for a serious game-based simulator. Dennis McGrath, from Dartmouth College, has experienced this need working with the Unreal game engine:

“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 [23].”

The integration of a high fidelity engineering algorithms and models of systems into a video game environment is evaluated as part of this thesis.

Specifically, I evaluate the use of DESs to model system architectures and leverage existing simulations for a scenario involving first responders to a mass casualty event; which will be described in detail in a later chapter. One problem with video games is that there are no real world architectures to monitor the execution of events [18]. To understand this problem, I need to look at the way a video game executes its main function. A video game execution loop first tests the user interaction by listening to some type of input device (usually a keyboard or hand-held controller). Next, the video game will simulate the environment in that time step by querying each object involved for
information about its current state. It will compute a time step or a tick and render the current scene (usually three-dimensional graphics). This main loop supposes that video games use a continuous simulation scheme where the simulation cycle is defined as the time elapsed in a run of the program main loop. This can be a disadvantage because the simulation and rendering are highly coupled (in general, each world evolution always requires a full world rendering). The simulation events are artificially synchronized to match the sampling period; therefore, they may not be executed at the moment they happen. Also, the sampling frequency depends on topics that can change during the game, such as available computer power, world complexity, other active tasks in system, network overload or current simulation and rendering load. Thus, the sampling frequency is variable and not predefined. Events are not time ordered. Events are executed in the order in which the objects management structure is accessed. It is possible that the simulation be erroneous due to: the disorderly of execution events and the execution of canceled events [18].

2.6 DEFINITION OF DISCRETE EVENT SIMULATION

Simulation is a widely used and increasingly popular method for studying complex systems. DES is used to model, analyze, and understand complex systems and scenarios by computing the time associated with real events and real system execution. It models the system as it evolves over time by a representation in which the state variables change instantaneously at separate points in time. A discrete event is something that occurs at an instant in time that may change the state of the system. I will use DES to manage the execution of events throughout the video game simulation and make sure that they happen in the correct order and according to the structure of an architecture.
Such a DES will execute these events in a sequential order, specified by the architecture, over a certain period of time. It will perform the system events management and apply a discrete event scheme for the video game in the serious games-based simulator. As events are executed, transactions or entities (units of traffic) flow through the system. This is important because it allows us to model activities performed within an operational view of an architecture and the system functions performed within a systems view of the architecture. The dynamic execution and visualization of these views is an example of executable architectures. Executable architectures enable the serious games-based simulator to be directly traceable to the architectures. Although, for this study, I do not implement an automated or semi-automated way to translate the architectures into a DES.

Within each node of the architecture, activities and functions can be modeled as two distinct and complementary events. If I associate a time value with each discrete event, then I can model the duration of the activities as the difference between the times associated with the events marking the beginning and the end of each activity. From this information, I can perform timeline analysis to determine if the architecture behaves as expected in accordance with time requirements. Resources can be utilized during the execution of the activity to simulate, for example, a human completing a task or leveraging a system during the activity. I can also determine utilization of resources and perform optimization with the DES. The same modeling and simulation formulation can be done from a systems view. A DES can also be used for queueing and bandwidth analysis for each node in the architecture.

Assuming that simulator rendering can run faster than real time, another benefit of
using a DES is the ability of the simulation to execute the events in faster than real time (wall clock time). This is done in order to compute the physical time that would have occurred if operating in real time. In order to achieve this, the DES uses an event queue to sequentially order the events. This guarantees that events will be handled in the correct order.

### 2.7 ADVANTAGES OF DISCRETE EVENT SIMULATION

The advantages of DES include:

- Most complex, real-world systems with stochastic elements cannot be accurately described by a mathematical model that can be evaluated analytically.
- Simulation allows one to estimate the performance of an existing system under some projected set of operating conditions.
- Alternative proposed designs (or alternative operating policies for a single system) can be compared via simulation to see which best meets a specified requirement.
- In a simulation I can maintain much better control over experimental conditions that would generally be impossible when experimenting with the system itself.
- Simulation allows us to study a system with a long time frame in a compressed time, or alternatively study the detailed workings of a system in expanded time [26].

### 2.8 DISADVANTAGES OF DISCRETE EVENT SIMULATION

The disadvantages of DES include:

- Each run of a stochastic simulation model produces only estimates of a model's true characteristics for a particular set of input parameters. Thus, several independent runs of the model will probably be required for each set of input parameters to be studied. For this reason, simulation models are generally not as good at optimization as they
are at comparing a fixed number of specified alternative designs. On the other hand, an analytical model, if appropriate, can often easily produce the exact true characteristics of that model for a variety of sets of input parameters. Thus, if a “valid” analytical model is available or can be easily developed, it will generally be preferable to a simulation model.

- Simulation models are often expensive and time-consuming to develop
- The large volume of numbers produced by a simulation study or the persuasive impact of a realistic animation often creates a tendency to place greater confidence in a study than is justified. If a model is not a “valid” representation of a system under study, the simulation results, no matter how impressive they appear, will provide little useful information about the actual system [26].

2.9 CHOOSING A DES

The criteria for choosing a DES for the serious games-based simulator will be based on a few factors such as cost, ease of use, maturity, real-time execution, API, Parallel Process flows, extensibility, documentation and support, and my knowledge and experience with the tool. There are many DES software products on the commercial market; however I will focus on a select few.

2.9.1 HYPERFORMIX WORKBENCH

The first alternative investigated was HyPerformix Workbench. HyPerformix Workbench is a visual simulation environment used to model complex systems for performance analysis and functional verification. It can model anything that consumes resources and determine the impact of the net result. Workbench is particularly well suited for specifying and evaluating complex systems involving a high degree of
concurrent processing. Workbench has been engineered to help the user construct models that map the system architecture being evaluated. Configurable modeling objects such as queues, delays, arcs, and resources are represented by graphical components, allowing the user to create a model of arbitrary abstraction and complexity in an easy, graphical manner. Workbench has very thorough documentation and an API; however, the tool itself is very software intensive. In order to get a model functioning, you will need to write code and have knowledge of programming structures. The benefit of programmability is extensibility and scalability. You can easily integrate other simulators and algorithms to increase fidelity of the model. Also, the software is compiled instead of interpreted which yields better performance during execution. Workbench allows the user to animate the model and view statistics during runtime. Even though I have a detailed understanding and knowledge of Workbench, I was not able to use it for this project because of the cost of the tool.

2.9.2 EXTEND

The next alternative evaluated was Extend. You use Extend to create dynamic models from building blocks, explore the processes involved, and see how they relate. Blocks are the basic model-building components that represent some part of the process being modeled. They contain unique procedural information and are grouped into libraries according to function. You create an Extend model by dragging blocks from a library onto a worksheet, connecting them, and then entering the appropriate data in the dialog. For visual clarity, and for reuse in other projects, you can encapsulate sections of the model into hierarchical blocks. Once you are satisfied that your model accurately represents the system, you can explore alternatives by changing parameter data inside the
model's blocks, or leverage the global array blocks to read from Excel spreadsheets. You can also perform sensitivity analysis on various parameters or use the optimizer to maximize or minimize important variables. You are able to get a full evaluation copy for 160 days or the software can be purchased for roughly $4,000; this is a big improvement over Workbench's $40,000. Extend has tutorials and technical support, including a detailed API for development. The ability to drag and drop blocks and reuse them leads to faster development than Workbench; however, customization is more difficult because you need to use their proprietary Pascal-based language called ModL or integrate a DLL. Extend is easy to use but tends to have an electrical engineering look and feel. This can be confusing to someone looking at the model if they are familiar with the architecture being modeled but not the Extend simulation environment. Because of this, the developer needs to create hierarchical blocks to simplify the look of the simulation. Another drawback is that in animation mode, you cannot view the name of the messages or transactions as they flow through the model, rather you need to create an image and overlay it on the entity. Also, I did not have a full license for Extend; therefore, I did not choose Extend for this project.

2.9.3 ARENA BY ROCKWELL AUTOMATION

Arena by Rockwell Automation is an easy to use DES, similar to Extend, that is used heavily in academia. DES courses at universities can use the student version for free. Arena Professional Edition is used to create customized simulation products, i.e. templates focused on specific applications or industries. With Arena, users can develop custom templates that consist of "libraries" of modeling objects that make it significantly easier and faster to develop models that require repeating logic. Arena includes: a Basic
Process Template, an Advanced Process Template, an Advanced Transfer Template, a
Blocks Template, an Elements Template and the additional functionality required to
create templates. Arena has many benefits such as a free student license that provides
access to necessary libraries in order to create the average simulation. However, the
student version has several limitations that would force you to buy a full license.
Limitations include a 150 entity limit within the model and a 10 minute simulation time
limit when running in real time mode. Like Extend, Arena allows the user to drag and
drop process blocks into the model, enabling faster development. Arena has well
documented tutorials and college level textbooks with examples and sample projects.
Also, the API supports Visual Basic Application programming and user-coded DLL
linking. This leads to easy integration with other simulations and high fidelity models.
However, Arena is an interpreted simulator which may cause performance problems with
large models. Due to Arena's free license, prior knowledge of the tool, and availability of
support resources from Rockwell Automation and Old Dominion University, I have
chosen Arena as the DES for use in the serious games-based simulator.

2.10 HOW COULD I GET THE DES INTEGRATED WITH A GAME ENGINE?

2.10.1 ARENA DLL

Arena includes a feature called Usercode, which is a communication capability
involving DLLs. This option allows you to incorporate user-coded routines into Arena for
use with a particular model. If selected, you must specify the name of the dynamic link
library (.dll) file that you wish to include. Arena has provisions for linking in user-coded
C routines. The C functions listed in the User-Coded Routines section provide the link
from SIMAN models to user-written code. These routines provide great flexibility for
writing initialization, ending, and logic routines.

2.10.2 ARENA RT MODE

Arena RT allows users to execute the DES model in real-time and send commands via the user coded dll to external processes. It increases the capabilities of Arena by adding the following features:

- It allows you to run a simulation model in execution mode. When in this mode, Arena can coordinate simulation logic with the external process of a real system. The external processes and Arena communicate via a messaging system, whereby entities in the Arena model send messages to the external applications to indicate simulated tasks, and the external applications send "message responses" back to Arena to indicate the tasks have been completed. Unsolicited messages can also be sent to Arena to indicate special events (e.g., the arrival of raw material or customer orders).
- It allows you to set the speed of Arena’s simulation clock as a factor of the real-time clock of the resident operating system.
- The above features might be leveraged when developing simulation-based applications for:
  - Gaming or training,
  - Controlling or monitoring the real-time status of a system,
  - Testing the control software of a system.

Arena RT’s evaluation mode allows a model to be run with a real-time clock and/or in execution mode for a maximum of 10 minutes. To run a model with these features for an unlimited time, you must purchase Arena RT with full commercial authorization.

2.10.3 GAMEBOTS
In order to communicate between the DES and video game environment, I will use an API developed by researchers at the University of Southern California’s Information Sciences Institute. They developed this package with the goal of making the game a suitable test environment for artificial intelligence software. Gamebots is a modification to Unreal Tournament that enables the user to control bots through a simple API:

"Gamebots is a modification to the Unreal Tournament game that allows bots to be controlled through a normal TCP/IP socket (Kaminka et al. 2002). Gamebots talks to the game engine directly, and opens its own networking sockets. . . . With a simple text-based TCP/IP protocol Gamebots can be used to create and manipulate bots in an Unreal Tournament instance. Because the full range of bot commands and Unreal scripts can be accessed over this connection, Gamebots provides a more powerful and flexible entry into the simulation than the player interface [17]."

![Gamebots System Diagram](image)

Gamebots uses a high fidelity physics engine called the Karma Engine developed by Math Engine to provide physics modeling, rigid body dynamics with constraints and collision detection. Karma is the new physics engine included in Unreal Engine 2, part of UT2004, and it runs alongside the regular physics engine. It uses specialized algorithms to allow the rendering of the 3D graphics at the appropriate speed while at the same time
calculating the Newtonian physics for each object [17].

Gamebots 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"). Upon creating a BotConnection, the Gamebots client sends 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.

Gamebots will not only be used to explore artificial intelligence, it will be extended to interact with Arena DES to access architectural information and other distributed simulations. "Within Unreal there are two ways to simulate physical events: scripted behavior (animation), and bespoke solutions. Scripted behaviors control the movement of an object according to a predefined sequence of events. Bespoke solutions generate movements by applying the appropriate mathematical equations. These two approaches can be used together or switched according to context [17]."

Gamebots will enable communication between nodes in the Arena executable architecture and the video game objects in Unreal Tournament scenario. As transactions flow through the architecture, the code within the nodes will call commands using the Gamebots API to spawn events in the video game. Gamebots has been implemented in numerous research domains such as Urban Search and Rescue robotics and Homeland Security Training applications.

There is a desire to build a serious games-based simulator to analyze real world scenarios. Specifically, emergency management and disaster response is a topic of interest that is in need of systems engineering tools and modeling and simulation
capabilities to evaluate a variety of events. In the next chapter, I will review the modeling and simulation technologies that can be found today in this field of study.
CHAPTER III

RELATED WORK: MODELING AND SIMULATION IN EMERGENCY MANAGEMENT AND RESPONSE

There are many modeling and simulation tools that have been developed for emergency management and response applications. Various applications exist such as first responder serious gaming environments, emergency room and hospital process models, plume simulation for chemical, biological, radiological, nuclear, and explosive dispersions, building fire simulations, and natural disaster simulation. Usually, these simulations are comprised of high fidelity models and algorithms for specific applications. Below is a discussion of modeling and simulation applications currently available.

3.1 UNREAL TRIAGE

Unreal Triage is a SEERS simulation of a mass casualty incident based on a modification (mod) of Unreal Tournament 2004. SEERS was a two-year research project at Dartmouth College funded by the Department of Homeland Security. It is focused on the use of game technology to create synthetic environments for emergency responders at all levels. Like other serious game projects, SEERS realizes the utility of game engines for serious (mass casualty response) applications, but it is unique because the goal is to integrate games with "real" emergency response information systems. The simulation involves multiple emergency response players at the scene of a small airplane crash with 30 casualties. Players participate as first responders on the scene. The players' objective is to perform primary triage based on the Simple Triage and Rapid Treatment (START) protocol. Players must locate and classify the casualties into one of four treatment
categories. The player interviews each casualty to determine cognitive health and then
examines the casualty for the status of the airway, breathing and circulation. Victims are
then tagged as red (immediate), yellow (urgent), green (delayed), or black (fatally
wounded). For the game environment, a terrain model (map) based on a real-world
airport was created from digital elevation data, satellite imagery, and local engineering
data. Using Unreal Triage as a synthetic environment, you are able to place first
responders in an immersive mass-casualty scenario to learn or practice triage, and by
making the game interact with prototype or experimental systems I can also test and
develop new products and evaluate their effectiveness in a realistic context.

The Unreal Triage game has been integrated with incident command software and
hand-held triage applications to create an immersive, synthetic environment. The
integration creates a virtual test bed, allowing emergency responders to evaluate new
technologies in a real-time, user-in-the-loop context. Currently, Unreal Triage is
integrated with a weapon of mass destruction simulation/game with Crisis Information
Management Software (CIMS).
Also, programmers added a command called “SEER” to the Gamebots API that allows one to gain access to data such as casualty events. This is called CasualtyEvent, and it communicates a sensor ID (or identifier of a victim), the casualty’s pulse, respiration rate, and blood oxygenation level. “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 [23].” The Java interface enables external programs to send data and receive data to and from the Unreal Engine. In this application, a company called Anesoft provided a stream of live or recorded physiological data to read into the Java interface. “The VizConnection interface is useful for exporting game data to other simulations or applications. They extracted scenario characters’ physiological and location data using
this interface and passed it to a remote triage casualty management system 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 [23]." This is an example of how Gamebots was implemented in a real world application.

3.2 PLAY2TRAIN

Second Life is a massively multiplayer online game (MMOG) virtual world freely accessible to anyone with an internet connection. Second Life was created by Linden Labs and was opened to the public in 2003 as a 3-D environment built and owned by its residents. People can purchase and inhabit land in the virtual environment and experience a virtual life with a population of over 9 million people from across the globe. People have rights to build digital creations in the world which they can buy, sell, and trade with others. The Marketplace within Second Life supports real transactions through its own unit-of-trade called Linden dollars. US dollars and Linden dollars can be exchanged through online Linden dollar exchanges. This enables a free flowing commerce within the virtual world. Commerce is not the only aspect of Second Life that attracts people; collaboration is also an interesting and successful outcome. Play2Train.org is an example of how the emergency management and disaster response community has embraced collaboration through this online virtual world.

"Play2Train is a virtual training space in Second Life designed to support Strategic National Stockpile (SNS), Simple Triage Rapid Transportation (START), Risk Communication and Incident Command System (ICS) Training [41]." The virtual environment was created by the Idaho Bioterrorism Awareness and Preparedness Program (IBAPP) and is located on two islands within Second Life called Asterix and
Obelix. Play2Train provides opportunities for training through interactive role playing and is the foundation for IBAPP's emergency preparedness educational machinima. There have been various experiments using Play2Train's island including Healthcare Facility Sidewalk Triage Training, Alternative Care Facility Mobile Quarantine, and Town Hall Meeting.

The Healthcare Facility Sidewalk Triage Training event begins with a pandemic and evaluates how hospitals face an overwhelming surge. The state decides that the healthcare facilities should move triage outside of the building in order to handle the surge. Also, the staff at the health care facility needs help training people to set-up and operate sidewalk triage stations. Because the pandemic involves an influenza breakout, the stations need to be designed to separate those infected from others and identify outpatient from inpatient groups via triage methods. The target participants are clinical and non-clinical workers, and the mission is to participate in training scenario experiments for mitigating hospital surge capacity with sidewalk triage.

Communications amongst participants in Second Life is usually done through third party voice applications such as Skype. These applications enable Voice Over IP (VOIP) conferencing; however, Play2Train has experienced trouble with most vendor applications. Play2Train has integrated a 3D voice capability in the First Look Voice Client to allow participants to identify the location of peer speakers. "The sound of a speaker's voice gets panned and attenuated based on the spatial inter-relationship between the sound source and the listener [41]." This will improve the communications and reduce the learning curve for the actors. The ability to plug and train with different technologies, such as VOIP in this case, is a very important aspect of evaluating
technology insertion relative to first-person perspectives. The 3D virtual environment provides a unique way to accomplish this.

### 3.3 ARENA EMERGENCY OPERATIONS CENTER PROCESS MODEL

An emergency operations center (EOC) was found to be modeled after the Incident Command System (ICS) in Arena DES by Lockheed Martin Simulation, Training & Support in Orlando, FL. The model's purpose is to represent the resources and operational activities involved in local and state EOCs.

![Fig. 3.2 Example EOC Model](image)

As seen in Fig. 3.2, the Arena EOC simulates shifts and activities for resources. Personnel resources include staff positions for command, operations, planning, logistics, finance, doctors, nurses, receptionists, etc. The model can also represent physical resources such as fax machines, telephones, computers, and servers and physical spaces such as meeting rooms, examination rooms and beds. Resources can also represent inventory of supplies such as bandages, cotton balls, needles, petri dishes and antibiotics.
Activities that the personnel complete include interoffice communication, computer work, and decision making processes. The model can help analyze end to end service times, decision making process times, etc. [21]. The details of this model can be seen in Appendix D.

3.4 SIMULATION SOLUTIONS AND SYSTEMS ENGINEERING

"Simulation modeling has been identified as one of the leading techniques for helping improve the incident management capabilities (National Research Council 2002) [29]." Simulations can provide an array of data for engineering and analysts to churn through in order to get a better perspective on complex scenarios involved in emergency management and disaster response. Emergency response agencies can take the output of simulation tools and develop their action plans. These simulations can also be used for training, allowing many different users to experience and develop action plans. Users can play what-if drills and compare alternative procedures. Simulation can also provide a multi-perspective, interactive environment; however, most tools in this field are meant for standalone use and do not attempt to address the overall emergency incident response. A specific example is the case where the standalone plume simulation can only provide the emergency response personnel information as to how the plume will spread. However, the leadership needs to plan the movement of first responder units, prepare evacuation plans, prepare plans for handling casualties, etc. Also, tools need to be integrated together to reduce the time and effort for their use.

Simulations can be used for a myriad of activities such as training, planning, concept development, and systems engineering. Systems engineering involves a holistic approach for the development and organization of independently developed, relatively
complex systems into a single system or system of systems. The importance of systems engineering and analysis in emergency management and response to terrorist events can be noted in a recent report by the Committee on Science and Technology for Countering Terrorism of the National Research Council:

“The report identified 'systems analysis, modeling and simulation' as the first of the seven crosscutting challenges to be addressed to counter the terrorism threat. Systems analysis and modeling tools are required for threat assessment; identification of infrastructure vulnerabilities and interdependencies; and planning and decision making (particularly for threat detection, identification and response coordination). Modeling and simulation also have great value for training first responders and supporting research on preparing for, and responding to, biological, chemical and other terrorist attacks. National Research Council (2002) [30]”

3.5 A CONCEPT PROTOTYPE FOR INTEGRATED GAMING AND SIMULATION FOR INCIDENT MANAGEMENT

Before the concept prototype for the serious games-based simulator is discussed in detail, the current concepts for integrating modeling and simulation tools for emergency management and disaster response should be addressed. Recently, there have been some advancements in integrating stove-piped, independently developed modeling and simulation tools to address the problems in the field. Specifically, a concept prototype was developed by Sanjay Jain and Charles McLean at the National Institute of Standards and Technology (NIST) to evaluate the value of utilizing simulation for incident management applications and the value of using simulation and gaming for training applications [29]. Fig. 3.3 shows the conceptual architecture from integrating simulation and gaming for incident management.
The architecture consists of a gaming and a simulation infrastructure integrated over socket-based client server communications, HLA communications, and customized adapters. The data synchronization and transfer processor enables an integrated capability that allows joint training of first responders and the management level personnel. The gaming module has the ability to allow civilian population and opposing forces, on scene response, support institutions, and other live elements to participate in the simulation. The simulation module has the ability to allow the social behavior, environment, physical phenomena, organizational, and infrastructure system simulators to participate.

Of particular interest, the organizational simulator category seen in Fig. 3.3, models the actions of the health care organizations including emergency operations centers and hospitals. These simulators help model the resources required and activities/tasks such as triage, transport, treatment of casualties at the emergency site and hospitals. The simulators also have the ability to include policies and procedures for emergency situations. The health care simulator was developed using a DES called ProModel/MedModel.
Another interesting piece of the simulation framework is called the strategy gaming application within the *Response Management* module of gaming applications shown in Fig. 3.3. This game can be used by management personnel in the responding agencies and personnel at the Emergency Operations Center (EOC) to plan the response resource deployments:

"The module shows a map of the incident site together with the locations of response resource providers including police stations, fire stations, and hospitals. The map also shows the important buildings around the incident site. The interface provides the capability to place icons representing response resources on the map thus making and visualizing the deployments. The application allows decision makers to develop an awareness of the situation and make decisions for resource deployment. These decisions can then be communicated to the responding teams. The board can be used with a real incident or with a simulated incident modeled using the concept demonstration prototype [29]."

### 3.6 THE CLOSEST TOOLS THAT REPRESENT THE INTEGRATION OF A DES TO A GAME ENGINE

The previous example was a custom framework and infrastructure that integrated the simulation environments with the game environments. The following section will discuss available Commercial Off The Shelf (COTS) applications that could potentially meet our needs for a serious games-based simulator. Below, I will discuss the advantages and applicability of COTS products.

#### 3.6.1 MICRO SAINT SHARP DISCRETE EVENT SIMULATION

*Micro Saint Sharp* is another DES tool that enables the user to develop 3D visualization of the scenario to be simulated. It allows you to create a virtual environment that represents a process flow with 3D moving images. The product has a built-in 3D objects library with models from multiple industries for creating realistic,
detailed scenes. It allows users to navigate through the virtual world by zooming in and out and rotating, and it uses Microsoft's latest DirectX technology for the graphics API.

In the UK, a group in the Department of Management Science at the Management School at Lancaster University developed a DES model of the Accident and Emergency (A&E) Department by using Micro Saint Sharp. They based the simulation and analysis on a task network representing the process flow of patients. They examined the effect of performance targets on UK hospitals, which is an important part of the National Health Service (NHS) performance assessment regime in the UK. Pressures on A&Es force the medical staff to take action meeting these targets with limited resources, and they used simulation modeling to help understand the factors affecting this performance. From data analysis gathered by running the simulation, they observed that the 5 category triage system is not being used in practice and patients are actually triaged by 3 categories. This is considered a positive result because triaging patients adds extra time to the total patient times in the department and affects performance.
The previous example demonstrated how Micro Saint was used for task analysis in faster than real time rather than real-time interactive simulation. The visualization through the 3D interface is completely controlled by the DES environment rather than the user, as it would be in a video game engine. However, the user does have the ability to zoom and pan. Therefore, the 3D interface is not interactive with the user during simulation execution. The entities in the 3D environment have a scripted path that tends to be unrealistic in an event driven dynamic world.

3.6.2 EXTEND V7

According to the *Extend by Imagine That!* website, Extend Sim 7 is a tightly...
integrated 3D environment boasts over 100 pre-built 3D objects and variations. You can run 3D animations concurrently with your simulation or in Buffered mode for post-processing. Animation can include sounds, shadows, footprints, and vehicle trails.

After speaking with some of the developers, I determined that the simulation leverages the *Torque Game Engine by Garage Games*. It provides access to some of the functionality of the game engine; however, you cannot customize the 3D interface as you could if you had a full independent version of the game engine. This critical piece of information is yet to be verified because I have not been able to get a beta copy of Extend v7.

In order to achieve the interactive capability, I will need to develop my own interface to a game engine and sync it with the execution of the DES in real-time.
CHAPTER IV
SIMULATION ARCHITECTURE DESIGN AND IMPLEMENTATION

4.1 SERIOUS GAMES-BASED SIMULATOR DESIGN

The Serious Games-based Simulator will be designed for detailed operational and system architecture analysis. The specific scenario will focus on the analysis of emergency operations architectures from a first responder’s perspective. Today’s operational environment consists of complex scenarios usually involving operations other than war (OOTW). Current events include complex operations such as terrorism, hostage/siege scenarios, environmental disasters (fire, flood, etc.), explosions (natural, accidental, deliberate), incidents (natural, accidental, deliberate), industrial hazards, public order, law enforcement, riot control, and movement control, etc.

In response to OOTW, our nation’s local city/county, state, regional, and national agencies are reorganizing, and it has become increasingly apparent that significant efforts are required for these organizations to develop, implement, and test the operating procedures that determine how agencies will respond, manage, and recover from catastrophic incidents. “In the past, technological solutions often failed because they did not take into account realities of emergency management decision making and inter-/intra-organizational management practices [28].” Few, if any, opportunities exist where an agency’s response plan and standard operating procedures can be tested. This is particularly true when an incident crosses local, regional, or national boundaries requiring communication and coordinated activities between agencies. There is a need to understand the critical time-lines and operational skills required to respond to an event
involving different agencies and organizations:

"The most critical period following such an event can be the time taken to respond, restore normality and take control of the situation, especially when initially there is incomplete information available. Operational officers (i.e. commanders) need to manage their doctrinal, organizational and leadership skills and then plan, exercise and test their theories in order to clearly see and understand how these situations evolve from initiation through to a successful achievement of the desired state [16]."

The same is true for the first responders to the event, who need to coordinate with the commanders and leadership managing the response along with the organizations and systems that can send the right data at the right time. This must happen as efficiently and effectively as possible in order for first responders to achieve their mission, which in this case is to save lives.

To achieve agile first response to a mass casualty event, there needs to be a way to quickly understand the information requirements of the parties involved in the mission, particularly the first responders. I will evaluate how the serious games-based simulator will be able to provide a mechanism to understand the requirements to produce an efficient, effective, rapid response to a mass casualty scenario.

4.2 TECHNICAL INTEGRATION IMPLEMENTATION

The serious games-based simulator will evaluate an emergency room architecture as it pertains to the execution of the first responder scenario in the Unreal Triage video game. "Game engines are particularly ill suited for multiplayer, real-time representations of reality, and game-based emergency response simulations integrated with information systems can provide a powerful method for evaluating human system integration [22]."

This simulation environment is just one example of how you can apply this type of systems engineering to evaluate complex scenarios. The serious games-based simulator
environment will be flexible enough to allow developers to create scenarios such as these with Unreal Tournament and stress their alternative architectures and process models with Arena.

Below is a description of the system architecture and technical integration plan that was implemented.

4.3 SIMULATION PROCESS ARCHITECTURE

First responders in the Unreal Triage training simulation will walk around the virtual environment and encounter victims that need to be treated for various injuries suffered in a disaster scenario. This can be seen in Fig. 4.1.

![Unreal Triage](image.png)

**Fig. 4.1 Unreal Triage**

The players within Unreal Triage can assume the role of the first responder. The
emergency scene provided by Unreal Triage consists of a collection of models that represent the first responder environment such as number of victims/patients and their first responder assigned triage level. As seen in Fig. 4.2, the first responders encounter victims at the emergency site and perform triage based on the health and status information received from their interaction. As first responders classify victims, they spawn events called Status Requests that query the hospital architecture models for information about hospital status called Status Reports that will allow them to know where to send the patients. Trigger events communicate and pass transactions to the various activity models represented by operational architectures. By doing so, the first responder can more intelligently determine the hospital with the best availability and lowest wait time or queue sizes. When they tag the patients with the appropriate triage level, they also assign a hospital to which the ambulance takes the patient.

![Fig. 4.2 Simulation Process Architecture](image)

The victims are classified as patients and sent to the **Health Care Treatment**
Activity Models, which contain two Hospital Models that treat the patients sent to them by the first responders in the video game environment. More specifically, a hospital model created by Fatemah Aldouli as a DES is used to represent hospitals, and this is described in detail in a later section. As seen in Fig. 4.2, the architecture implements two hospital models: one represents a hospital flooded with patient arrivals and one represents a hospital with a normal number of patient arrivals. The DES models are utilized to analyze and refine standard operating procedures and results of events and decisions by verifying interoperability between entities, identifying gaps and bottlenecks in existing plans, enhancing resource utilization and plan functionality, and rapidly exploring options to improve/refine plans. These efforts also address the integration challenges of system integration, data translation and model development.

4.4 SIMULATION SYSTEM ARCHITECTURE

As seen in Fig. 4.3, the hospital simulation model's architecture is loaded into the Arena DES. The Arena environment transforms the activities and systems functions associated with the architecture views into a dynamic executable architecture. Operational activities become process models and system functions become algorithmically represented. As transactions continue to flow through the Arena model, functions are called from the user coded DLL at each node to interface with external applications. In this case, the video game engine of Unreal Triage is the external application. More specifically, the DLL enables events in the DES to trigger messages to traverse the communications networks to external applications via socket connections. Meanwhile, human players can join the game environment via the Unreal Tournament Client and effect the way the architecture behaves by spawning events in the video game
environment that cause transactions to flow back to the DES. An example of such an event occurs when the human player (assuming the role of a first responder) places the triage tag and sensor on the victim: this spawns an event in the game engine to send the victims vitals and triage number to the assigned hospital. The hospital model in the Arena discrete event simulator receives this information and proceeds accordingly. This design enables bi-directional communication between the users playing in the video game world and the Arena modeled architecture.

Fig. 4.3 Simulation System Architecture

There was a desire to keep the integration effort modular in order to ease the verification and testing process. An external application called **UT-Arena Interface Application** was created to broker the socket communication and message formatting between Arena Health Care Treatment Activity Model and Unreal Triage, as seen in Fig. 4.2. This allowed a majority of the code development and testing for the
interface/message handling to be completed easier. The technical details will be described in detail below.

Three applications are involved in order to create real-time communication for the serious games-based simulator, and the protocol used for passing messages between the three applications is socket communication.

4.4.1 MODIFICATION OF UNREAL TRIAGE AND THE USE OF GAMEBOTS MESSAGING

Unreal Triage allows the user to play the role of emergency medical technicians or firemen conducting triage following an airplane accident. Unreal Triage is coded in UnrealScript to leverage the functionality in Unreal Tournament Game Engine. The Unreal Triage game was modified to include the Gamebots API so that it could communicate between external applications. Gamebots is implemented in Unreal Triage with UnrealScript. Gamebots classes are embedded in Unreal Triage code that create a Gamebots Server and allow Gamebots Clients to connect and pass messages. The clients can be implemented outside of the Unreal Engine in a language of your choice. An example coded in Java can be seen in Fig. 4.4. The UT-Arena Interface Application contains C code that behaves as a Gamebots Client and connects a Bot to the video game; this will be explained in further detail in a later section. In order to understand the messages, the following section explains the Gamebots API in detail.
This section contains a description of the sensory information that is passed to the Gamebots Client and the commands the client can send to control the bot and request additional information. Messages from the server are always of the form "MSGTYPE {arg1 arg1val} {arg2 arg2val}..." (Real messages do not have the quotes) In a later section, I will discuss how a parser was written based on the following assumptions:

- “All characters up to, but not including the first space are the message type. (All message types are currently 3 characters long, but best not to live by that assumption).
- Everything else in the message will be in the form of attr/val pairs enclosed by "{ }".
- The attribute name in a attr/val pair consists of every character up to, but not including, the first space.
- The value includes all characters after the space terminating the attr up to the "}" and may include spaces [35].”

A correct parsing of "MSG {Id Player-1} {String Attack the base!} {Location
12,23,34} would be:

Message type = "MSG"
Attr1Type = "Id"
Attr1Value = "Player-1"
Attr2Type = "String"
Attr2Value = "Attack the base!"
Attr3Type = "Location"
Attr3Value = "12,23,34"

Commands that the client sends to the server follow the same basic format: a message type, followed by a space, followed by attr/val pairs enclosed in "{}". Each attr in an attr/val pair should be space terminated.

4.4.1.2 SENSORY MESSAGES

The sensory messages sent to the client from the game consist of two types, synchronous and asynchronous. In the lists below, each message type is listed, along with current argument types.

Synchronous messages arrive at the client in a batch at a configurable interval. They include things like a visual update of what the bot sees and a status report of the bot itself. At the start of a batch, the server transmits a "BEG" message marked with a timestamp. All messages received until an "END" message with the same timestamp are part of the synchronous batch. They are all sent at the same instant of gametime and thus refer to a single discrete state of the game.

The synchronous messages handled in the implementation of the serious games-based simulator include:

- BEG - beginning of a synchronous batch;
- Time - timestamp from the game;
- SLF - information about your bot's state;
- Id - a unique id, assigned by the game;
- Rotation - the direction the player is facing in absolute terms;
- Location - an absolute location;
• Velocity - absolute velocity in UT units;
• Name – the player's human readable name;
• Team - the team the player is on: 255 is no team and 0-3 are red, blue, green, gold in that order;
• Health - how much health the bot has left. Starts at 100, ranges from 0 to 200.
• Weapon - weapon the player is holding.
• CurrentAmmo - how much ammo the bot has left for the current weapon
• Armor - how much armor the bot is wearing, starting at 0 and ranging up to 200;
• END - end of a synchronous batch;
• Time - timestamp from the game.

The serious games-based simulator parses a “SLF {Location 12,23,34}” message through the UT-Arena Interface Application immediately after connecting a Bot to the Gamebots Server to verify that the Bot is actually in the Unreal Triage game.

Asynchronous messages are sent from the server as events happen in the game engine. They represent things that may happen at any point in the simulation at random, less frequent intervals such as taking damage, a message broadcast by another player, or running into a wall. It is always the case that an event triggering an asynchronous message occurs in game time, between the synchronous batches before and after the event. However, there is no guarantee that an asynchronous message refers to the same discrete state of the simulation than any other message refers to.

The asynchronous messages handled in the implementation of the serious games-based simulator include:

• NFO - helpful info about the game provided right after connection is made to the server;
• Gametype – the type of game you are playing (SEER, Capture the Flag, etc);
• Level - name of the map in play;
• TimeLimit – the maximum time the game will last;
• FragLimit – the number of kills needed to win the game (BotDeathMatchPlus only);
• GoalTeamScore – the number of points a team needs to win the game (BotTeamGame, BotDomination);
• MaxTeams – the max number of teams, with a valid team range from 0 to (MaxTeams – 1);
• MaxTeamSize – the max number of players per side;
• VMS – a received message from the global chat channel;
• String - a human readable message sent by another player in the game on the global channel.

The serious games-based simulator waits for and parses the “NFO” message through the UT-Arena Interface Application before trying to send "init" back to the Gamebots Server. It also parses the “VMS {String Hello Jason}” message to receive messages from other players in the game. For this example, Players and Bots in the simulation will receive “Hello Jason” as a message on their screen or in the Gamebots Client, respectively.

4.4.1.3 GAMEBOTS COMMANDS

A Bot takes action in the world by transmitting commands to the Gamebots Server. Commands are formatted like the server messages: a command name, followed by zero or more arguments with values, each surrounded by "{}" and separated by spaces. For example, the message to initialize your Bot with a name of MYBOT on team 1 would look like this:

"init {Team 1} {Name MYBOT}"

Parsing at the Gamebots Server is case insensitive, so it does not matter in which case commands, argument names, and their values are sent. Another interesting point is that arguments can also be supplied in any order. For example, the message above could have passed the name before the team and the command would have been the same. Also, most of the commands have persistent effects. For example, once movement and rotation are started, the Bot will continue until it reaches the destination specified. As another example, when a bot is commanded to start shooting, it will keep shooting until commanded otherwise.

The serious games-based simulator sends commands from Unreal Triage to the
Arena Hospital Models through the UT-Arena Interface Application using the MESSAGE command.

- MESSAGE - sends a message to the world or just your team;
- String - the string to send;
- Global - if True, the message is sent to everyone, otherwise (or if not specified), the message is only sent to your team.

4.5 UNREAL TRIAGE MODDING/EDITING

One of the design goals of the UnrealScript programming language was to provide Java-style programming simplicity, object-orientation, and compile-time error checking [36]. As Java brings a clean programming platform to web programmers, UnrealScript provides an equally clean, simple, and robust programming language [36]. The major programming concepts which UnrealScript derives from Java are: a pointer-less environment with automatic garbage collection, a simple single-inheritance class graph, strong compile-time type checking, a safe client-side execution "sandbox," and the familiar look and feel of C/C++/Java code [36].

The Gamebots modification to the Unreal Triage training simulation is written in UnrealScript and added as a series of classes to the simulation code. Some of the major classes include BotServerConnection, BotServer, BotClientConnection, RemoteBot, RemoteBotInfo, and BotMessage. The server classes create a Gamebots Server on the specified port (I used port 3000), which allows up to 16 connections. The Bot classes handle connecting to the server, message parsing and formatting, and bot behavior.

The objective of the integration was to be able to set up the communication between Unreal Triage and the Arena Hospital Models. The purpose of this objective is to enable technology that will allow the first responder to make more informed decisions. When the first responders in the game attach the victim to a sensor, it was desired to have
a *Statistics Request* message be sent from the training simulation to hospital models automatically. The Statistics Request message is a request for hospital *Status Reports* so that the first responders know which hospital has the least load, which will help them to make a more informed decision as to where to send the victim. When the first responders give the victim a triage number and respective triage tag, the victim becomes a *patient*, and I would like to send them to the Hospital Model with information regarding the patient's status. The patient will arrive at the hospital with the best statistics. Alternatively, I might want to send the patients to a hospital based on the type of injury incurred. In some cases, hospitals cannot accept certain injuries due to lack of a particular surgeon or medical equipment. For this study, I will focus on hospital status information relative to the patient load or queue sizes.

In order to achieve this objective, functions needed to be coded within the game, using UnrealScript, that execute when receiving tagging events. *WOTgreal* was used as an Integrated Development Environment (IDE) because it is freely downloadable from the internet, and it has many features that make the coding process easier. Some of the features include syntax highlighting for UnrealScript, code insight/intellisense, hyperlink file jumping, class and package browsers, an integrated compiler, and an integrated debugger.

The function that was leveraged to send messages to Gamebots Clients was *SendMessagetoEveryone*. *SendMessagetoEveryone* is located in the *RemoteBot* class file and sends a VMS message to all Players and Bots that are listening. Again, *SendMessagetoEveryone* is called when the first responders attach a sensor and triage tag to the victim.
Most of the player functionality is located in the `SEERPawn` class. The `SEERPawn` class contains the data structures, events, and functions for objects in the game that are controlled by players. When a first responder attaches a sensor to the victim, the UnrealScript calls the `SendHospitalStatusRequest` function. The `SendHospitalStatusRequest` function formats information about the victims (soon to be patients) and sends a string message to `SendMessageToEveryone`, which in turn sends the message to Bots via the `Gamebots API`. The format of the message sent from `SendHospitalStatusRequest` to `SendMessageToEveryone` is:

"REQUEST Hospital = Status"

`SendMessageToEveryone` sends this message to Gamebots Clients as described in the `Gamebots API` in the format seen below:

"MESSAGE {String REQUEST Hospital = Status}"

When a first responder attaches a sensor to the victim, the UnrealScript calls the `SendPawnInfoToGB` function in `SEERPawn` class. The `SendPawnInfoToGB` function formats information about the victims and sends it to `SendMessageToEveryone`, which in turn sends the message to Bots via the `Gamebots API`. The format of the message being sent from `SendPawnInfoToGB` to `SendMessageToEveryone` is:

"INFORMATION TAG = 'SerialNumber', Triage = 'Triage', Breathing = 'Breathing', Pulse = 'Pulse', Injury = 'InjuryStat"

`SendMessageToEveryone` sends this message to Gamebots Clients as described in the `Gamebots API`.

### 4.6 UT-ARENA INTERFACE APPLICATION

The **UT-Arena Interface Application** brokers the socket communication and
message handling between Arena Hospital Models and the Unreal Triage game. It is an executable application developed in Visual Studio Express in the C programming language. It uses three libraries called pthreads, GAPI, and ARAPI to connect to sockets and receive, format, and send messages using the Gamebots API. Below is a detailed discussion about the overall functionality of the application and the individual libraries.

The UT-Arena Interface Application is the main executable that brokers the program flow and communication between Unreal Triage and Hospital Models. The application has two main tasks: open and close sockets and message handling. The pseudocode below shows how the application behaves:

```
Initialize the sockets (winsock2)

If (Unreal Triage is enabled)
  If (there is a game on port 3000)
    connect to Unreal Triage Game
    add Bot to game
    start message handlers for sending and receiving messages from the game

If (Arena is enabled)
  If (there is a Simulation on port 4334)
    connect to Arena
    start message handlers for sending and receiving messages from Arena

Loop until Game and Simulation End
```

The UT-Arena Interface Application uses ARAPI and GAPI to seamlessly broker communications between Unreal Triage and Arena. Below is a detailed discussion of the implementation of the libraries that enable this.
4.7 PTHREADS-W32

Pthreads is an application programming interface (API) for writing multi-threaded applications defined by the POSIX 1003.1-2001 standard. Many modern operating systems include a threading library of some kind: Solaris (UI) threads, Win32 threads, DCE threads, DECthreads, or any of the draft revisions of the pthreads standard. Most of these systems are slowly adopting the pthreads standard API, with application developers following suit to reduce porting woes. Win32 does not, and is unlikely to ever, support pthreads natively. The pthreads-w32 project provides a freely available and high-quality solution to this problem [37].

Various individuals have been working on independent implementations of this well-documented and standardized threading API, but most implementations never see the light of day. The tendency is for people to only implement what they personally need, and that usually does not help others. The prthreads-w32 project attempts to consolidate these implementations into one implementation of pthreads for Win32 [37].

The Win32 pthreads is normally implemented as a dynamic link library (DLL). This has some notable advantages from the Win32 point of view, but it also more closely models existing pthread libraries on UNIX, which are usually shared objects (e.g. libpthread.so) [37]. Also, the library can be built for static linking if necessary.

The pthreads-w32 project was built as a static library for the GAPI, ARAPI, and UT-Arena interface Application to keep the project more modular. The use of the pthreads-w32 static library allows the UT-Arena Interface Application to be a multi-threaded application in a Windows environment.
4.8 GAPI

**GAPI** is a static library built in the C programming language that links to the UT-Arena Interface Application. The library provides functionality that acts as a Gamebots Client; it connects to a Gamebots Server, adds Bots, and handles messaging between Unreal Tournament and itself.

**GAPI** provides the functionality to connect to the Gamebots Server using winsock2. The `gbclient_new` function will create a client for the Gamebots Server given a `host ip address` and a `port number`. The specific configuration for the UT-Arena Interface Application will be discussed in the next section. The `gbclient_connect` function will try to connect to the configured client, look for a game on the server and port, retrieve the Gamebots Server `game info` and return it. The `gapi_connect` function will connect to the Gamebots Server and initialize the server with a `Bot (player)` and team information. Once the client is connected and the `Bot` is in the game, I can send and receive messages from the game server. The `gbclient_receive` function receives a message from the Gamebots Server via the client.

**GAPI** provides a way to listen to the sockets for all `Gamebots` messages. In order to do this and share the socket connection, both functions use the `__gbclient_lock_mutex` and `__gbclient_unlock_mutex` to obtain and release sole access to the client. Once a message is received, there are `gbmessage` functions that can parse the Gamebots messages according to the API specifications. The parsing first looks for the message type then parses by attribute and value pairs. To take an example previously discussed, the function will parse “VMS {String REQUEST Hospital = Status}” from the Gamebots Server as:
Message Type = “VMS”
Attribute = “String”
Value = “REQUEST Hospital = Status”

As soon as a message is parsed, it can be handled through signals. A signal is a software interrupt delivered to a process. The operating system uses signals to report exceptional situations to an executing program. Some signals report errors such as references to invalid memory addresses; others report asynchronous events, such as, in this case, a Gamebots message. I anticipate events that cause signals; handler functions are defined that tell the program to run a certain function when that particular type of signal arrives.

GAPI also provides a way to send message commands to the Gamebots Server. This is done by formatting messages according to the Gamebots API. For the purposes of the serious games-based simulator, I will be using the MESSAGE command for communications.

4.9 ARAPI

ARAPI is a static library built in the C programming language that links to the UT-Arena Interface Application. The library provides similar functionality to the GAPI; however, ARAPI acts as an Arena Client. It connects to an Arena RT Server and handles messaging between Arena and itself.

ARAPI provides the functionality to connect to the Arena RT Server using winsock2. The arclient_new function will create a client for the Arena server given a host ip address and a port number. The specific configuration for the UT-Arena Interface Application will be discussed in the next section. The arclient_connect function will try to connect the configured client, look for the server and port, retrieve the information and
return it. The *arpi_connect* function will connect to the Arena RT Server. Once the client is connected, I can send and receive messages from the Arena RT Server. The *arclient_receive* function receives a message from the Arena server via the client.

ARAPI provides a way to listen to the sockets for all Arena messages. In order to do this and share the socket connection, both functions use the *_gbclient_lock_mutex* and *_gbclient_unlock_mutex* to obtain and release sole access to the client. Once a message is received, *armessage* functions parse the Arena messages according to the specifications defined in the *Arena Execution Mode* and the *Implementing Inter-Process Communications (IPC)* section of this paper. The parsing first looks for the message type then parses by attribute and value pairs. For example, the function will parse “SendMsg Hospital=2, PatientTotalTime=0.33, TriageAreaQueue=11, TNOW=4.7;” from the Arena server as:

```
Message Type = “SENDMSG”
Attribute = “Hospital”
Value = “2”
Attribute = “PatientTotalTime”
Value = “0.33”
Attribute = “TriageAreaQueue”
Value = “11”
Attribute = “TNOW”
Value = “4.7”
```

As soon as a message is parsed, it can be handled through signals. The *signals* check the message type and execute the appropriate functionality. As seen in the text at the bottom of Fig. 4.4, the UT-Arena Interface Application would send this message, which contains hospital statistics, to Unreal Triage.
ARAPI also provides a way to send message commands to Arena. This is done by formatting messages according to the specifications defined in the *Arena Execution Mode* and the *Implementing Inter-Process Communications (IPC)* sections of this paper.

### 4.10 FATEMAH ALDOULI'S HOSPITAL MODEL

The hospital model presented in “Applying A Simulation-Based Real-time Decision Making Method to Mass Casualty Scenarios in Hospital Emergency Departments” by Fatemah Aldouli from Old Dominion University is used to represent the two hospitals in the serious games-based simulator scenario. Aldouli’s hospital model was developed to improve the decisions that healthcare staff take during catastrophic
mass casualty events. Aldouli developed the hospital model as a DES model to support Simulation-Based Real-time Decision Making (SRDM) for the hospital staff. The serious games-based simulator uses her hospital model to simulate and evaluate the information exchanges between the hospital and first responders in the Unreal Triage scenario. The serious games-based simulator will use the hospital models from a Systems Engineering perspective rather than as an SRDM.

First, the hospital model simulation development will be discussed in detail according to Aldouli’s thesis. The ER in Children's Hospital of the King’s Daughters (CHKD) was chosen as the candidate hospital architecture because its challenges in handling large numbers of patients. The CHKD ER data archives revealed that during conditions such as disasters, certain disease seasons, or heavy load weekends, patient waiting time lasted hours before patients were seen by a physician. These adverse conditions resulted in frustration, panic, and uncertainty that inevitably, negatively affect the decisions ER staff might make [33]. Some of the objectives of Aldouli’s study were to utilize DES modeling techniques to resolve problems that occur during mass casualty events, to optimize ER performance when handling mass casualties, and to employ SRDM in a software tool to support the decisions ER staff make [33].

4.10.1 HOSPITAL DESCRIPTION

The hospital model represents the Emergency Department at CHKD, which is located at 601 Children's Lane, Norfolk, VA 23507. Annually, more than 5,700 children are treated at CHKD as inpatients and nearly 85,000 as outpatients. The ER treats an additional 40,000 children each year [33].

CHKD’s ER has a total of 28 beds located in four care areas: the Main Treatment
Area (MTA), the Resuscitation Area (RA), the Fast Track Area (FTA), and the Monitored Bed Units (MBU), also known as the overflow area [33]. The resources are distributed as follows:

- The ER has 5 Doctors;
- There are 2 Triage Nurses and 2 Triage rooms;
- The registration desk has 2 people;
- The MTA contains eleven beds;
- The RA is comprised of four rooms with each room containing only one bed;
- The FTA has seven beds although one of the beds is seldom used;
- The MBU has seven beds; however, one of the beds is not used due to storage space.

**Fig. 4.1 Patient Flow through the Hospital Model ER [33]**

### 4.10.2 ER PATIENT PROCESS FLOW

The process flow of a patient through the ER involves a complex series of decisions, activities, and interactions with ER personnel [33]. The patient enters the ER through the triage area where a nurse at the triage desk will take care of the patient's immediate medical needs. Examples of these activities include placing a bandage on an
open wound, applying ice, or providing fever or pain medication [33]. The triage nurse examines the patient and records the patient’s weight, temperature, heart and breathing rate, and blood pressure. The triage nurse assigns the patient an acuity (triage) level from 1 to 3 depending on the patient’s condition (level one being the most severe and level three being the least severe) [33].

The patient will proceed to the main waiting area to wait for their name to be called by a staff member at the registration desk. The registrar will obtain the patient’s personal information and insurance information, make/update a medical record, and administer a name band to be worn on the patient's wrist or ankle [33]. The patient waits to be called for treatment by the charge nurse based on acuity level. When called by the nurse, the patient is escorted to one of the five areas of the ER where they wait for treatment by a physician [33]. Fig. 4.2 shows the flow for the ER patients; numbered circles show the patient’s movement from “0” to “5” respectively. Circles labeled “5” are the ER care areas where a patient is expected to go based on his/her acuity level [33].
4.10.3 MODEL INPUT DATA

Aldouli gathered input data from CHKD and used the *Input Analyzer* tool in Arena to fit probability distributions for: patients’ inter-arrival times, waiting time, and service times[33]. Table 4.1 shows the probability distributions.
Patients' arrival rate varies depending on factors such as whether they are walk-ins and the day of the week. Tuesday, Wednesday, and Thursday were found to have low arrival rates and patients' arrival rates increased starting on Friday through Monday. The inter-arrival rates of the patients are non-stationary Poisson distributions as seen in Fig. 4.3 and Fig. 4.4.
Fig. 4.3 Non-stationary Poisson Distribution for the average of walk-in patient arrival rate between Tuesday and Thursday [33]
Fig. 4.4 Non-stationary Poisson Distribution for the average of walk-in patient arrival rate between Friday and Monday [33]

4.10.4 ASSUMPTIONS FOR THE CHKD ER MODEL

Aldouli outlines the following assumptions used in CHKD ER model:

- “The system starts idle at the beginning of each day;
- Patients arrive only using personal transportation (considered to be "walk-in patients") or using the hospital ambulance;
- Patients always arrive as individuals, never more than one patient at the same time;
- Although the ER has more than one entrance, only one was modeled;
- Entities’ travel time between different areas in the ER was neglected;
- A room could be occupied by one patient only;
- The role of the ER nurses was omitted;
- There is always five doctors in the ER;
- The input data included records of the time a patient spent between each main stop in the ER. For example: TriageToRegistration, RegistrationToRoom, etc.
These records don't separate the delay that patients face before accessing the ER resources (waiting for an idle triage nurse or a registration clerk). Therefore, these records provided were used as the resources’ service times. As an example, \textit{TriageToRegistration} was used as the triage nurse service time. \textit{RegistrationToRoom} was used as the registration service time as well;

- The input data provided for the study included negative values and empty cells. The negative values indicate that certain times were recorded in an illogical order, which is usually a result of making mistakes in recording the data (for example: triage time is 10:00 while registration time is 9:55. This makes the \textit{TriageToRegistration} time -5). On the other hand, the empty cells in the input data sheet are a result of not recoding the beginning or the end or both of the times needed to calculate time intervals that a patients spends between two areas in the ER. As a result of omitting such information, more than 25\% of the input data was not used in the model. This affected patients’ arrival rate and, hence, the model throughput [33].”

Fig. 4.5 - Patient Flow in the ER [33]

4.10.5 CONSTRUCTING THE HOSPITAL MODEL

At this phase, the goal was to obtain a representation of the ER in CHKD in order
to simulate the flow of the patients [33]. The flow of patients is shown in Fig. 4.5.

The model was built in Arena 5.0; however, it was converted into Arena 11.0 for the purposes of this study. Fig. 4.6 contains three modules (two Create modules and one Dispose module) and six submodels. Submodels are used to partition the model in order to help organize it.

ER patients are represented as entities, treated as a single entity type, and generated using two Create modules labeled Walk In Patient Enters ER and Ambulance Patient Enters ER [33]. Entities then enter the Before Doctor Activities submodel, and, as seen in Fig. 4.7, entities are directed to the Triage Area module. During this process, patients see triage nurses and use triage room resources where a triage nurse provides patients basic help and assigns acuity levels using a Decide module [33].
Fig. 4.7 - Before Doctor Activities Submodel [33]

Input data suggests that 4.3% of the patients were assigned level 1 acuity, 16.92% assigned level 2 acuity, and the rest assigned level 3 acuity [33]. The purpose of this module is to direct the entities to different modules to assign their acuity levels [33]. Attribute values are assigned to the entities based on the level of acuity. The patient exits the triage area and registers to be seen in the ER. This is accomplished using the Registration and Wait for Room process module seen in Fig. 4.6. The delay in this process represents the time spent waiting for a registration clerk to become available, the time spent acquiring the patient’s information, and the time spent waiting for an available room [33].

As seen in Fig. 4.8, entities leave the Before Doctor Activities submodel and enter the Determine A Room submodel. This submodel uses three Decide modules to read the acuity level of an incoming entity and route to the appropriate area in the ER.
Patients with acuity level 1 are sent to the RA if it is not full and to the MTA if otherwise [33]. The FTA accepts patients with acuity level 3 if it is not full; otherwise, patients are sent to the MTA [33]. Patients with acuity level 2 are sent to the MTA if it is not full and the MBU, also known as the overflow area, otherwise [33].

*Determine A Room* submodel routes to the four different areas in the ER named Resuscitation Area, Main Treatment Area, Fast Track Area, or Overflow Area submodel. As you can see in Fig. 4.9, these submodels share the same logic. Fig. 4.9 shows the details of the MTA submodel [33]. The service time and the number of beds available are different in each area.
The time patients spend in any of the ER areas depends on their condition and was included in the input data provided by the ER statistician. However, the amount of time doctors and nurses spend assisting patients was not available; 10 minutes was assumed to be the doctor service time [33].

A Dispose module, shown in Fig. 4.6, is used to show that an entity left the model and means that the patient might have left the ER because he was admitted to the hospital, discharged, or died [33].

4.10.6 VERIFICATION AND VALIDATION OF HOSPITAL MODEL

In order to assure that the model was built correctly (or verified), Aldouli ran the model under several testing scenarios, evaluated the simulation output, and used the animation mode in Arena to make sure the patients were flowing through the correct paths. She also gave the model to another analyst who was experienced with Arena as a verification method.

To make sure that the model accurately represents the real world hospital, validation included degenerate tests and internal validity methods:
“Degenerate Tests are defined as follows: Input variables and other internal parameters could be varied to test the model reaction to worst-case situations. This technique helps in studying the simulation model degeneracy. Internal Validity is defined as follows: It is desirable to have consistent outcomes when the same circumstances are used to run a simulation model several times [33].”

4.10.7 EXCURSION INTERARRIVAL INPUT

For the CHKD experiment excursion, Aldouli wanted to simulate the patient flow during a disaster or mass casualty event. In order to represent this increase in arrivals due to the disaster, the interarrival rates needed to be provided to Arena. As seen in Fig. 4.10, a non-stationary Poisson distribution is used for the average patient arrival rate of walk-ins [33].

Fig. 4.10 - Non-Stationary Poisson Distribution for the Average of Walk-In Patient Arrival Rate During Heavy Load Times [33]
4.11 MODIFICATIONS MADE TO THE HOSPITAL MODEL

Fatemah Aldouli’s Arena Hospital model was used in the simulation study for the serious games-based simulator to simulate the operational activities and patient flow through the CHKD emergency room. However, the CHKD model required slight modification in order to achieve the study objectives for this thesis. For the study, I duplicated the hospital models in order to represent two hospitals. One hospital operates with an inter-arrival rate schedule of patients representative of the disaster scenario schedule. The other hospital operates with an inter-arrival rate of patients representative of the Friday to Monday schedule.

I also needed Arena to execute in RT (Real Time) mode so that I could execute inter-process communications (IPC) with external applications. In this case, the external application was the message broker called the UT-Arena Interface Application so that hospital models can have bi-directional communication with Unreal Triage. **Arena RT mode** allows Arena to communicate over sockets and utilize the IPC queue to handle sending and receiving messages based on events in the Hospital Model. The next section presents a detailed discussion of implementation.

4.11.1 ARENA EXECUTION MODE AND IMPLEMENTING INTER-PROCESS COMMUNICATIONS

Running in the *Execution Mode* option in Arena, allows you to enable inter-process communications between the model and an external application. The IPC can be coded in Visual Basic or C++. For the serious games-based simulator, I used the C++ *user-code library* as described below. The routines are located in the *UserCode.dll*. Also,
Arena has provisions for linking in user-coded C routines. The C functions listed in the user-coded routines section provide the link from SIMAN models to user-written code. These routines provide great flexibility for writing initialization, ending, and logic routines [40].

To create my own user-code, I must edit the `userc.cpp` file (for C++) and replace the dummy code with my own code for any or all routines. Next, I build the project to create a *Win32 Dynamic Linked Library (DLL)*. By default, the DLL is named `msuserc.dll`. To use the DLL, I must open a model in Arena and click the *Run/Setup* menu option. I select the *Run Control* page and type the name of the DLL at the *Load User-coded .DLL* prompt. This information will be saved with the model file.

Alternatively, I can choose to specify a DLL to be used by all new models by selecting the *Tools/Options* menu item, selecting the *Run Control* page, and typing the name of the DLL at the *Load User-coded .DLL* prompt.

“*The routines described below are within* `userc.cpp` *and are only called if Arena is running in execution mode:*

- **InitProcess** - Called at the beginning of the first replication. Place code that initializes inter-process communications here.
- **ReadIPCQueue** - Called when an entity tries to send a message to the external process. Place code that sends the message to the IPC queue here.
- **WriteIPCQueue** - Place code that retrieves messages from the IPC queue and passes them to Arena here.
- **ShutdownIPC** - Called at the end of the last replication. Place code that terminates inter-process communications here.

If Arena is running in execution mode and the user-coded DLL and VBA approaches have both been implemented, then the execution mode routines in the user-coded DLL will be called and the VBA code ignored [40].”

For the serious games-based simulator, Arena sets up a socket on the local ip
address and port 4334.

4.11.2 SENDING MESSAGES TO THE EXTERNAL PROCESS

Once the user-coded DLL was in place, I needed to define the message strings that simulation entities may send to an external process in the TASKS element. The syntax and the basic use is shown below:

```
TASKS: Number, TaskID, Execute, Format: Parameters, ...: repeats;
TASKS: LoadPart, 1, "Load part %f", IDENT;
```

For example and explanation purposes, I use generic manufacturing terms such as `LoadPart` and `ProcessPart`. "Whenever the task LoadPart is referenced in the model, a simple message will be sent to the external process. A typical message would be "Load part 7.0" for entity 7. The entity will wait at this block until a task complete message or timeout occurs. A typical task complete message would be "0 7 0". The three numbers indicate the message type (must be 0), the entity ID (must match entity that sent message), and the return code (non-zero indicates error) [40].”

If Arena is running in execution mode, then a message can be sent when an entity enters one of the following constructs:

- DELAY module or block
- ROUTE module or block
- TRANSPORT module or block
- MOVE module or block
- PROCESS module
- ENTER and LEAVE modules

For the serious games-based simulator, the DELAY and ROUTE blocks were used to send information. The details of how to use these blocks can be found in Appendices A and B."
4.11.3 RECEIVING MESSAGES FROM THE EXTERNAL PROCESS

The two types of messages that an Arena model can receive from an external process are:

- A response to an entity task (i.e., a response to a message that was sent by an entity when it executed a construct’s TASKID expression);
- An unsolicited message to Arena.

For the serious games-based simulator, I will define and accept unsolicited messages using the ARRIVALS block. The details of how to use this block can be found in Appendix C.

4.11.4 SPECIFIC IMPLEMENTATION FOR SERIOUS GAMES-BASED SIMULATOR

As discussed in the previous section, the serious games-based simulator duplicated the CHKD model to represent two hospitals. Fig. 4.12 shows that along with duplicating the hospital models, message handlers were created to allow for IPC, and entity handlers were created to route entities.
The Message Receiver submodel was created to enable the hospital models to receive messages from Unreal Triage. As seen in Fig. 4.13, Message Receiver implements the Arrivals element so that it can receive unsolicited messages. Unreal Triage sends unsolicited messages of Message Type “1” or “2”. Message Type “1” is a Patient entity and Message Type “2” is a Statistics Request entity. A message being sent from UT will arrive at the Arrivals element and be parsed. The Arrivals element will parse the incoming message by identifier (based on message type; either “1” or “2”) and attribute/value pairs.

For Patient arrivals, the message from UT would be:

1 2 105 3 {Message Type "1"; Assign the values "2","105", and "3" to the specified attributes att_Hospital, att_TagID, and att_Triage}

The Patient would enter the Message Receiver station and travel to Hospital 2 with a Tag ID =105 and Triage/Acuity Level = 3.
For Statistics Request arrivals, the message from UT would be simply 2 because it does not have any attributes.

![Message Receiver submodel](image)

**Fig. 4.13 Message Receiver submodel**

If the Message Receiver handles a Patient entity, it will be routed to the *UT Patient Arrival* submodel seen in Fig. 4.14. This functionality assumes that Patient entities that have already been assigned to a hospital and have already been triage tagged by the first responder in Unreal Triage. Therefore, the routing will check the attributes of the entities and route as appropriate. This submodel will route the Patient by the hospital attribute. For example, if the `hospital_att` is equal to 1, then the Patient will be routed to Hospital 1. Once the submodel has the Patient routed to the appropriate hospital, it will then route to the appropriate submodel by the triage/acuity level attribute. Because these Patients have been triaged by the first responder in the video game, the routing of the entities will bypass the triage activities at the hospital and go straight to *Registration and Wait for Room* submodel.
If the Message Receiver handles a Statistics Request, the message will be routed to the *Get Statistics* submodel seen in Fig. 4.15. For each Hospital, the submodel will assign statistics to variables. Then, those variable values will be assigned to the message attributes. The routing will send the message straight to the *Message Sender* submodel.

---

**Fig. 4.15 Get Statistics Submodel**

The Message Sender submodel was created to enable the hospital models to send a message to Unreal Triage. Message Sender implements the *Tasks* element which will
format a message to send to UT from the name of the task and the attributes of the
message. In this case the name of the task is SendMsg and the attributes are
att_entityType, IDENT, att_Hospital, att_PatientTotalTime, etc. The format of the
message is below:

"SendMsg Type=%1.0f, TGID=%1.0f, Hospital=%1.0f, PatientTotalTime=%5.2f,
TriageAreaQueue=%1.0f, TNOW=%5.2f;"

Fig. 4.16 Message Sender Submodel

The previous section discussed the modeling that needed to be completed to allow
for IPC and representation of two or more hospitals. The next section will discuss
running the simulation using the Real Time Factor.

4.11.5 ADVANCE SIMULATION TIME USING REAL TIME FACTOR

This model option allows you to set the speed of Arena’s simulation clock as a
factor of the real-time clock or wall clock of the resident operating system. It is specified
in Run > Setup > Run Speed. You can specify the default setting of this option for new
models in Tools > Options > Run Speed.

As an example, if this option is checked and the Real Time Factor is 1, then Arena
will advance the simulation clock in real-time (i.e., 1 simulated minute will correspond to
1 real minute). If this option is checked and the Real Time Factor is 2, then Arena will
advance the simulation clock at a speed twice as fast as real-time (i.e., 1 simulated minute will correspond to 30 real seconds).

4.11.6 ISSUES RUNNING IN REAL TIME

Again, this option is a feature of Arena RT. Arena RT's evaluation mode allows a model to be run with a real-time clock and/or in execution mode for a maximum of 10 minutes. To run a model with these features for an unlimited time, you must purchase Arena RT with full commercial authorization. Initially, it was difficult to obtain a license for Arena RT within the time period of my experimentation schedule, so I needed to run faster than real time and complete each experiment run in under 10 minutes.

Another problem with running in real time is that it takes almost two hours for the scenario to play out completely, making the experimentation and data collection process lengthy. The scenario starts with everything initialized to zero, e.g. all queues at the hospitals and work in progress equal zero. During the disaster, roughly 5 people arrive at the hospital each hour or 120 arrive in a 24 hours span. It takes roughly 3 hours for a patient to run completely through the hospital process in a disaster scenario. Therefore, I wouldn't expect representative performance metrics about the hospital's status to be available for almost two hours after the start of the model. To obtain statistically significant data for each experiment, I need to run the simulation about 30 times. If I execute 30 replications for each experiment and it takes 2 hours for each run, I would need to spend 60 hours running the model for each experiment. To avoid this lengthy simulation time for data collection, I ran the Arena Hospital Models faster than real time. I had two options to execute my experiments.

One option was to *pre-load* the model and execute it in real time with this data
during the experiments. In order to pre-load the model, it is possible to run the model without animation for 24 (simulation clock) hours, 30 replications, before each experiment; this takes roughly 10 seconds of *Wall Clock Time*. The results from the faster than real time executions provide values for the average queue lengths and work in process to be used to front-load/pre-load the model during experiment runs. The pre-loading process happens at $t = 0$, and the model does not continue to the next time step until all pre-loaded values are set. The main issue with this option is that the pre-loaded entities will start in various places of the model, sometimes closer to the exit of the hospital. When Arena calculates the total time of the pre-loaded patients, it will not take into consideration the activity times associated with the processes that they would have gone through *before* the pre-load insert point. Due to this issue, I did not use this option.

The other option is to modify the Real Time Factor of the Arena model as described in a previous section. First, I must determine the amount of Wall Clock Time I would like to use to run the experiment. Due to licensing issues, I could only execute the experiment simulation run for 10 minutes. Next, I must decide how long (simulation time) I would like the Arena model to run. I call this *Simulation Replication Length*. These two times must be in the same unit of measurement. I can determine the *Real Time Factor* by dividing the Simulation Replication Length by the Wall Clock Time as seen below:

$$\text{Real Time Factor} = \frac{\text{Simulation Replication Length}}{\text{Wall Clock Time}};$$

For simulation purposes, I used a Simulation Replication Length of 24 hours and a Wall Clock Time of 10 minutes, yielding a Real Time Factor of ~ 144. Note, I round up and err on the higher side of the calculation so that Arena will complete before the 10
minute wall clock period. Remember, Arena stops execution at 10 minutes. Therefore, I would like to finish simulation time slightly before 10 minutes so that Arena can calculate statistics before it stops execution.

To reiterate, this technique forced the simulation to execute 24 hours within 10 minutes. After running some simulations to test this technique, it was determined that it cannot produce statistically significant results. The data collected has a very large variance because the difference in time between sending the first victim to the hospital and the next victim to the hospital is an hour or more from the perspective of the hospital model. For example, the victims are sent to the hospital at time \( t \) in the video game environment and arrive at the hospital at time \( t \times 145 \). If the first responder triages victim1 at a video game simulation time of 5 minutes, that victim will arrive at the hospital (assuming transportation time is 0) at 725 minutes. If the next victim gets triaged at 7 minutes into the video game, then she/he will arrive at the hospital at 1015 minutes. The difference in the video game is only 2 minutes; however, it is 290 minutes in the hospital model. Clearly, within the 5 hours at the hospital the conditions could have changed drastically, therefore significantly increasing the variance of the data.

The solution to this problem is to set the Simulation Clock in the Arena hospital model faster than real time until the time at which you would like to start the triage process, slow Arena’s simulation clock to execute in real time, complete the triage process in the video game, then speed Arena’s simulation clock up to faster than real time until the end of the simulation. In order to achieve this, Arena provides support for user customization to synchronize simulation time advances with an external process or clock if the Advance Simulation Time Using Real Time Factor option is enabled. For the
purposes of experimentation with the serious games-based simulator, it was deemed necessary to purchase a full license of Arena RT because of the desire to execute the simulation for more than 10 minutes. It is possible to programmatically modify Arena's simulation clock using the `UserInitializeMaxTimeAdvance` function and `UserGetMaxTimeAdvance` function in the Arena C++ usercode dll.

`UserInitializeMaxTimeAdvance` is called at the beginning of the first replication. Initialization code is placed in this function to perform the clock synchronization at the simulation's start. The `userInitialSysTime` variable is used to store the start time. The `UserGetMaxTimeAdvance` function is called regularly by Arena during the simulation run. This contains the code that calculates and returns the maximum time (in hundredths of a second) that the simulation time may advance. The algorithm's pseudocode to implement variability for Arena's simulation clock advance for the serious games-based simulator is below:

```verbatim
if current_sim_time is less than triage_start_time
    return (rt_factor * (elapsed_time - start_time)) //faster than real time
else if current_sim_time is greater than triage_end_time
    return (rt_factor * (elapsed_time - start_time + start_triage_time +
                      triage_end_time)) //faster than real time
else
    return (elapsed_time - start_time + start_triage_time) //real time
```

This algorithm appropriately and accurately implements a clock that synchronizes the Arena Hospital Models to suit the needs of the serious games-based simulator.
4.12 VERIFICATION AND VALIDATION OF THE SERIOUS GAMES-BASED SIMULATOR

Verification and validation was completed in a series of steps that allowed testing of the individual components of the serious games-based simulator. First, the Arena model was validated and verified after adding the Arena RT blocks and necessary functionality to route the new entities correctly. Next, the Unreal Triage game was validated and verified after adding the UnrealScript to allow the game to send and receive correctly formatted messages after certain events. Finally, the UT-Arena Interface Application was tested after the complete integration of the Arena hospital models and Unreal Triage. Also, a series of external applications were used or developed to make verification and validation easier.

The RTConsole.exe tool was a quick and easy way to send input to the Arena hospital models. The RTConsole.exe, which can be seen in Fig. 4.17, is a Visual Basic 6.0 application provided by Arena as an example client application. The source code files of RTConsole.exe are installed in Arena’s Examples folder. RTConsole.exe is also embedded in the example model RT Execution Mode.doe for demonstration purposes. There are step-by-step instructions embedded in RT Execution Mode.doe for running this example. The RTConsole.exe allows the user to send and receive IPC data, so the new Arena hospital models were tested by sending Patient Messages and Statistics Messages to the model during execution. This also provided a way to test the message formats that needed to be sent and received by the UT-Arena Interface Application.
JavaBots, seen in Fig. 4.18, was used to verify and validate the Unreal Triage modifications and UT-Arena Interface Application. Andrew Marshall at USC-ISI created the JavaBot API to provide a higher-level interface to the Gamebots for Unreal protocol and make it much easier for research groups to develop agents/bots for this rich domain. Developers can create Unreal Tournament software players without having to worry about the specific Gamebots protocol, network socket programming, message passing and other pragmatic issues [38]. JavaBots allows the user to send and receive Gamebots messages to Unreal Triage. The new functionality in Unreal Triage was tested by sending
Statistics Messages to the model during execution. This also provided a way to test the message formats that needed to be sent and received by the UT-Arena Interface Application.

To rigorously test the UT-Arena Interface Application, ARAPI and GAPI, various C functions were developed to print outputs and write to output log files. This created a single location to check logs for output messages.

Verifying that the integrated components are working as expected was accomplished using the same methods used to verify the original ER model. Verification started by assuming that the original model was verified before the integration of Unreal
Triage and Arena.

In order to verify that the Arena model was free of bugs and errors, I placed various blocks in the model to check entity types and attributes. If an entity type or attribute was not recognized or did not belong in the part of the process, the model would use a decision block and route the entity to a dispose block. The model was run with various inputs and the dispose blocks statistics were checked in the *Crystal report* after completion of simulation runs. Arena's reporting of statistics proved very useful in verifying the model because it shows queue sizes, utilization, and entity counts in an organized, structured way.

After verifying the UT-Arena Interface Application and Arena models, the same validation methods used to validate the original ER model were used to validate the model with Unreal Triage added to it. The new functionality of Arena RT added various blocks that may have affected the way the model behaved. A complete validation process implies that the simulation is behaving just like the real-world system and this can be almost impossible because the system may not exist yet. However, the integration process started by assuming that Unreal Triage and Arena were created with valid models. Therefore, validation with fixed values was used to validate the application. Fixed values were used for the model input, internal variables, and parameters instead of using random values. Simulation results were calculated using these fixed values. Then, these calculated values were compared to the simulation results. Specifically, simulation results were compared to the original, validated models and simulation results.
CHAPTER V

EXPERIMENTATION AND ANALYSIS

5.1 THE PROPOSED SCENARIO

As seen in Fig. 5.1, there are two mass casualty events in one day in Hampton Roads that result in a major fire and casualties. One event is an explosion at a commercial building in Portsmouth, Virginia. The information reaches the 911-call center within minutes of the occurrence and the first units of police and fire vehicles arrive at the scene. The explosion happens at a peak business hour and appears to have resulted in a large number of casualties on a high floor of the building.

Fig. 5.1 Scenario Map
The other event is a plane crash at Norfolk International Airport (ORF) where there are very serious injuries and casualties in the double digits happening eight hours after the Portsmouth incident.

![Fig. 5.2 Burning Plane and Victims](image)

Fig. 5.2 Burning Plane and Victims

The burning plane and victims surrounding the crash site can be seen in Fig. 5.2. First responders arrive at the plane crash, and they know of two hospitals in the region that can accept patients with these types of injuries. Fig. 5.3 shows the first responder walking through the crash site and examining patients with various injuries.
Fig. 5.3 Plane Crash Victims

One hospital is the Children's Hospital of the King's Daughters (CHKD) and it is filled to capacity by patients with various injuries from the Portsmouth event. The other hospital (DePaul Medical Center) is west of the airport and is experiencing normal, everyday patient flow.

Scenario 1 - Base Case Experiment: The first responders to the plane crash do not have information about the hospital status and statistics. However, they do know from experience that the hospitals are equidistant from the airport and can treat victims with the types of injuries the first responders expect. Therefore, a first responder randomly chooses a hospital with a 50% probability, to send the patients from the crash site.
Scenario 2 - Experiment with Arena Sending Hospital Information: The first responder will be equipped with technology, such as a PDA or cell phone, having text message communications with each hospital in order to receive status information and statistics. The PDA also has a smart agent application that will recommend the hospital with the lowest queues. This can be seen in the text message at the bottom of Fig. 5.4. The first responder has free will to choose the hospital that she/he feels has the best chance to serve the patient fastest.

![Sensor ID 129 Press c to carry](image)

"<No response>"
Breathing Irregular and labored
Pulse Racing and weak
Penetrating chest wound
Right chest

Fig. 5.4 Scenario 2 Smart Agent

5.2 ASSUMPTIONS FOR THE EXPERIMENT

The following assumptions were made during these experiment runs:
• Scenario starts 8 hours after Portsmouth event when first responders arrive at the plane crash and triage starts at a simulation time of 8 hours;
• Scenario ends after 24 hours of hospital simulation time;
• There are 2 hospitals that are equidistant from the plane crash;
• Each hospital has the same number of resources, layout of facilities, and capabilities;
• Hospital 1 is closer to the Portsmouth mass casualty event;
• The system starts idle at the beginning of each day;
• Patients arrive at the hospital as "walk-in patients" or using the hospital ambulance;
• Patients always arrive as individuals, never more than one patient at the same time;
• Entities’ travel time between different areas in the ER was not considered;
• Entities’ travel time between the plane crash and the hospital was not considered;
• There are an unlimited number of ambulances available for the scenario and they are always available;
• An ambulance can be occupied by one patient only;
• A room can be occupied by one patient only;
• The role of the ER nurses was omitted;
• There are always doctors in each Hospital ER;
• There is one first responder at the accident scene doing triage;
• Technology insertion provides better than or equal to current capabilities during Experiment with Arena Sending Hospital Information versus the Base Case;
• The first responder’s PDA has full connectivity and access to all data at the hospitals;
• The first responder and/or agent application on the PDA does not use prior knowledge of previous patient’s hospital assignment or hospital’s statistics to make their decision, only uses instantaneous knowledge and data;
• Hospital systems are always monitoring current statistics at the hospital;
• Patient Total Time is calculated from the moment a patient is triaged by the first responder and the time she/he leaves the ER;
• Patients triaged 3 and 4 or ones that die are not considered in calculations.

5.3 EXPERIMENTAL DESIGN AND OUTPUT ANALYSIS

The following experiments will help answer the research questions presented earlier and determine if the new configuration of the simulation environment will allow a first responder to make a more informed decision to ultimately reduce the average time it takes a patient to get through the ER. Experimentation between the alternatives will produce the data necessary to allow us to evaluate whether one is better than the other.
“Experimentation contributes to and thus advances a body of knowledge that, when applied, allows us to develop new capabilities. Experimentation involves not just the conduct of experiments but also the following elemental tasks of knowledge acquisition: development of a theory, construction of a conceptual model that embodies the key elements of the theory, formulation of questions (Descartes’ doubts) and hypotheses (Bacon’s analytical foci), collection of evidence, and analysis [31].” I have formulated the research questions and I need to create my hypothesis and test it through experimentation. This experimentation process will be thoroughly implemented and discussed throughout the following section.

The overall objective of this thesis is to use experimentation to explore the utility of the newly developed serious games-based simulator for evaluating first responders’ needs during a mass casualty event.

**Hypothesis:** Providing real-time hospital status information to the first responder will enable patients to obtain a faster average total time through the region's Hospital ERs.

**Null Hypothesis:** There is no statistically significant difference in the average total time it takes a patient to be served at a given hospital's emergency room when first responders have a PDA which provides real-time hospital status information.

The **Key Performance Indicator (KPI)** for these experiments is called Patient Total Time -- the amount of time it takes a patient to get from triage assignment through a hospital’s emergency room.

One of the real utilities of simulation lies in the ability to compare several different alternative system designs or operating policies. Simulation is used to generate a better understanding, expected performance, and effects of the alternative system
configurations, operational activities, and/or procedures relative to a certain scenario. In order to determine which configuration is the best, statistical methods must be applied to analyze the output of simulation runs. This is done to avoid making serious errors and drawing fallacious conclusions, thus yielding poor decisions. The approach is to exhaustively simulate and analyze the alternatives before implementation to reduce this risk [26].

Due to the stochastic nature of simulation input, it is expected that the experimentation output will produce stochastic output. Therefore, the simulation must execute many replications for each configuration in order to obtain enough independent observations to produce statistically significant output data. Also, I try to reduce variation during the experiment by using the following techniques for each simulation run and for each scenario:

- I do not change the random seed;
- I triage the victims in the same order;
- and I make sure I completely evaluate the victims before proceeding to the next victim.

The paired-t confidence interval will be used to find this difference because it cannot be guaranteed that the samples across the alternatives are completely independent. The procedures of calculating a paired-t confidence interval start with selecting two alternatives to examine \( i = 1, 2 \). Then select the number of independent replications \( n \) over which Key Performance Indicator (KPI) values are going to be collected:

\[
X_{i1}, X_{i2}, \ldots, X_{ij} \quad \text{for } i = 1 \text{ or } 2 \text{ and } j = 1, 2, \ldots, n. \quad [26]
\]

Then, \( X_{ij} \) and \( X_{2j} \) can be paired to obtain:

\[
Z_j = X_{ij} - X_{2j} \quad \text{for } j = 1, 2, \ldots, n. \quad [26]
\]

This implies that \( Z_j \)'s represents the difference between the expected values (of the
selected KPI) between alternatives for a given replication, \( j \). A confidence interval will be constructed for difference between the expected values of both alternatives, or the \( Z_j \)'s. This confidence interval is defined as:

\[
Z(n) + \pm t_{n-1,\alpha/2} \sqrt{\text{Var}[Z(n)]} \quad [26]
\]

For this thesis, I will compare two configurations by constructing a confidence interval for the differences between their performance measures to see where the observed difference is significantly different than zero. The confidence interval will give us information so I can "reject" or "fail-to-reject" the null-hypothesis as well as provide quantified information as to how much the expectations differ, or the magnitude of difference [26]. The 95% confidence interval will tell us that for 95 out of 100 runs, the difference between average patient total times for Scenario 1 versus Scenario 2 will reside within the range of the interval. If analysis reveals that zero is not contained within the confidence interval, the null hypothesis will be rejected, because there will be statistical difference between Scenario 1 and Scenario 2.

5.4 RESULTS

The data suggests, simply when looking at the data's averages for each scenario, that providing real-time hospital status information to the first responder through the use of a PDA or hand-held technology will enable patients to obtain a faster average total time through a region's hospital ERs. This capability was analyzed using the integrated game and DES simulation technology. If I analyze the two scenarios independently, I can see that the average patient total time through a hospital's ER for Scenario 1 is 280.85 with a standard deviation of 13.74 while Scenario 2 is 238.07 with a standard deviation of
13.4. This data shows that there is an average savings of 42.78 minutes. However, to be statistically significant, the confidence interval must be taken into consideration. As discussed earlier, I use the paired-t comparison between Scenario 1 and Scenario 2. In Table 5.1, I compare the simulation output for each run and obtain a difference between Scenario 1 and Scenario 2. More specifically, I show the difference between average patient total time served by a first responder with and without the PDA for each run. As seen in Table 5.2, the 90% confidence interval suggests that there is a significant difference between the two scenarios because zero is not contained within the confidence interval.
Table 5.1 Average Patient Total Time for Thirty Independent Replications

<table>
<thead>
<tr>
<th>Run</th>
<th>X(S1)</th>
<th>Y(S2)</th>
<th>X-Y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>262.15</td>
<td>253.62</td>
<td>8.53</td>
</tr>
<tr>
<td>2</td>
<td>278.62</td>
<td>250.81</td>
<td>27.81</td>
</tr>
<tr>
<td>3</td>
<td>260.12</td>
<td>255.42</td>
<td>4.7</td>
</tr>
<tr>
<td>4</td>
<td>273.92</td>
<td>256.73</td>
<td>17.2</td>
</tr>
<tr>
<td>5</td>
<td>281.67</td>
<td>244.94</td>
<td>36.73</td>
</tr>
<tr>
<td>6</td>
<td>276</td>
<td>255.75</td>
<td>20.25</td>
</tr>
<tr>
<td>7</td>
<td>273.74</td>
<td>238.47</td>
<td>35.27</td>
</tr>
<tr>
<td>8</td>
<td>315.23</td>
<td>231.83</td>
<td>83.4</td>
</tr>
<tr>
<td>9</td>
<td>273.77</td>
<td>214.28</td>
<td>59.49</td>
</tr>
<tr>
<td>10</td>
<td>270.59</td>
<td>222.95</td>
<td>47.63</td>
</tr>
<tr>
<td>11</td>
<td>284.14</td>
<td>251.22</td>
<td>32.92</td>
</tr>
<tr>
<td>12</td>
<td>293.67</td>
<td>230.77</td>
<td>62.9</td>
</tr>
<tr>
<td>13</td>
<td>278.63</td>
<td>230.86</td>
<td>47.77</td>
</tr>
<tr>
<td>14</td>
<td>267.61</td>
<td>243.67</td>
<td>23.94</td>
</tr>
<tr>
<td>15</td>
<td>268.68</td>
<td>225.25</td>
<td>43.43</td>
</tr>
<tr>
<td>16</td>
<td>274.71</td>
<td>230.93</td>
<td>43.78</td>
</tr>
<tr>
<td>17</td>
<td>274.69</td>
<td>224.03</td>
<td>50.66</td>
</tr>
<tr>
<td>18</td>
<td>278.7</td>
<td>224.2</td>
<td>54.5</td>
</tr>
<tr>
<td>19</td>
<td>270.25</td>
<td>250.79</td>
<td>19.46</td>
</tr>
<tr>
<td>20</td>
<td>278.12</td>
<td>222.36</td>
<td>55.77</td>
</tr>
<tr>
<td>21</td>
<td>283.93</td>
<td>263.03</td>
<td>20.89</td>
</tr>
<tr>
<td>22</td>
<td>301.74</td>
<td>230.26</td>
<td>71.48</td>
</tr>
<tr>
<td>23</td>
<td>290.63</td>
<td>237.05</td>
<td>53.58</td>
</tr>
<tr>
<td>24</td>
<td>268.77</td>
<td>224.15</td>
<td>44.62</td>
</tr>
<tr>
<td>25</td>
<td>274.29</td>
<td>235.13</td>
<td>39.16</td>
</tr>
<tr>
<td>26</td>
<td>308.53</td>
<td>235.3</td>
<td>73.23</td>
</tr>
<tr>
<td>27</td>
<td>291.28</td>
<td>222.35</td>
<td>68.93</td>
</tr>
<tr>
<td>28</td>
<td>274.64</td>
<td>248.69</td>
<td>25.94</td>
</tr>
<tr>
<td>29</td>
<td>286.78</td>
<td>230.94</td>
<td>55.84</td>
</tr>
<tr>
<td>30</td>
<td>309.97</td>
<td>256.23</td>
<td>53.74</td>
</tr>
</tbody>
</table>

Table 5.2 90% Confidence Interval for Average Patient Total Time

\[
W(30) = 42.785 \\
\text{Var}[W(30)] = 395.574 \\
\sqrt{\text{Var}[W(30)]} = 19.889 \\
T_{29,0.95} = 1.699 \\
90\% \text{ Confidence Interval} = W(30) \pm t_{29,0.95}\sqrt{\text{Var}[W(30)]} \\
\text{upper bound} = 76.576 \\
\text{lower bound} = 8.993
\]
Table 5.3 95% Confidence Interval for Average Patient Total Time

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>( W(30) )</td>
<td>42.785</td>
</tr>
<tr>
<td>( \text{Var}[W(30)] )</td>
<td>395.574</td>
</tr>
<tr>
<td>( \sqrt{\text{Var}[W(30)]} )</td>
<td>19.889</td>
</tr>
<tr>
<td>( T_{29,0.975} )</td>
<td>2.045</td>
</tr>
</tbody>
</table>

\[
\text{95% Confidence Interval} = W(30) \pm t_{29,0.975}\sqrt{\text{Var}[W(30)]}
\]

<table>
<thead>
<tr>
<th></th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>upper bound</td>
<td>83.458</td>
</tr>
<tr>
<td>lower bound</td>
<td>2.112</td>
</tr>
</tbody>
</table>

The 95% confidence interval suggests that for 95 out of 100 runs, the difference between average patient total time for Scenario 1 versus Scenario 2 will reside within the interval between 2.11 minutes and 83.46 minutes. This analysis also reveals that zero is not contained within the confidence interval; therefore, I reject the null hypothesis because there is a statistical difference between Scenario 1 and Scenario 2. With the paired-t confidence interval I prove that providing real-time hospital status information to the first responder will enable patients to obtain a faster average total time through the region’s hospital ERs.
CHAPTER VI

CONCLUSIONS

6.1 CONCLUSION DISCUSSION AND CONTRIBUTIONS

The overall objective of this thesis was to use experimentation to explore the utility and show the usefulness of the newly developed serious games-based simulator for evaluating first responders' needs during a mass casualty event. More specifically, the serious games-based simulator was proven to be useful when comparing a mass casualty scenario where a first responder does not have information about hospital statistics and a scenario where the first responder has a handheld mobile communications device, or PDA, that provides real-time hospital status information. The goal of this experiment was to evaluate the average patient total time resulting from the first responder with and without using updated hospital status information.

This thesis proved that I can analyze and optimize real-time decisions of a first responder to a mass casualty event with modeling and simulation. I can integrate DES and game technology to evaluate the system/operational architecture and first-person decision making, respectively. Also, the integration of these two technologies can provide analysts with enough data to evaluate scenarios from many perspectives including systems engineering, data engineering, and human factors. All of the research questions have been addressed and have been evaluated using the serious games-based simulator. It has been determined that this technology can allow us to analyze and optimize real-time decisions of a first responder to a mass casualty event through modeling and simulation tools such as DES and video games, architecture methodologies, and systems engineering
processes. Video games are useful and interesting tools for real-time training and analysis of first responders in a mass casualty event. This technology provides an environment which allows the user to experience these types of scenarios from a first-person perspective in a repeatable and traceable fashion. I can run through the scenarios many times and evaluate the decisions and information needs. As a first responder, I learned that information about the status of the regional hospitals was useful when responding to an event with mass casualties. More specifically, modeling regional hospitals' ER operations and architectures in a DES provides enough information for a first responder in a video game environment to have increased situational awareness, thus yielding better decision making and increasing the number of lives saved. I have proven that there are programming techniques to simulate a scenario faster than real-time in a DES tool, then slow it to real-time for the first-person interaction in the video game, then run faster than real-time when completed to obtain the results and analyze process flows, system event traces, and messages.

The major contributions for this thesis include an integrated simulation capability between Arena DES and Unreal Tournament game engine and a framework for analysis of first responders' information needs when triaging victims at a mass casualty event that served to provide these answers.

6.2 FUTURE WORK

Although the results of the analysis suggest that there is a significant difference between the two scenarios, there is still a need to complete more simulation runs and evaluate more data. This is because the confidence interval is larger than expected, which
suggests that more data needs to be collected or the key performance indicator needs to be scoped differently. Thus, further analysis will be needed to completely evaluate the alternative configuration with the PDA.

There are also some enhancements that could be made to more robustly test the serious games-based simulator capability. The video game could be used to introduce more people into the scenario to conduct triage collaboratively or allow players to assume different roles. Also, better user interfaces could be developed to test data visualization for the first responder. The analyst can change the type or amount of data displayed to the first responder to better understand data needs and usefulness. The video game players can use the chat or voice capabilities of the video game engine to communicate with others. Alternatively, voice over IP or video teleconferencing technologies could be integrated into the game engine to enable communication with other actors outside of the video game. The DES models of hospitals should be tailored to truly represent different patient flows, staffing requirements, and technologies. Hospitals can be added to simulate more than just two hospitals in the scenario. I can also use DES models of emergency operations centers and other operational nodes that participate in this scenario.

Future work should also specify a generic architecture and identify other research questions that can be evaluated with this serious games-based simulator framework. Other domains and scenarios could be evaluated with this framework such as military (urban operations and peace keeping), medical (hospital staff, surgeons), education, and homeland security. The generic architectural framework would make a more robust, plug and play simulation environment in order to support other DES tools and video game engines. This framework may also leverage a different simulation communications
protocol other than sockets, such as HLA and DIS, to connect other types of virtual and constructive simulation engines such as JSAF, Crowd Federates, JWARS, flight simulators, etc. The framework could allow connectivity to real world operational systems, prototypes, hardware in the loop, web technologies (WebEOC, Google Earth), communications, etc.
REFERENCES


[9] Phongsak, Prasithsangaree, “UTSAF: A Multi-Agent-Based Software Bridge for Interoperability between Distributed Military and Commercial Gaming Simulation,” Department of Information Sciences and Telecommunications, University of Pittsburgh,


[12] HTTP://BF2.FUN-O-MATIC.ORG/INDEX.PHP/LEARNING_PYTHON


[14] HTTP://WWW.DELTA3D.ORG/INDEX.PHP


[27] HTTP://WWW.TCNJ.EDU/~MCGRATH5/SEERS.BACKGROUND.HTM


[33] Fatemah Aldouli, “Applying A Simulation-Based Real-time Decision Making
Method to Mass Casualty Scenarios in Hospital Emergency Departments”, Old Dominion University, 2005.


[35] HTTP://GAMEBOTS.PLANETUNREAL.GAMESPY.COM/INDEX.HTML

[36] HTTP://UNREAL.EPICGAMES.COM/UNREALSCRIPT.HTM

[37] HTTP://SOURCEWARE.ORG/PTHREADS-WIN32/

[38] HTTP://UTBOT.SOURCEFORGE.NET/INDEX.SHTML


[41] HTTP://PLAY2TRAIN.HOPTO.ORG/

APPENDIX A – ARENA TASKS BLOCK

“To specify that a message should be sent, the following expression was entered in the duration or velocity field of the construct: TASKID (Value, TaskID [, TimeOutInterval][, ErrorLabel]), where the parameters are defined as follows:

- **Value**—Expression defining the duration or velocity of the construct for simulation purposes.
- **TaskID**—Name or number of the TASKS element to be executed (i.e., message to be sent) if Arena is running in execution mode.
- **TimeOutInterval**—Maximum time (in base time units) the entity will wait for a response from the external process after the simulation of the delay or transfer has been completed. The default is infinity. If this field is specified as NOWAIT, a message will be sent but the entity will not wait and no reply is expected or allowed.
- **ErrorLabel**—Block label to which the entity is redirected if a timeout occurs (defaults to next block).

If the duration or velocity field of a module is specified using the TASKID expression, then the actual delay or transfer time of the entity is either the Value specified (if Arena is not running in execution mode) or the actual task time that occurs in the "real" system to which the model is directly interfaced (if Arena is running in execution mode).”[40]

“The logic of a TASKID expression is executed as two threads in parallel. The first thread emulates the delay or transfer time using the specified Value. The second thread executes the real-time task by sending a message to the real system to start an activity; it then waits for the system to respond with a "task completed" message. If the execution thread finishes before the emulation thread, the emulation thread is terminated and the entity departs the block. If the emulation thread finishes first, the entity remains suspended in the block until either (a) the execution thread completes, or (b) the actual task time exceeds Value by an amount that is greater-than-or-equal to the TimeOutInterval, in which case the task is terminated with a timeout error and the entity is sent to the block specified by ErrorLabel[40].”
APPENDIX B – ARENA DELAY BLOCK

"The syntax is as follows:

DELAY: TASKID(UNIF(1,2),ProcessPart),,Other;

If Arena is not running in execution mode, then just simulate a delay of UNIF(1,2) base time units. If Arena is running in execution mode, send the message associated with TASKS element ProcessPart upon entering the DELAY block. Then simulate a delay of UNIF(1,2) base time units in parallel with the "real" task. If the simulated delay finishes before a response from the external process is received (signifying that the task has been completed in the real system), then wait an infinite amount of time at the block for the response.

DELAY: TASKID(0.0,OrderArrival,NoWait),,Other;

If Arena is not running in execution mode, then just simulate a delay of 0.0 base time units. If Arena is running in execution mode, send the message associated with TASKS element OrderArrival upon entering the DELAY block. Simulate a delay of 0.0 base time units and exit the block. A response to the OrderArrival message is not expected from the external process.

DELAY: TASKID(UNIF(1,2),ProcessPart,5),,Other;

If Arena is not running in execution mode, then just simulate a delay of UNIF(1,2) base time units. If Arena is running in execution mode, send the message associated with TASKS element ProcessPart upon entering the DELAY block. Then simulate a delay of UNIF(1,2) base time units in parallel with the "real" task. If the simulated delay finishes before a response from the external process is received (signifying that the task has been completed in the real system), then wait an additional 5 base time units for the response. After that time, if a response from the external client still has not been received, exit the DELAY and proceed to the next block."[40]
APPENDIX C – ARENA RECEIVING MESSAGES

“When an external process sends a message to an Arena model, the message is one of two types:

- A response to an entity task (i.e., a response to a message that was sent by an entity when it executed a construct’s TASKID expression).
- An unsolicited message to Arena.

Arena requires a fixed format for both types of incoming messages. These formats are described below. The format of a task response message from the external process must consist of three integer fields:

```
0 EntityNumber ReturnCode
```

EntityNumber must be the IDENT of the entity that sent the message the application is responding to. This should always be included as one of the tokens in the task string sent by the entity. ReturnCode is an integer return code used to provide Arena with additional information regarding the response (e.g., whether or not the task was successful). The return code field is assigned to the global variable J when the entity associated with IDENT departs the logic module or block. You may then use a BRANCH block or DECIDE module to branch the entity based on the value of J. The protocol requires at least one space between each field. A Response Message example is below:

```
0 5 1 {Response to message sent by entity number 5 with a return code of 1}
```

To define the possible unsolicited messages that may be sent by the external process to Arena, use the ARRIVALS element. The format of unsolicited messages is as follows:

```
MessageType [Assignment1 Assignment2 ... AssignmentN]
```

MessageType is an integer greater than zero. This field is associated with the Message Type operand in the ARRIVALS element, and indicates which unsolicited message is being sent and thus which ARRIVALS logic should be executed. Assignment is a set of optional numeric parameters that are assigned to the attributes of the entity(s) created by the ARRIVALS element. Which attributes get assigned are defined in the ARRIVALS element. Note that the FIRST attribute defined in the ARRIVALS element is always assigned the value MessageType. The protocol requires at least one space between each field. Below is an Unsolicited Message example:
The corresponding explanation is: Execute entity arrivals for Message Type "3" in the ARRIVAL element. Assign the values "3","10.2", and "12" to the specified attributes of the created entity(ies)[40]."
APPENDIX D – ARENA EOC


“The model, as shown in Fig. 3.2, includes staff positions for command, operations, planning, logistics and finance and is instantiated for several local city and state operations centers including the Department of Health (DOH). Any staff position in any instance of an EOC has physical resources identified as fax, e-mail, telephone, computers, and meeting rooms.

![Example EOC Model](image)

A statistical analysis of the EOC staff activities can be performed using a Master Scenario Events List (MSEL) which is a list or table that chronologically lists and synopsizes key events and responses with scenario times and objectives as described in USDHS (2003). This data provides the timing and frequency for EOC staff activities. The model allows for such activities to include interoffice communication via phone or email, computer work to be completed such as the filling out of the necessary computer generated forms and procedurals, and the entire decision making processes needed within a given scenario. Within this incident an example of a decision made by the EOC model is the timing of the opening of the aforementioned distribution center.

Arena allows for model templates to be defined which have graphical user interfaces (GUls). This feature can be used to allow the user to select the resources each staff member has and the type of
contacts, communications, and activities performed by the EOC staff member. This allows the user to change any of the given parameters without actually having to open the model itself.

During the hypothetical scenario, a local city EOC is initially stimulated by the hospital model, which generates reports when hospital overcrowding conditions are reached. The integrated activity is achieved by local EOCs requesting status reports for hospital, patients and supplies from overcrowded hospitals. Communications are generated between local and state EOCs as the incident progresses. A generic event routing capability was developed to provide the flexibility necessary to allow the user to define EOC staff communication “connectivity”.

The hospital model includes waiting room, patient check in and check out, examination, bedrooms and patient transfers when overcrowding conditions exist. As the simulation executes, runtime statistics are displayed for each of the EOCs indicating staff load versus time. The EOCs statistics include utilization of the resources, fax, telephone, computer, e-mail, conference room and verbal conversations by staff position and location. The hospital model statistics include staffing and supply resource utilization costs. Utilizations and staffing schedules will be collected for doctors, nurses, receptionists, examination rooms and beds. Other resources such as supplies can be modeled including bandages, cotton balls, needles, petri dishes and antibiotics.”
VITA

JASON LOVELAND

EDUCATION

Master of Science (Modeling and Simulation), Old Dominion University, Norfolk, VA, May 2008
Bachelor of Science (Computer Engineering), Minor in Computer Science, Villanova University, Villanova, PA, May 2003

EXPERIENCE

Lockheed Martin Corp., Center For Innovation, Suffolk, VA
  Systems Engineer, December 2003 – current
Jacobs Engineering Group Inc., Conshohocken, PA
  Systems Engineer, June 2003 – December 2003
University Information Technologies, Villanova University, Villanova, PA
  Technical Repair and Help Desk, September 2001 – May 2003

SCIENTIFIC AND HONOR SOCIETIES MEMBERSHIP

  Member of Institute of Electrical and Electronics Engineers