Virtual training system for military operations on urbanized terrain

Ting, Shang Ping

2006

Ting, S. P. (2006). Virtual training system for military operations on urbanized terrain. Master's thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/35740>

<https://doi.org/10.32657/10356/35740>

a529461 / 154
2X
D/E/uc

VIRTUAL TRAINING SYSTEM FOR MILITARY OPERATIONS ON URBANIZED TERRAIN

TING SHANG PING

School of Computer Engineering

A thesis submitted to Nanyang Technological University
in fulfillment of the requirements for the degree of
Masters of Engineering

2006

ACKNOWLEDGEMENTS

I hope to thank my supervisor, Dr Zhou Suiping, for the invaluable advice and guidance during the term of the project. Throughout the project, he enlightened me with knowledge that I would not have comprehended. His passion for simulation research and approachable character has injected enthusiasm into my research. The numerous discussions with him have helped to me achieve a deeper depth in computer science and allowed us to provide useful contributions to the research community. Without his consistent support and patience, it will have been impossible for the project to be enriching and fulfilling.

TABLE OF CONTENTS

Acknowledgements i

Summary..... viii

Chapter 1: Introduction..... 1

1.1. Motivation for MOUT Research..... 1

1.2. Background..... 2

1.3. Purpose and Scope 5

1.4. Resources..... 5

1.5. Overview of Report 5

Chapter 2: Literature Review 7

2.1. Evolution of MOUT..... 7

2.2. Overview of Computer Games used for Simulation Research 8

2.2.1. Game Engines..... 9

2.2.2. Review of Game Engines 15

2.3. Case Studies..... 17

2.3.1. Full Spectrum Command..... 17

2.3.2. Attention Disorders in a Virtual Classroom..... 19

2.3.3. Analysis of Case Studies..... 20

2.4. Physics Simulation 20

2.4.1. Karma Engine 21

2.4.2. Qualitative Physics 22

2.5. Software Architecture..... 24

2.5.1. Simulation Interoperability Standards Organization..... 25

2.5.2. Base Object Model (BOM) 26

2.5.3. Joint Modelling and Simulation Environment for
Wargaming and Experimentation Labs (JEWEL) 27

Chapter 3: Twilight City Design..... 29

3.1. Analysis of MOUT scenarios..... 29

3.2. Virtual City Planning..... 31

3.3. Training Requirements..... 32

3.4.	Special Effects	33
3.5.	Interactive Voice Control	33
3.6.	AI and Bots Issues	33
3.7.	Customized Animation and Models.....	35
3.8.	Physics Simulation	35
3.9.	Architecture for Rapidly Re-configurable Simulations	36
3.10.	AI Framework for Group Coordination	37
Chapter 4: Implementing Twilight City.....		38
4.1.	Construction of the Environment.....	38
4.2.	AI Bots Paths Allocation.....	39
4.3.	Speech Synthesis and Recognition	41
4.3.1.	Speech Input Program	42
4.3.2.	Commands Input.....	43
4.3.3.	Interaction Class.....	43
4.4.	Modifications	43
4.4.1.	Bots Visualiser.....	44
4.4.2.	Geo-spatial Referencing.....	44
4.4.3.	Lightning, Thunder and Rain	45
4.4.4.	Night Vision	47
4.4.5.	Parachute	48
4.4.6.	Air Strikes	49
4.5.	Vehicles	50
4.6.	Improved Bots Movement	52
4.7.	AI Framework for Group Coordination	52
4.8.	Integrated Bots Control Interface	54
Chapter 5: Qualitative Approach For Physics Simulation		56
5.1.	Qualitative Behaviour Simulation	56
5.1.1.	Software Architecture	56
5.1.2.	Object Behavior.....	58
5.2.	Example Results	60
5.3.	Practical Applications	64
5.3.1.	Collapsible Structures	65

5.3.2. Road Blocks	67
Chapter 6: Architecture for Rapidly Re-configurable Simulations.	71
6.1. Motivation.....	71
6.2. Overview	72
6.2.1. Architecture for Rapid Configuration	74
6.2.2. Game-based Objects Standard Interface	76
6.2.3. ARC-Based Objects Repository	78
6.3. Example results	79
Chapter 7: Experimentation Results	83
7.1. Interactive Voice Controls	83
7.1.1. Analysis and Comments.....	83
7.2. Evaluation of the Modifications made	84
7.3. Benchmarking	85
7.4. Mean Opinion Score Tests.....	87
7.5. Frame Rate Analysis.....	90
Chapter 8: Conclusions and Future Work.....	93
8.1. Summary.....	93
8.2. Publications.....	95
8.2.1. Journals.....	95
8.2.2. Conferences.....	96
8.3. Future Work	96
8.3.1. Interactive Virtual Environment for the Visually Impaired	97
8.3.2. Software Architecture	97
8.3.3. Improved Navigation System	98
8.3.4. Test beds with Specific Behaviours.....	100
8.3.5. Steering Behaviour for Autonomous Agents	101
8.3.6. 3D Virtual Worlds	103
8.3.7. Conducting a Game AI Course for NTU	103
References	105

LIST OF FIGURES

Figure 2-1: Modular game engine structure	9
Figure 2-2: Quake's architecture	11
Figure 2-3: Counter Strike – Modified Game using Quake Engine.....	12
Figure 2-4: Unreal Tournament Game	13
Figure 2-5: Unreal Architecture	14
Figure 2-6: Jupiter Effect & Infiltration	15
Figure 2-7: Full Spectrum Command	18
Figure 2-8: Virtual Classroom Commercial Version.....	19
Figure 2-9: Movable Objects	21
Figure 3-1: Twilight City Top View.....	32
Figure 4-1: Twilight City.....	38
Figure 4-2: Interactive Voice Control Process	42
Figure 4-3: Radar Interface in Twilight City	44
Figure 4-4: Geo-Spatial Referencing.....	45
Figure 4-5: Lightning and Rain Effects in Twilight City	46
Figure 4-6: Night Vision in Twilight City	47
Figure 4-7: Parachute in Twilight City.....	48
Figure 4-8: Air Strikes in Twilight City.....	49
Figure 4-9: Vehicles in Twilight City	51
Figure 4-10: Follow (Top left), Gather (Top right) and Hunt (Bottom).....	53
Figure 4-11: Integrated Bots Control	55
Figure 5-1: Qualitative Physics Framework.....	58
Figure 5-2: Effects of applying forces at different contact points	60
Figure 5-3: Different behaviors of a barrel.....	59
Figure 5-4: Uses of Qualitative Physics in Twilight City.....	61
Figure 5-5: Qualitative Process of Collisions	62
Figure 5-6: Object behaviour in water	64
Figure 5-7: Collapse of structure during an air strike.....	66
Figure 5-8: Barricades and Tires	68
Figure 5-9: Barricades Layout	70

LIST OF FIGURES

Figure 6-1: Class Hierarchy of ARC based objects	73
Figure 6-2: Architecture for Rapid Configuration	75
Figure 6-3: Game-Based Object Standard Interface compliant classes ...	77
Figure 6-4: Development process for ARC-based objects	79
Figure 6-5: Qualitative Physics Framework.....	80
Figure 6-6: ARC- based Qualitative Physics	81
Figure 6-7: ARC- based QP Movable Objects.....	82
Figure 7-1: Shattered Glass In Twilight City	85
Figure 7-2: Mean Opinion Score Tests of Twilight City	88
Figure 7-3: Mean Opinion Score Tests Results.....	90
Figure 7-4: Frame Rate Analysis Results.....	92
Figure 8-1: Simple AI Architecture.....	98
Figure 8-2: Terrain with 8 waypoints connected between 2 areas	99
Figure 8-3: MOUT Airborne Operations as Future Developments	101
Figure 8-4: Flocking Behaviours.....	102
Figure 8-5: Hierarchy of Motion Behaviours in AI Architecture.....	103

LIST OF TABLES

Table 2-1: Summary of Features..... 15

Table 7-1: Benchmark Analysis.....87

SUMMARY

Military Operations in Urban Terrain (MOUT) is the term used to describe any type of military action where battles take place in an environment where man-made construction or high population density is the dominant feature, such as a city or town. This type of fighting requires strategies that differ immensely from fighting on other types of terrain, such as large deserts or jungles. Modern military forces exploit simulation technology to extend their capabilities for MOUT training and decision analysis. Although there are considerable efforts and time invested into this area, there are still many challenges facing existing simulation systems. To this end, we had identified several key areas for our research. These areas include developing new physics simulation techniques, designing flexible architectures and looking into innovative ways of building low-cost simulation environments. This thesis presents our research in developing new physics simulation techniques and designing rapidly reconfigurable software architectures to improve on the current state of the art in military simulation. At the same time, we will describe some of our work done in developing an interactive virtual environment test-bed for MOUT.

During the course of this project, we found out that existing MOUT simulations are lacking in realistic physics simulation. Thus, we developed and incorporated qualitative methods of realistic physics modeling at low computational overheads into our training system. Such qualitative physic simulation techniques can be further extended into other simulation environments.

In our efforts to create a rapidly re-configurable simulation system, we proposed a generic framework that can provide an agile and adaptable

simulation system that can be transformed easily to handle frequently changing MOUT simulation requirements.

Thus, we had adapted a commercial game engine to develop a virtual environment for multi-agent research. This innovative approach allows us to provide a high quality immersive simulation environment at low cost. Our success in building such a game engine based training system was noted by the Defence Science & Technology Agency (DSTA) and we are currently integrating our training system into their simulation systems.

CHAPTER 1: INTRODUCTION

1.1. Motivation for MOUT Research

MOUT (Military Operations in Urban Terrain) is the term used to describe any type of military action where battles take place in an environment where man-made construction or high population density is the dominant feature, such as a city or town [1]. This type of fighting requires strategies that differ immensely from fighting on other types of terrain, such as large deserts or jungles.

Fighting in urban terrain brings with it a host of challenges and obstacles that are not usually found on an open battlefield. High numbers of civilians, culturally important structures, and narrow streets and alleys, for example, make for difficult, dangerous fighting. The modern day soldier must contend with all kinds of distractions and stresses as he moves through a hostile city, such as differentiating hostile targets from innocents, constantly watching for sniper fire, and maintaining a state of heightened awareness at all times [2].

As civilian casualties and collateral damage must be avoided as much as possible, modern artillery and large-scale weapons are inappropriate for urban combat. For this reason, city fighting benefits the defender. Attackers lose any advantage of firepower and mobility. A city can ingest an invading army, paralyze it for weeks on end, and render it ineffective [3, 4].

Today's military forces are constantly refining their strategies for fighting in large urban environments. An invading army that needs to conduct operations in a hostile city must have a good grasp of MOUT strategies and tactics.

1.2. Background

In the past decade, 3D virtual environments have been successfully implemented in games and applications. However, there are only a small number of applications available for research in simulation environments. Virtual environments can be realized as 3D worlds incorporated with real life physics. They would possess a certain level of AI, collaborating with humans to achieve a certain objective. With the advancement in technology, the solutions available for building such systems are expensive and extremely difficult to develop. Fortunately, many researchers have discovered a number of alternatives. Research in simulation needs a rich environment that is complex and dynamic, making the study of human behaviour and collaborative agents possible. Such environments had been found in computer games running on personal computers (PCs). First Person Shooter (FPS) games are such games that are reputed to be extremely realistic. Game engines that provide that realism have also evolved to be the attractive choice to use as part of a virtual environment research.

One of the major considerations in building a virtual MOUT training environment is the fidelity of the virtual environment. As movable objects such as bottles, boxes and chairs, etc. are quite common in real-life, it is important to incorporate these objects in the virtual environments. However, in the existing MOUT simulations, besides the avatars of the soldiers and some vehicles, most objects are static. The reason for this is that modelling and implementing movable objects is a challenging task in terms of computational costs and mathematical complexities. Traditional methods to implement movable objects are very time-consuming as they are based on exact physics models and numeric analysis [5]. The responsiveness of the system may be greatly affected by the behaviour of the movable objects.

The usage of virtual environments in simulation studies has been widely investigated based on different platforms. A team of researchers from Carnegie Mellon University and University of Southern California worked together to develop Gamebots, which is a virtual reality platform that allows the simulation and evaluation of intelligent behaviours. Marc Cavazza and his team of researchers successfully used the Unreal Engine to create a simulation environment for qualitative physics. However, their research area did not focus on MOUT.

In our work, we adopt an approach of using qualitative physics to model the behaviours of movable objects in MOUT simulations. This approach will help to reduce the computational cost of simulating the behaviour of movable objects and increase the fidelity of the system.

Considerable efforts are needed to construct a virtual environment for MOUT. The models of various objects used in one application are often specifically designed for that application, thus are not generally suitable for other applications. We often have to rebuild many object models for a new simulation setup, which is very tedious and time consuming. To reduce the cost and time in building up different simulation environments, it is therefore highly desirable to have a repository of object models that are interoperable and extensible thus can be reused in different applications. To this end, we propose a generic model framework, which supports model interoperability and reusability for MOUT simulations. Our experiences with Twilight City, a virtual training environment for MOUT, show that the proposed framework is successful in creating a repository of models for various simulation setups.

In this work, the objective is to connect existing technologies in order to produce a research test-bed for Military Operations on Urbanized Terrain (MOUT). With this test bed, we seek to advance the state of

CHAPTER 1 – INTRODUCTION

the art of distributed interactive simulation, which improves on current simulation methods on MOUT.

1.3. Purpose and Scope

This project aims, firstly, to develop a virtual environment for simulation research on MOUT by using an affordable existing commercial game that has multiplayer capabilities and any programming language that supports TCP/IP¹ to be used as an interface to the agents. It also aims to offer a valuable research tool for Human-Level AI research for the students in NTU in the future. Secondly, we seek to develop new simulation techniques and architecture with the use of our test bed.

1.4. Resources

This project was fully implemented and developed by using software running personal computers. The resources used in the project are listed as follows:

- Unreal Tournament 2004
- Unreal Editor 3.0
- WOTgreal, The Editing Tool for the Unreal Engine. V 3.006
- 3DS Studio Max, The 3D Modeling and Animation Package
- UMark, The Benchmarking Utility
- Java

1.5. Overview of Report

The report is organized into 7 chapters:

Chapter 1 presents the background, purpose and objectives for developing the interactive virtual environment, Twilight City. It also covers the scope and limitation of the project.

¹ TCP/IP refers to Transport Control Protocol / Internet Protocol, which is used in today's information highway, the internet.

Chapter 2 introduces the game engines that are used in scientific research. The chapter also compares existing virtual environments in the market and examines the state of the art in simulation techniques and architectures used in the project.

Chapter 3 talks about the virtual environment design and also described the design of the simulation techniques used for the project.

Chapter 4 provides detailed description of implementing the simulation techniques and a short description on the construction of the Twilight City.

Chapter 5 describes the usage of qualitative physics within Twilight City to achieve movable objects with minimal computational overheads.

Chapter 6 explains the Architecture for Rapid Configuration which evolves the Twilight City into a rapidly reconfigurable simulation system.

Chapter 7 discusses the scenarios that are tested in Twilight City and the experimentation results obtained.

Chapter 8 draws conclusion of our research and discusses some potential direction for future work and enhancements.

CHAPTER 2: LITERATURE REVIEW

This chapter provides a review of current state of the art in MOUT simulations. As we intend to adapt game engines for MOUT simulations, an overview of relevant commercial computer games is done. Apart from the implementation and design of the simulation platform, there is a need to further advance on existing physics simulation and software architectures. Thus, qualitative physics simulation techniques and software architecture designs are discussed in this chapter as well.

2.1. Evolution of MOUT

Throughout the history of the military, terrain has played a pivotal role in influencing the outcome of battles. From amphibious assaults to fighting in wide-open deserts, to hacking through dense tropical forests, each unique landscape presents a new set of challenges and requirements [5].

In ancient and medieval times, a city's only real military significance was its fortifications and its garrison. Once these obstacles were overcome, a city ceased to be a major military impediment. Most of the actual fighting was done on battlefields far from city centers.

In modern times, however, as the world becomes more and more urbanized, most fighting now takes place in the cities themselves and these urban centers are proving to be major military obstacles that are not easily overcome. In addition to being large enough to completely block a strategic avenue of approach onto enemy soil, a city's population poses major security, administrative, and logistical problems for the invader.

Urban operations require a completely new type of warfare. Where

armies used to win battles based on their numbers and the destructive power of their weapons, today's armies now find that their old strategies do not work when fighting in urban environments. In a city setting, the enemy can attack from any direction, and then disappear back into the civilian population quickly. Heavy artillery and airspace superiority becomes ineffectual in a city, where civilian casualties must be avoided.

Today's military forces are discovering that the only way to effectively cleanse a city of its hostile elements is with the use of small, technologically superior squads of soldiers that excel in block-to-block combat. This method is much more dangerous for the invader, however, and requires highly trained and well-coordinated teams of individual soldiers.

Tactics for fighting in urban terrain are constantly evolving as tomorrow's challenges arise. Where the tank and heavy artillery used to reign supreme on the battlefield of yesterday, it is carefully coordinated squads of individuals that will determine a battle's outcome in the new urban battlegrounds of tomorrow.

2.2. Overview of Computer Games used for Simulation Research

Today, commercial game software running on Personal Computers (PCs) are giving simulation software a run for their money with regards to the high standards of graphics, physics and realistic simulations. During the past few years, the gaming industry had even outgrown the movie industry. With the evolution of PCs, the most complex rendering pipelines can be found in the market for less than S\$500. With First-Person-Shooter (FPS) games being able to provide a high-fidelity virtual environment, or at least a very close match, it can be a suitable environment for scientific research, simulations and applications [6]. We aim to raise awareness on the usage of such high-power and low

cost game engines in research. In fact, simulation research around the world has been using computer games for research [7, 8].

2.2.1. Game Engines

The rationale for using a game engine in the research is that it supports advanced graphic rendering and animation control which makes it an ideal development environment. The game engine is the most crucial component of a computer game. It handles rendering and include additional tasks such as AI, game physics, collision detection between game objects, etc. The most common element that a game engine provides is graphics rendering facilities (2D or 3D). In today’s context where PCs are so affordable, 3D rendering capabilities which produce realistic virtual environments allow complex animation and human-like behaviours to be generated. Game engines were also designed with the content developed separately. This separability allows game codes to be used in scientific research.

Figure 2-1: Modular game engine structure

In a game engine structure as shown in Figure 2-1, the game logic and environment level are not explicitly shown. The virtual world can be designed and interact with the game engine together with the game logic. This can be seen at the top level in Figure 2-1. In addition, these levels allow a complex environment and game logic to be developed quickly without the burden of graphics programming and creating motion physics of characters and objects.

In a multiplayer game, the engine can work with the server through the network code to render and create the global environment shared by all clients. There are many popular commercial game engines available, but two of the most popular ones are Unreal engine and Quake engine. They are reputed to provide an efficient and realistic framework to a FPS environment [9, 10, 11].

2.2.1.1. Quake Engine

Quake is a client-server application, as shown in Figure 2-2. All simulations are performed on the server, and all input and output take place on the client, which is basically nothing more than a specialized terminal. In multiplayer games, the client and server are separate processes, running on different machines [12].

Quake was designed from the start for multiplayer gaming. It uses reliable packet delivery only for information such as scores and level changes. The Quake server maintains the game's time base and state, performs object movement and physics, and runs the core AI. The most interesting aspect of the server is the extent to which it's data-driven. Each level (the current "world") is completely described by object locations and types, wall locations, and so on stored in a database loaded from disk. With Quake engine as the first generation

game engine, it uses the C programming language as its source programming language.

Figure 2-2: Quake's architecture

Quake only has a limited number of games that have been modified. This is due to the complexity of its modular game engine structure where modification is very difficult. Thus many coders only concentrate on creating the map/level and agents' intelligence. The game, Counter Strike as shown in Figure 2-3, would be most suitable for the 3D environment for this project.

Figure 2-3: Counter Strike – Modified Game using Quake Engine

2.2.1.2. Unreal Engine

The Unreal engine was originally released by Epic in 1998 under the “Unreal” title [13]. The engine is designed to allow third party developers to create their own games either by developing a modification that plays under one of Epic’s games, or by licensing the engine itself to produce a whole new game. Unreal engine basically powers the popular futuristic game, Unreal Tournament as shown in Figure 2-4. Unreal Tournament is an off-the-shelf extendable 3D game engine that supports networked play.

Figure 2-4: Unreal Tournament Game

Unreal introduces gaming with an approach termed the generalized client-server model. Furthermore, the "game state" is self-described by an extensible, object-oriented scripting language which fully decouples the game logic from the network code. This achieves a goal of object-orientation which allows extensibility without introducing dependencies on other pieces of code which are hard-wired to know about the internal implementation of that object. This is the most important aspect of the engine. As opposed to Quake engine, Unreal engine have its own scripting language, Unreal script. Unreal Script can be used for programmers to dwell into the engine's object-oriented model [14]. The engine's architecture is shown in Figure 2-5.

Figure 2-5: Unreal Architecture

The main goal of the network code is to enable the server to communicate a reasonable approximation of the game state to the clients so that the clients can render an interactive view of the world which is close to a shared reality with a reasonable given bandwidth limitations. Instructing the clients to execute their own functions does this.

Unreal Tournament can be modified into many games because the Unreal Engine allows different game plays or scenarios to be made. By class extension, the programmer can modify the game to whatever game play limited only by imagination. For example in Figure 2-6, Jupiter Effect is one of the modified games from a FPS to an aircraft simulation game. There are also modified games developed to be suitable for this project. One of it is Infiltration, which has been used in multi agent research in universities like Carnegie Mellon University (CMU) and University of Southern California [15].

Figure 2-6: Jupiter Effect & Infiltration

2.2.2. Review of Game Engines

Table 2-1 compares the two games against basic criteria and functionality of the systems.

Game engine	Quake	Unreal Tournament
Price	Aprox. \$20	Aprox. \$20
Response	Fast	Fast
High fidelity 3D environment	Yes	Yes
Level/Map editor	Third party software	Yes
Object Oriented Programming	No (modular)	Yes
Easy to understand	No	Yes
Supports multiplayer	Yes	Yes
Customizable Game plays (game type)	Yes	Yes
Suitability of modified games	No	Yes
Debugging Facility	Yes	Yes

Table 2-1: Summary of Features

From Table 2-1, there are differences between Quake and Unreal engine, and both provide high quality features for research purposes. Quake engine provides hard core coding for the programmer. It dwells in the C language for most part of the development and does not support Object Oriented programming (OOP). To modify the code and create a custom game play, it will be demanding for the inexperienced

programmer. Though it has a large community of coders and tools; its core language makes it difficult for inexperienced programmers to work on it. In addition, the game does not come with its own level editor tool.

In contrast, Unreal Tournament (UT) is designed for programming. Many tools and resources are available to everyone. One competitive advantage over Quake on programming with Unreal Tournament is that, the programmer can avoid needless reinventing while benefiting from a proven development platform. With OOP as the core programming technique, a programmer can make robust changes in game behaviour without requiring detailed knowledge of the involved code. The Java like scripting language, Unreal Script, is also easy to understand. The game even comes with a well-designed UnrealEd² development environment, a level editing tool. It is used as a Web 3D Authoring Environment in other area of research [16].

The choice is clear now that there is much more for the researcher in UT than in Quake. Both games have their strength and weaknesses; however the OOP aspect of UT has a greater advantage over Quake.

As UT was released in 1998, more powerful versions of the series such as Unreal Tournament 2003 (UT2003) and Unreal Tournament 2004 (UT2004) are currently in the market. UT2004 runs on the Unreal Engine 2.0 instead of Unreal Engine 1.0 as in UT. Besides that, the game editor is also improved from UnrealEd 2.0 to the UnrealEd 3.0. Although UT2004 has higher hardware requirements than Unreal Tournament 1999 (UT1999), UT2004 dominates the Unreal Tournament 1999 in all other areas.

² UnrealEd – A 3D Editing tool used to create maps/levels for the game.

After careful consideration, it was decided that Unreal Tournament 2004 be used as the environment for this project.

2.3. Case Studies

We study some case studies on how game technologies had benefited other areas such as military simulation and education.

2.3.1. Full Spectrum Command

Full Spectrum Command is a PC strategy game employed as a training tool to simulate battlefield co-ordination at the company level [17]. The training aid spawned from a technology partnering agreement between the US Army and Singapore Armed Forces. The game is designed to develop the infantry commander's cognitive skills, tactical decision-making, resource management and adaptive thinking through various scenarios. These scenarios are focused on asymmetric threats within peacekeeping and peace-enforcement operations. A scene in the game is shown in Figure 2-7. It allows the user to understand the importance of terrain analysis and the potential fratricide situations that can occur during a battle.

Figure 2-7: Full Spectrum Command

Key Features:

- Custom designed to reflect current operational tactics
- Head-to-Head multiplayer capability via LAN.
- User-level editor for scenario customization.
- Instructor evaluation mode for curriculum-based usage.

FSL 1.0 is developed by the Institute of Creative Technologies (ICT) in California, US. ICT was formed by the US Army and the University of Southern California, bringing together the Hollywood film community and Silicon Valley-based electronic games developers to work on state-of-art immersive training simulation. Besides the US Army, the Singapore Army is one of the few users to use FSL 1.0.

2.3.2. Attention Disorders in a Virtual Classroom

Research in simulation can be used for the analysis of clinical populations with central nervous system dysfunction. The University of Southern California's Integrated Media Systems Center and Digital Media- Works have partnered to develop The Virtual Classroom for the assessment of attention processes in children who suffer from attention deficit hyperactivity disorder. The Virtual Classroom, as shown in Figure 2-8, provides an efficient and scalable tool for conducting attention testing. Within this scenario, the child's attention performance will be assessed while a series of common classroom distracters were systematically presented.

Figure 2-8: Virtual Classroom Commercial Version

Building a virtual environment as visually and functionally realistic as possible is a primary goal. Budget and resource constraints spawned the innovative approach of using mainstream commercially available PC hardware and software. The team decided to use the Unreal game engine as the foundation for the application's real-time rendering component as such software would provide the production team with a rapid prototyping tool capable of producing a quality product without placing dramatic demands on available budget and resources.

2.3.3. Analysis of Case Studies

Evident from the case studies, the game engines and game environment are being used by researchers and the militaries to create simulation tools. This trend adds strength to the project objectives of employing a game engine to produce a virtual environment as a simulation tool. The lessons learned from the case studies are used to set the standards of the project development goals. This project seeks to provide a virtual environment that can add certain academic value to the MOUT simulation research of Nanyang Technological University.

The Full Spectrum Command is a powerful training tool to the US and Singapore armies but the program is mainly based on jungle terrain. Furthermore, the program not open-source and is only available to the military. There are bound to be high licensing costs for the School to acquire the software.

The virtual classroom provides a good case study for design of a modification to the Unreal Engine 2.0. The goal of studying the subjects through a simulation environment is similar to the objectives of the Twilight City.

Thus, the development of a simulation tool with the Unreal Engine 2.0 as a platform is a feasible approach. Furthermore, the implementation of this simulation tool, which focuses on MOUT environment instead of jungle terrain, will aid researchers in MOUT simulation and save huge costs at the same time.

2.4. Physics Simulation

One of the major considerations in building a virtual MOUT training environment is the fidelity of the virtual environment. As movable

objects such as bottles, boxes and chairs, etc. are quite common in real-life, it is important to incorporate these objects (see Figure 2-9) in the virtual environments.

Figure 2-9: Movable Objects

However, in the existing MOUT simulations, besides the avatars of the soldiers and vehicles, most objects are static. The reason for this is that modeling and implementing movable objects is a challenging task in terms of computational costs and mathematical complexities. Traditional methods to implement movable objects are based on exact physics models and numeric analysis, thus are very time-consuming [19]. The responsiveness of the system may be greatly affected by the behavior of the movable objects.

Due to limitations in computational resources, simulation of movable objects within a simulation environment is often neglected. We examined the state of the art in 3D physics simulation and study qualitative physics methods in greater detail.

2.4.1. Karma Engine

A physics engine is one that implements a general mathematical model of real-world objects and their interactions. The application configures the model and defines the objects within it, and the physics engine evolves the positions and velocities of the objects over time in response

to input from the application. A study on a commercial physics engine, Karma, is done.

Karma Engine is a physics and collision detection software package [18]. Software libraries provided contain routines that users can call on to quickly and easily add physical behaviour to their 2D or 3D environment. While the Karma product is suitable for a wide range of applications users should note that it is targeted at the games and entertainment markets. Karma is aimed at developers of real-time entertainment simulation software who are familiar with the C programming language and have a basic knowledge of maths. In broad terms Karma consists of collision detection and dynamic simulation modules that may be used alone or together. The Karma Bridge provides an API that simplifies the interoperation of Karma Dynamics and Karma Collision. A basic cross platform renderer that wraps the DirectX and OpenGL graphics libraries is provided. Karma computes exact physics and takes up a huge amount of computational resources. Thus, a more efficient technique such as qualitative physics should be used to improve on the existing state of the art.

2.4.2. Qualitative Physics

Qualitative physics is a research area that aims to provide a new repertoire of reasoning techniques for computers and automate human reasoning about the physical world. Qualitative physics attempts to model the real world using some qualitative rules rather than resorting to the exact physics [20, 21, 22]. According to the hypothesis of qualitative physics, commonsense reasoning and expert reasoning are similar enough to justify a unified treatment. The common challenge is to answer qualitative questions about complex systems based on partial

knowledge. The soccer player has only a rough idea of how the ball moves and the basketballer knows only approximately how high the ball

Kuipers sees qualitative reasoning as the task of formulating and analyzing dynamical systems that model commonsense or expert domains [55]. Like most qualitative physics researchers, he believes that ordinary differential equations are inappropriate for qualitative reasoning because they presuppose complete, precise models of dynamical systems, which are often unrealistic and unnecessary. People cross streets without knowing exactly how fast the traffic is moving or how quickly it can screech to a halt. Engineers design and repair artifacts whose physics is unmanageably complex or incompletely understood. Kuipers extends the language of ordinary differential equations with constructs that encode partial knowledge about variable values and about the structure of the equations. The task of qualitative reasoning is to formulate and analyze generalized equations.

New modeling languages have been developed with the progress of qualitative physics. These languages describe entities and processes in conceptual terms and embody natural notions of causality. Qualitative physics has been used in various areas. For instance, Marc Cavazza and his team used qualitative physics in digital arts to simulate various fancy behaviors of objects [23, 24]. In their work, they describe a new approach to the creation of virtual environments, which uses qualitative physics to implement object behaviour [25, 26]. Qualitative Process Theory was adopted as a qualitative reasoning formalism, due to its representational properties (e.g., its orientation towards process ontologies and its explicit formulation of process' pre-conditions). The system they describe is developed using a game engine and takes advantage of its event-based system to integrate qualitative process simulation in an interactive fashion. They use a virtual kitchen as a test environment.

Over the last decade there have been several important efforts aimed at using qualitative physics in digital arts but little effort has been our work, we have implemented various behaviours: physical object behaviour, complex device behaviour (appliances) and “alternative” (i.e. nonrealistic) behaviours, which can all be simulated in user real-time. As Marc Cavazza and his team were mainly focused on heat transfer and water flow simulation in their work, they did not consider simulation of movable object with qualitative physics. In our research, we seek to improve on simulation of movable objects through qualitative physics.

2.5. Software Architecture

Due to the rapidly changing needs of the military users, we realize that just building a good test bed is not enough. We need to have a scalable and flexible architecture that can be reconfigured swiftly to meet simulation requirements. Such a system is possible if composing existing models from a model repository can minimize development efforts. To achieve reusability and composability, the simulation systems must have interoperable models. Presently, model interoperability is a huge challenge due to the application specific nature of existing models. These models are built with a stovepipe development approach. As a result, replicates of existing models are continually being rebuilt for different applications. This needless reinvention causes costly and time consuming development iterations. To address these issues, an elegant architecture is needed to quickly develop virtual environments and support reuse from a common repository of interoperable and extensible models. We did a study on simulation interoperability issues and attended the Spring Simulation Interoperability Workshop 2006 in Huntsville, Alabama, USA. At the same time, we did a review on some existing systems that support simulation model reuse.

2.5.1. Simulation Interoperability Standards Organization (SISO)

The Simulation Interoperability Standards Organization (SISO) is an international organization dedicated to the promotion of modeling and simulation interoperability and reuse for the benefit of a broad range of M&S communities. SISO's Conference Committee organizes Simulation Interoperability Workshops (SIWs) in the US and Europe. SISO's Standards Activity Committee develops and supports simulation interoperability standards, both independently and in conjunction with other organizations. SISO is recognized as a Standards Development Organization (SDO) by NATO and as a Standards Sponsor by IEEE.

In 2003, the IEEE Computer Society Standards Activities Board (SAB) granted the SISO Standards Activities Committee (SAC) status as a recognized IEEE Sponsor Committee. SISO now maintains two major families of IEEE standards, IEEE 1278 (Distributed Interactive Simulation) and IEEE 1516 (High Level Architecture for Modeling and Simulation).

2.5.2. Base Object Model (BOM)

The Simulation Interoperability Standards Organization (SISO) focuses on facilitating simulation interoperability across government and non-government applications worldwide. One of SISO's interests is to explore methods that support and promote reuse of simulation components and encourage agile, rapid, and efficient development and maintenance of models.

Base Object Models (BOMs) provide a key mechanism in facilitating interoperability, reuse, and composability. BOMs are specifically identified in the IEEE 1516.3 High Level Architecture (HLA) Federation Development and Execution Process (FEDEP) as a potential facilitator for providing reusable model components used for the rapid construction and modification of federates and federations. The open standardization of BOM representations is considered essential for encouraging their development, distribution and use [27].

The BOM concept is based on the assumption that piece-parts of simulations and federations can be extracted and reused as modeling building blocks or components. The interplay within a simulation or federation can be captured and characterized in the form of reusable patterns. These patterns of simulation interplay are sequences of events between simulation elements. The implementation of the pattern using HLA object model constructs is also captured in the BOM [28].

Although BOM is an emerging standard for model interoperability, it does not support game engine based interoperability. In fact, there is not much work done in support game engine interoperability. In our work, we take the lessons learnt from BOMs and improve on our own game engine-based models.

2.5.3. Joint Modelling and Simulation Environment for Wargaming and Experimentation Labs (JEWEL)

As a small nation, Singapore has limited human resources, land and airspace. The strategic use of Modelling and Simulation (M&S) to help Singapore overcome these constraints is therefore crucial. Thus, Singapore formulated her simulation master plan, called the Vision for SAF Simulations (VSS), back in the mid-1990s. The Joint Modelling and Simulation Environment for Wargaming and Experimentation Labs (JEWEL) was conceived as the simulation environment in support of VSS [29, 30]. This enterprise-wide approach to simulation is analogous to what is happening in the business and command and control (C2) worlds.

Designed with reusability and interoperability as its primary precepts, JEWEL would be an open software environment that allows the incorporation of new technologies and standards from governmental, commercial and R&D bodies. It would be a launching platform from which new application needs can be satisfied accurately and quickly. To maintain openness and as a result future-proof JEWEL, DSTA believes that substantial attention must be devoted to its information architecture, both in terms of representation as well as content, as demonstrated in their adoption of HLA and Extensible Markup Language (XML), among other standards. JEWEL would support the Singapore Armed Forces (SAF) in training, analysis, experimentation and acquisition.

JEWEL is a collection of data and interface specification standards, frameworks and tools, and composable models and databases. Repositories of Models and databases refers to the physical set of reusable models and databases that are developed or acquired to perform specific simulation functions. A layer of frameworks and tools provide the glue essential for the different simulation tools to integrate seamlessly with one another. Some of these tools are available

commercially of the shelf while others are custom-developed for the JEWEL environment. In order to enhance the level of interoperability among simulation systems, and between simulation and C4I systems, it is important that these systems adopt consistent data and interface specification standards. This would minimise the problems associated with the interpretation and mapping of data as a result of representation differences.

Although JEWEL can provide a environment for model interoperability and support re-configurable simulations, there is no work done at the moment to enable game engine-based simulation model interoperability within JEWEL. As our training system is currently being integrating into the JEWEL environment, we need to consider the design of the interoperability of our game engine based models. Thus, as covered in Chapter 5, we hope to build on existing work and expand the present suite of interoperable models to include game engine based models.

CHAPTER 3: TWILIGHT CITY DESIGN

The goal of our *Twilight City* project is to create high-fidelity training and analyzing system for MOUT. A typical application scenario is the special squad operation for saving hostages held in a building by a group of terrorists. Various operation tactics need to be investigated before the real squad operation. The constructed virtual environment needs to resemble the real operation environment not only in terms of human's visual and audio perceptions but also in terms of the behavior of the non-player characters (i.e., AI bots). Creating such a high-fidelity virtual environment is a challenging task. Although there is some work on the application of FPS engines to military training, the design requirements may be different for different training tasks and scenarios. As pointed out by E. Lewis and M. Barlow in [31], "*The games must have sufficient fidelity to provide valid results. This fidelity does not have to be complete in all aspects, just enough in the variables of interest.*" In the following sections, we identify the major considerations on the design of *Twilight City*.

3.1. Analysis of MOUT scenarios

Military operations in Panama, Somalia, Kuwait and Iraq demonstrate the current and future requirements for modern military forces to be able to operate effectively in an urban environment. Operations in an urban environment will present unique and complex challenges. Our MOUT training systems should be able to support MOUT operations which are typically stability operations and support operations. include Peacekeeping Operations, Combating Terrorism, Noncombatant Evacuations, Nation Assistance, Civil Disturbance Operations, Humanitarian Assistance [52]. To succeed in battle in built-up areas, commanders and leaders at all levels must understand the nature of the environment. To assist commanders, MOUT simulations must

assist in the analysis of urban terrain on enemy forces, unaligned elements, and friendly forces [53].

Terrain analysis in an urban environment differs from that of open terrain in many respects. The analysis of the five military aspects of terrain--obstacles, avenues of approach, key terrain, observation and fields of fire, concealment and cover (OAKOC)--still applies [54].

Some of the important structures often involved in MOUT operations are listed below with some supporting reasons.

- Sewer and subway systems can provide infiltration routes.
- Elevated railways and mass transit routes provide mobility on which the urban residents depend; if operations destroy or disable these facilities, congestion will occur.
- Utilities such as electrical, gas, or water facilities may be key targets.
- While forces cannot attack hospitals and clinics when not under use for military purposes, they may be a source of medical support for all factions and elements.
- Stadiums, parks, and sports fields may serve as holding areas, enemy prisoner of war (EPW) facilities, or landing and pickup zones.

The soldier must somewhat alter analysis of the five military aspects of terrain (OAKOC) to consider fully the unique aspects of urban terrain. More than any other environment, the urban battlefield is dynamic. Depending on the street layout patterns, people can create or improvise manmade obstacles quickly to block narrow streets or these obstacles may not be a significant factor where streets are wider.

Natural obstacles arguably pose less of a problem in urban terrain than in open terrain. Rubble caused by direct or indirect fire may impede both mounted and dismounted movement. In relatively rare circumstances, rubble may actually aid movement, such as when a building collapses across a canal, thereby providing access to the other side. These are the types of factors that make the urban environment dynamic

Analysis of cover and concealment is also vital to success on the urban battlefield. Building characteristics, masonry, wood, brick, and even glass can all provide varying degrees of protection from observation, as well as the effects of weapons and munitions.

To build a good MOUT simulation system, it is not enough to describe the general characteristics of an urban area. The number of floors and rooms in a building are essential to determining the proper allocation of forces. A staff will not be able to allocate adequate resources to seize an objective or to isolate a series of buildings if the MOUT environment does not provide this level of detail. Troops need to react well towards surprise attacks, poor visibility, movable objects, smoke and noise. Apart from dealing with environmental effects, they must manage their voice communications, night vision, weapons and other equipments.

3.2. Virtual City Planning

The construction of the city environment took a lot of planning before the network of interconnected buildings and streets were finally built. Various technical and design considerations were necessary to produce the simulation system for the MOUT environment.

The city layout was planned with numerous interconnected buildings. A network of roads shall be placed in the city map to allow vehicular movement during the MOUT operations. Lifts will be placed in various

buildings to allow the soldiers an alternative movement route. Besides underground levels, a sewage system is placed into the environment as well. Following the principles of MOUT, some of the buildings are designed to act as a firebase or reconnaissance point.

Figure 3-1: Twilight City Top View

3.3. Training Requirements

The MOUT environment seeks to provide a test-bed for military simulation on urban terrain. The environment seeks to allow as much flexibility as possible to the potential users in terms of the types of MOUT operations that can be simulated within the environment. Some of the MOUT operations are hostage rescue, terrorists search, ambush and patrol. Taking the nature of the identified MOUT operations into account, high-rise buildings are designed to allow sniper attacks and reconnaissance posts. Buildings often complicate military operations in urban areas. Their composition, frontages, size, and window locations affecting troops positioning and weapons deployment considerations. Sewers should be built to facilitate the sewerage sabotage by the terrorists.

3.4. Special Effects

Special effects are needed to enhance the realism of Twilight City. These special effects can help to provide an immersive environment for the MOUT simulation. More importantly, the special effects such as smoke, shattering glass, water bodies and rain can help to the testing of different MOUT situations. By having shattering glasses, fire and smoke during an MOUT battle increases the effectiveness of the simulation and brings the trainee closer to real life operations. For example, smoke effects reduce the visibility of the soldier in the virtual environment and allows the trainee to experience the challenges of taking on the enemy when smoke screen is blocking him.

3.5. Interactive Voice Control

In most existing MOUT simulations, it seems that much more efforts are put on the visual effects rather than on the audio interactions. However, we believe that speech synthesis and recognition are integral parts of immersive MOUT simulations. With these functionalities, human players could communicate with the AI bots verbally, which may greatly enhance a human player's sense of immersion in the simulation. For example, instead of sending command messages to the AI bots in the squad through the keyboard, a human player may issue verbal commands to those bots. The bots can also respond to a human player "verbally" rather than by sending some text messages to the human player's screen. As will be described later, the human factor tests with the *Twilight City* show that these modules indeed help to enhance the realism of the virtual environment

3.6. AI and Bots Issues

Unreal Tournament 2004 supports the redefinition of object behaviours. The object-oriented language, Unreal Script, allows us to extend new classes, which can inherit properties of their parents while

adding new properties at the same time. User defined classes can be extended from the in-game classes and incorporated into the game play. These user-defined classes are powerful tools that are capable of affecting physical properties of the virtual environments and can be incorporated into maps to influence the AI logic. The user-defined classes extended from the in-game Mutator Class are commonly known as mutators.

For Twilight City to act as an efficient simulation tool, some features must be added to the systems. For example, the game engine can be modified to simulate the effects of tsunamis on the Esplanade or to test the efficiency of an evacuation plan during a gas attack during a concert in Singapore Indoor Stadium. These features can be added in the form of modifications to the game engine. Some of the useful modifications to game rules and bots AI are identified below.

Poison Gas Volume is made for the purpose of simulating the effects of a gas attack. The Bots Visualiser displays the locations of all the soldiers activated within Twilight City. Night Vision is a modification to the HUD to allow the testing of night mission with Night Vision Goggles. Parachute is a mutator that allows the simulation of airborne operations with parachutes in MOUT. Improved Bots Movements is a modification to the game engine to change the FSM structure of the bots logic in the game engine. This modification will result in a marked improvement in bots movement. The bots will become smarter and will demonstrate better movement such as increased dodges and jumps.

Mutators are used to control and adjust properties and events. However, an entire overhaul of the game engine can be done. This is called a Mod (Modification). The Infiltration mod created for Unreal Tournament is a powerful example of such a system [17]. The Wanted Man Mod created in this project can be used to simulate a MOUT operation in which a terrorist is being hunt down by the rest of the

soldiers. Lastly, the Hostages Rescue Operations can simulate MOUT hostage rescues.

3.7. Customized Animation and Models

The Unreal Engine is developed to support futuristic First-Person-Shooter games, so the embedded animations available are all standard game motions. For the MOUT simulations, there is a need to add custom animations to fulfill specific needs. The purpose of this area of research in this project is to explore and develop techniques for creating custom animations and models. To test the custom animations and models, a short clip will be made from the animations of some custom models. The animations and models created will aid greatly in enhancing the realism and accuracy of simulation. In the later sections, descriptions of the techniques explored and used to develop the animations will be covered.

3.8. Physics Simulation

Movable objects can increase realism during simulation. However, most simulated objects are non-movable due to the need to limit the computational overheads. To overcome this limitation, we applied qualitative physics to provide acceptable physics behaviors at a low computation cost. As this is considered a major contributed of our work, we shall cover this in detail in Chapter 5.

Traditional physics-based simulation is based on the kinematics equations of an object. The behaviour of the object is then generated by numerically solving these equations, which may need huge computational resources for real-time simulations. Thus, it may not be feasible to perform such precise simulations at a large scale in real-time applications like MOUT simulations. In most cases, achieving a rapid estimate of the behaviour can provide a lightweight yet *good-enough* alternative as compared to the precise solutions of the kinematics equations.

Qualitative physics aims to perform behaviour simulation at certain acceptable levels of realism and accuracy so as to fulfill some operational requirements while keeping the computational load low. In essence, qualitative physics applies our knowledge of the qualitative (causal) relations of various factors in the physical world to the behaviour modelling of the objects in the virtual environment [32]. Human beings are more sensitive to the qualitative (causal) relations among various objects and factors rather than on the exact physics of the objects. They rely on these relations in their reasoning about the environment and make decisions. Therefore, qualitative physics focuses on the changes in the environment, e.g., changes in water level, position changes, orientation changes, etc. In qualitative physics, changes in the environment are described in terms of *process*. Examples of processes are movement of water from a high ground to a low ground, heat transfer from an object with higher temperature to another object with lower temperature, etc. Such processes are activated due to the relationships that exist among the objects in the environment. A process is determined to be active or inactive by assertions of some preconditions and quantity conditions. The process specifications allow the system to automatically infer a set of active processes. An active process will then perform the necessary changes through its influences.

3.9. Architecture for Rapidly Re-configurable Simulations

Unreal engine is mainly a visualization engine, which can be supported by external applications. These applications can be qualitative physics engines to provide augmented reality. The external engines can be used to monitor bots actions or perform voice controls as well. Work should be done on the software architecture of the system so that Twilight City can demonstrate good extendibility and cooperation with external modules. Therefore, the project will come up with effective architecture designs on which the virtual environments are developed.

The main objective that our architecture seek to fulfill is to support rapid development of future environments according to the changing requirements of modern MOUT warfare. These architecture setups will serve as templates and guides for future researchers. Greater detail on our rapidly re-configurable architecture will be covered in Chapter 6.

3.10. AI Framework for Group Coordination

As the Twilight City will be employed for group size operations, it will be beneficial to set up a framework for group movements and tactics. Three types of group maneuvers are identified. They are namely Follow, Gather and Hunt. Follow refers to a movement when the bots follow a leader. Gather will command the bots to gather at a specified location. Hunt will release the bots to search for terrorists. The maneuvers will be performed through the use of the FSM structure of the bots and voice commands will be used for the activation. These group coordination techniques will be useful in the scenarios described in the next sections.

CHAPTER 4: IMPLEMENTING TWILIGHT CITY

This chapter describes the implementation of the Twilight City. Minor details like setting the virtual environment up will not be covered in detail as it will be a tedious procedure. The main emphasis of this chapter will be implementing the part of using this system as a simulation platform for MOUT operations. This will entice the reader or researcher about implementing MOUT simulations with Twilight City.

4.1. Construction of the Environment

The construction of a virtual environment is a tedious process, which involves many time-consuming and repetitive steps. A lot of the simple steps that can be picked by through online tutorials will not be covered here. Therefore, this section will focus on the major development of the environment building.

Figure 4-1: Twilight City

The above environment was built up from Unreal Editor 3.0 and most of the models were developed within 3D Studio Max [33]. The buildings were completed with interiors such as corridors, staircases, lifts and furniture. We built up an environment of an 8km by 8km area with a city in the centre and vegetation on the outskirts. There are 35 buildings in the city and the sizes of the buildings ranges from 2 to 15 stories. Vehicles, helicopters and ships are built in 3D Studio Max and imported into the environment as well. We have 4 types of ships including a submarine. We had added over 200 civilians and soldiers into the city to provide crowd simulation. The development of the environment and its models was done over 2 years and is an ongoing process. Construction of human models and vehicles are time consuming as behaviours are to be added via Unreal Script after the 3D models are produced.

4.2. AI Bots Paths Allocation

AI bots path allocation prevents the bots from standing in the corners and ignoring the rest of the entities. To be specific, PathNodes need to be laid out. Bots prefer wide paths and perform better when given lots of choices in paths selection. Bots will tend to take the shortest route when running to the mission objective. Thus, AssaultPaths should be set up so that the bots can coordinate their attacks among multiple routes. They are primarily used to recommend varied attack routes to the bases. The paths are assigned weights so that the bots can select a particular route. Leaving the paths all at equal weights and the bots should use them all equally.

Bots paths are built with the Unreal Editor. We can easily place path nodes into Twilight City with the Unreal Editor. We can place PathNodes (a subclass of NavigationPoint) on surfaces which agents can walk on, or in volumes which agents can swim in. PlayerStarts are also NavigationPoints, and they perform the same navigation function.

For PathNodes to connect, they should be less than 1200 Unreal units apart (programmers can modify MAXPATHDIST in UnPath.h to change this value). Having two NavigationPoints too close together (overlapping) can cause AI navigation problems and should be avoided. When placing PathNodes, the goal is to make sure that every area of the level is covered by a PathNode or some other NavigationPoint. An area is covered if an agent could walk to some PathNode less than 1200 units away completely unobstructed (i.e. without having to step around anything).

After placing PathNodes, we can build the connections between NavigationPoints by using the Build AI Paths option in the Build menu (or by doing a full rebuild). Once paths have been built, they can be seen in the various view windows by using the Show Paths option under the View menu.

Paths will be displayed as lines from one NavigationPoint to another. If agents can traverse the path in either direction, there will be two lines, with an arrowhead pointing in each direction. Otherwise, the line will show with an arrowhead in which direction the path can be traversed.

Even if the agents using the paths are small, it is always better to tweak the PathNode positions to make the connecting paths as wide as possible. agents will smoothly round corners, or strafe back and forth within a path, so larger paths will result in more natural looking movement.

Once a level has a large number of paths, it can take a while to rebuild all the paths. To tweak path placement, use the Build Changed Paths button in the Build Options menu. This will only rebuild paths between NavigationPoints which have been added, removed, or moved. Before saving and playing the level, however, a full path rebuild is required.

After building paths, a window may pop up in the editor with a list of pathing errors. Click on an error to take you to the offending `NavigationPoint`.

4.3. Speech Synthesis and Recognition

To enhance the interactivity of *Twilight City*, we have implemented a speech synthesis and recognition module using Microsoft's Speech SDK 5.1. We chose this toolkit for speech synthesis and recognition for two reasons. First, it has good APIs and is easy to use. Second, it reduces the code overhead required for an application to use the speech synthesis and recognition. Speech synthesis is straightforward by using the Text-To-Speech module in the toolkit. With the speech synthesis function, the bots can communicate with a human player by some simple words like "Yes, Sir!", "Danger!", and "I am on my way." etc. We believe that the voice synthesis functionality can greatly enrich the experiences of a human player in the environment, though it is still rather simple at the current stage of our project.

The implementation of the human voice recognition function needs much more effort. From Figure 4-2, we can observe that human voices are firstly received and translated into text messages. To this end, voice training must be done with Microsoft Speech SDK. Then, the built-in *Interaction* class of UT 2004 is extended so that it can intercept not only the keyboard inputs, but also the text commands from the voice module.

Figure 4-2: Interactive Voice Control Process

A series of tests have been done to evaluate the performance of the voice functionalities. Our first finding is that the voice recognition module performs best with commands of two or three syllables, e.g., “Attack!”, “Come to me!” etc. It is difficult for the module to recognize more complicated commands. On the other hand, if the command is too short, like “Jump!”, “Run!” etc., the results are more likely influenced by background noises. We also found that there is a positive correlation between the amount of training using the Microsoft Speech SDK and the response time of the bots to human commands. The response time could reach 3 seconds if the voice module is not trained properly. After sufficient and careful training, a bot could react to most of the human commands in the *Twilight City* within 1 second.

4.3.1. Speech Input Program

The framework of the speech function starts off with the user speaking into a microphone. The speech input is received by using Microsoft Speech SDK 5.1. To use this application, voice training must be done with the software. After sufficient training is done, the computer will be trained to recognise the voice of the user. The computer receives our

speech and converts the speech into text. This text is used to binds a verbal command to a keyboard button like a one-to-one function.

4.3.2. Commands Input

A module that reads the voice inputs from the Microsoft Speech SDK 5.1 and converts them to corresponding voice commands was incorporated into the framework of Twilight City. We then use scripting to set certain predefined events that will be activated once certain keys are pressed.

4.3.3. Interaction Class

The game codes are edited to correspond to key commands to bots actions. A mutator is made to extend the in-game Interaction class into a modified class that is able to detect and display each key is pressed while Twilight City is running. Interactions can intercept key input, and intercept string messages. This modified class is used to intercept the key events, which can be used to trigger actions.

Now Twilight City can be activated by voice commands. This voice control module allows the development of an interactive virtual environment, which can communicate verbally with the human player. With the use of this function and the overall architecture design, this function can be used as interactive tool to in the analysis of virtual environments by controlling bots maneuvers.

4.4. Modifications

The mutators controlling world properties and rules of the virtual environment are vital to simulation, AI testing, animation and even qualitative physics. In the following sub-sections, the modifications produced during the course of the project are discussed. The modifications are extended from the game engine and written in the Unreal Script before being compiled by the Unreal Engine 2.0.

4.4.1. Bots Visualiser

A radar system is created to allow the tracking of the bots in virtual environment as shown in Figure 4-3. The radar has a zooming function. The radar will display vehicles, players and game objectives. The radar rotates when turning. This radar system can aid greatly in the logging and analysis of team movements during simulations.

Figure 4-3: Radar Interface in Twilight City

The radar is placed on top of the head-ups display of the viewer and is implemented by adding a bots manager into the game engine. With this bots manager, the location of each individual bot is noted and display onto the radar screen.

4.4.2. Geo-spatial Referencing

Geo-spatial referencing is added to the Twilight City to allow the soldier to know the precise location they are at and the direction that they are facing. It is an important function that is specified by many military users. From Figure 4-4, we can see that the bot's position and orientation is at the bottom left corner and a horizontal artificial horizon is drawn across the screen.

Figure 4-4: Geo-Spatial Referencing

4.4.3. Lightning, Thunder and Rain

Techniques for simulating the lightning, thunder and rain are explored and developed in Twilight City. These are conditions that may determine the success of a MOUT operation as seen in Figure 4-5.

The lightning effect is created by using a pre-programmed emitter which makes use of a particle system to generate streaks of lightning across the sky. As lightning is always followed by thunder, the thunder effects were produced by playing a sound file after the lightning strikes.

To build the lightning effect, we first need to build a modified class call Lightning class and a random timed event. From the Unreal Engine, we can make use of the StochasticTrigger. With the StochasticTrigger, we can set minimum and maximum times and a probability within that range that an event will fire.

We can use TriggerLights class from the Unreal Engine can turn on, turn off, toggle on/off, or turn on for a time then shut themselves off. Subsequently, from the Unreal Editor, we can add a Dispatcher class

which can be triggered by the `StochasticTrigger`. This sets the Dispatcher dispatching the `TriggerLights`. Thus, with the visual effect from the `TriggerLight` class, lightning is created.

Figure 4-5: Lightning and Rain Effects in Twilight City

The rain is created by using an AI script which creates raindrops falling effect in front of the HUD (Heads Up Display) of each individual bots. Thus, the vision of the bots are blurred as expected during a rain. To set up rain, we need to add the customised Rain script into the R script delivers a limited range of raindrops. A few Rain classes may be necessary to create a heavy rain. To change the various settings for the rain, select all the Rain classes in your map and click on the properties dialog box, then click on the several Rain property menu's that appear(Rain_Behaviour, Rain_Looks, Rain_Sounds). The intensity variable should be kept between 0 and 5 or so to keep rendering speed good. DropSpeed should be negative for faster rain, and positive for slower rain, however beware that a positive integer greater than 100 or so will cause the raindrops to go up and hit the ceiling.

4.4.4. Night Vision

Through research and personal military experiences, NVG (night vision goggles) are an important tool in real life military missions. This is developed by altering the shading of the Heads Up Display (HUD) in Figure 4-6. By using image processing techniques of inverting the colour, we change the pixel values of the screen to make dark objects more prominent and to make the enemy almost white in colour so that they become more obvious.

Figure 4-6: Night Vision in Twilight City

The HUD is drawn on the Canvas. The Canvas is basically an object that draws thing directly on the player's screen. UT Canvas is similar to the Graphics object in Java applets. The HUD is basically a set of instructions that tell the Canvas object what to draw and where. The HUD is redrawn every tick. Whenever the HUD is redrawn, the "PostRender(Canvas Canvas)" function is called. This function is called by the PlayerPawn and gives us access to the Canvas class so we can draw things on it.

The whole HUD process starts with the "PostRender(Canvas Canvas)" function which is called about every tick (in other words, as often as possible). All the drawing is done by the Canvas object. Thus, the

image processing is then on the Canvas object and the visual effect of the NVG will then be displayed.

4.4.5. Parachute

The parachute allows the simulation of airborne operations with parachutes in MOUT. When activated, a parachute will spring out to break the descent of the soldiers during an airborne mission. The mutator alters the physics properties of gravity to reduce the speed and impact of the fall, Figure 4-7.

Figure 4-7: Parachute in Twilight City

The mutator is built by creating a parachute model with Unreal Editor and attaching it with behaviours from the Parachute class. The

Parachute class is extended from the underlying Mutator class via the Unreal Script.

4.4.6. Air Strikes

Modern warfare often involves bombardment via air strikes. To this end, we added air strikes capabilities into Twilight City. The users will be able to call for air strikes during their virtual MOUT missions. With the air strikes as in Figure 4-8, we can increase the realism of our simulations and add on to the variety of missions that can be simulated. By creating a Java adaptor that links the Unreal Engine to a Java-based 2D map representation of the environment, we can pinpoint the air strike locations easily.

Figure 4-8: Air Strikes in Twilight City

The Air Strike class is modified from the original Target Painter weapon in the Unreal Engine. Target Painter was originally used by the bots to

call for air strikes by painting onto exact points. We added a mutator to activate air strikes via a Java based external application that can control Target Painters in a god-like fashion. This means that air strikes can be called from anywhere via a 2D interface without the bots painting the exact point in the 3D environment.

4.5. Vehicles

Vehicles were designed and implemented into the Twilight City and can be utilized by the soldiers as part of their MOUT operations. The vehicles included into the virtual environment are light strike vehicles, medium strike vehicles and a custom made buggy. Furthermore, helicopters are included into the virtual environment to allow simulation of MOUT airborne operations, Figure 4-9. Motorcycles were also designed and incorporated into the environment. These motorcycles can carry a rider and a pillion each. We also made a suite of naval vessels to provide navy support for our simulation when needed.

Like the rest of the 3D models, all the customised vehicles are built from 3D Studio Max and imported into Unreal Editor. Behaviours are mostly extended from the original vehicle classes from the Unreal Engine and modified. These customised behaviours are then attached to the appropriate 3D models then added into Twilight City via Unreal Editor.

CHAPTER 4 – IMPLEMENTING TWILIGHT CITY

Figure 4-9: Vehicles in Twilight City

4.6. Improved Bots Movement

After studying bots logic and movement provided by the game engine, there is a motivation to adapt the FSM structure of the game engine towards MOUT operations. A class was extended from the bots controller module within the game engine. This class, the ImprovedFSM, will override the existing FSM architecture of the bots. The enemy bots will behave more like terrorists such that they will tend to run and hide before attacking the soldiers. The overriding of the Unreal Engine is possible due to the use of programming through object oriented techniques. The movement includes better weapon selections, better route selections, better dodging logic and tactical movements.

4.7. AI Framework for Group Coordination

To create group coordination for Follow, Gather and Hunt as seen in Figure 4-10, the FSM structures of the Unreal Engine are used to control each of the bots.

CHAPTER 4 – IMPLEMENTING TWILIGHT CITY

Figure 4-10: Follow, Gather and Hunt

For the Follow maneuver, the leader is defined an objective to the bots and the bots are commanded to move towards this objective. Thus, with the use of the existing FSM structures, a group of bots are made

to follow a leader when a voice command is issued through the speech control module implemented earlier. The Gather maneuver is simply a modification of the follow maneuver. This time, instead of a moving objective, the objective is fixed. Thus, the bots will gather at a user defined point when the Gather command is issued. To activate Hunt, a voice command is given to shift the bots into a search and destroy state in which they will look for terrorists and attack them.

4.8. Integrated Bots Control Interface

The integrated bots control, as shown in Figure 4-11, allows the users to give extremely specific orders to the bots using TeamSay or a special in-game menu. For example, we can instruct Soldier A to defend a spot for 300 seconds and then cover Soldier B with the machine gun. . With this integrated functionality, we can give commands with time limits or delay certain commands. There are a whole variety of commands and combinations that we can use. The integrated bots control can also declare spots during runtime. The user can mark a spot and issue a command involving that spot to the bots.

This feature is built using by extending for the existing Unreal Engine classes and altering the code via Unreal Script. Bots interactions are sent via messages between server and client-side implementations of Unreal Engine.

CHAPTER 4 – IMPLEMENTING TWILIGHT CITY

Figure 4-11: Integrated Bots Control

CHAPTER 5: QUALITATIVE APPROACH FOR PHYSICS SIMULATION

Movable objects such as chairs, boxes, bottles, etc. are quite common in real life. However, the computational cost will be high if the exact physics laws are used to model the behaviour of these objects. To reduce the computational cost of simulating moveable objects in Twilight City, a framework for qualitative behaviour simulation has been proposed and integrated into our simulation engine. Example results show that the proposed framework is successful in modelling the dynamic behaviour of various movable objects in Twilight City in terms of human players' perceptions and responsiveness of the system.

5.1. Qualitative Behaviour Simulation

In this section, we first describe the software architecture for qualitative behaviour simulation and then explain how to model the behaviour of a movable object using qualitative physics.

5.1.1. Software Architecture

To make the currently static objects movable, new object classes are extended from the Unreal engine's base classes. These new classes are added as Qualitative Physics (QP) Actors, and necessary models are attached to these Actors. A framework for implementing these QP Actors is proposed so that different movable objects could be simulated using a standard template. Figure 5-1 shows the software architecture of the framework, which consists of QP Actors and a QP Engine on top of the Unreal engine. The Unreal engine relies on event management system to support most of interactions among the objects in the game. To implement QP Actors, the event management

system of the Unreal engine thus needs to be investigated and extended.

The Unreal engine implements two types of events: *primitive events* built within the game engine and *programmed events* that can be scripted by developers for customization of the object interactions. To implement QP Actors, we need to override the native event management mechanism of the Unreal engine. The event handlers of the QP Actors are used to detect various actions such as users' actions and collisions, and to generate various events. These events are sent to the QP engine which will determine various effects on the QP Actor based on the logic rules. The logic rules may refer to various threshold values and kinematics libraries. The QP engine is built by adding another module into the Unreal Engine. This module process the inputs from the QP actors and returns appropriate to the QP actors. The QP engine works in tandem with the Kinematics Library and Threshold Values file. The Kinematics Library contains all the necessary actor behaviours required for their physics simulation. The behaviours are generated offline and repeatedly reused as deem fit by the QP engine. The Threshold Value file is editable by the user to allow calibration of the QP actor behaviours. For example, the user can increase the buoyancy of QP actor when in water or the user can adjust the weight of the QP actor. Thus, the QP Engine uses the threshold values to determine appropriate behaviours from the Kinematics Library and return relevant behavioural information to the QP actors.

Based on the information returned by the QP Engine, the QP actors will then exhibit suitable behaviours in the virtual environment.

Figure 5-1: Qualitative Physics Framework

5.1.2. Object Behavior

With qualitative physics, a continuous attribute (e.g., location) is quantified into a set of discrete values, and a set of desired behaviors are attached to these discrete values. The main idea of using qualitative physics is to reduce the quantitative precision of the behavioural descriptions but retain the main distinctions, thus the computational cost of simulating movable objects could be greatly reduced. The rationale of using qualitative physics to model movable objects in virtual environments is that humans in their daily lives are more sensitive to the qualitative (or causal) relations among various factors rather than the exact physics laws among these factors.

Each QP Actor will continuously update its attribute values. Once some of its attribute values reach a certain threshold, the QP Actor will send events to the QP Module. In turn, the QP Module identifies the situation and status of the object. Then, an analysis of the necessary

conditions and relationships affecting the situation is conducted with qualitative process methods [5] and appropriate results will be determined. The QP Module will send events to signal appropriate actions to the relevant QP Actors.

Figure 5-2: Different behaviors of a barrel

Figure 5-2 illustrates some qualitative behaviors of a barrel. Collision events are generated when the barrel (a QP object) comes into contact with other objects. It is common knowledge that objects may react differently when different points of contacts are involved. For example, when a barrel is hit at a low point on its body, it will slide across the floor. However, when the same barrel is hit at a high point on its body, it will tend to fall over. In Figure 5-3, contact regions A and B are marked out. When a force hits region B, an event containing B is sent to the QP Module. Assuming no other forces present, the QP Module will decide the barrel will slide across the floor. Similarly, if a force hits region A, the barrel will fall over. In this situation, the force must be large enough to overcome the inertia of the barrel for the barrel to move.

Figure 5-3: Effects of applying forces at different contact points

The QP actors use preconditions to identify the current situation that they are in and select the type of qualitative physics process to invoke. Each QP process has its own set of logic and rules. During the running of the QP process, the QP actors will send its QP values to the QP engine for computation. Subsequently, the QP engine sends the appropriate influences to the QP actors. Thus, using the influences to adjust the necessary parameters, the QP actors exhibit the appropriate behaviours. As a result, the QP actor maintains a standard flow of interactions with the QP Engine for all situations. With this framework, an object’s behavior can be easily described. For example, a heavy barrel will sink in water while a crate will float on water.

5.2. Example Results

In this section, we show some results of our qualitative simulation framework.

CHAPTER 5 – QUALITATIVE APPROACH FOR PHYSICS SIMULATION

(a) Falling Barrels

(b) Falling Boxes

Figure 5-4: Uses of Qualitative Physics in Twilight City

Physical simulations within the *Twilight City* include collisions, object kinematics, water flows, frictional forces, torque. We placed a number of boxes, crates, barrels and balls within the *Twilight City*. Using the concept of Object-Oriented Programming, the various objects are coded to hold individual characteristics and are affected by collisions, friction, gravity and buoyancy. Collision events are received when the QP objects come into contact with other objects. In figure 5-5.a, a logical formula of the qualitative process of collisions is presented. The event handlers within our qualitative physics engine are invoked by each of the objects during a collision. After being invoked, the event handlers will check the entities involved in the collisions. The QP actors use the preconditions to determine the current situations. From each situation, they will determine if the preconditions are fulfilled. The preconditions for this event is there must be contact between the objects involved and at least one of the objects must be exerting a force. If one of the preconditions is not fulfilled, the event handlers will stop operations immediately. In our framework, when the preconditions are met, the qualitative physics engine will start the computation as required by the QP logic of the QP actors involved. For the situation of object collisions, the forces exerted by the objects will be compared and the resulting motion will be generated. The

qualitative physics engine will analyze the contact regions of the objects as well. After its computation, the QP engine feedback the appropriate position and rotation parameters. The feedback values are also known as the influences to the object motions. In collisions, the influences affect the position and rotation of the object. The effects of the influences to the object behaviours are observed in figure 5-5.b and 5-5.c

Situation: Object Collision
Entities Involved: Object A & Object B
Preconditions: Contact(Object A, Object B), (?Force(Object A) ? Force(Object B))
QP Logic: ?(Force(Object A)> Force(Object B)) ?Contact_Regions(Object A) ?Contact_Regions(Object B)
Influences: Object Position Parameters, Object Rotation Parameters

(a) Logical Formula

(b) Before Collision: Tips of boxes are the contact points

(c) After Collision: The boxes ricochet away while spinning

Figure 5-5: Qualitative Process of Collisions

CHAPTER 5 – QUALITATIVE APPROACH FOR PHYSICS SIMULATION

Apart from simulation of the movable objects, we explored the use of qualitative process theory on water simulation. The movable objects and water are integrated within *Twilight City*. In figure 5-6.a, a logical formula of the qualitative process of object behaviours in water is presented. In the situation of objects within water, the qualitative physics objects invokes the event handlers, which reside within the qualitative physics engine. After being invoked, the event handlers will check the entities involved in the collisions. The precondition for this event is that there must be contact between the objects and water. When the preconditions are met, the QP logic within the QP actors will compute the influences through the use of the QP engines. The object buoyancy, water densities and water motion affect the behaviours of the QP objects. Using the object buoyancy, water density and water movement, the resulting motion will be generated. The QP engine feedback the influences to the object motions. In this situation, the influences refer to the position and rotation parameters of the object. Upon receiving the influences, the QP objects will exhibit the corresponding behaviours. Thus, we can observe the sinking and floating of different objects. Heavy barrels will sink while light boxes will rise as observed in figure 5-6.b and 5-6.c.

Situation: Objects in Water
Entities Involved: Objects & Water
Preconditions: ?Contact(Water, Object)
QP Logic: ?Density(Object, Water), ?Water_Flow(Water)
Influences: Object Position Parameters, Object Rotation Parameters

(a) Logical Formula

(b) Objects submerged in water

(c) Different object behaviours

Figure 5-6: Object behaviour in water

When objects collisions happen in water, we can see two QP situations occurring at the same time. Thus, the QP actors will run the two QP logic processes concurrently and the QP engine generates a larger set of influences consisting of both collision and water influences. With this combined set of influences, the QP objects will exhibit behaviours expected from object collisions under water as observed in Figure 5-6.

5.3. Practical Applications

To further emphasize the benefits of qualitative physics, we will present some applications of such a technology towards MOUT

simulation. In this section, two MOUT scenarios that are enhanced by qualitative physics will be discussed.

5.3.1. Collapsible Structures

Presently, most virtual urban environments are constructed by fitting 3D building blocks onto a map. These buildings have realistic textures and allow players to navigate within them. However, from our past human factor tests, it is observed that the existing buildings are not collapsible and cannot be damaged in any way. It is unrealistic that the buildings remain invulnerable in all situations. Apart from affecting the computing performance, it is also a challenge to model the secondary collisions between different various movable objects built from existing quantitative physics techniques. Therefore, we can utilize qualitative physics to build up a collapsible structure that will perform realistic behaviours when faced with air strikes or bombings as seen in Figure 5-7. Such collapsible structures are built from aggregating small movable objects. It is observed that such behaviours can be implemented with qualitative techniques without significant degradation to the performance.

CHAPTER 5 – QUALITATIVE APPROACH FOR PHYSICS SIMULATION

Figure 5-7: Collapse of structure during an air strike (numbered in sequence)

From this example, we can see that another advantage of qualitative physics is its efficiency in handling non-monotonic behaviours such as rocking back and forth. Numerical computation can become highly complex when such a huge number of objects experience almost

simultaneous collisions within a short time interval. With qualitative physics, we can reduce the complexity with selective activation of processes and simulation of object behaviours to an acceptable level as opposed to a precise level. Qualitative physics can model collapsible structures by composing a set of smaller movable objects. Thus, large-scale implementation of the qualitative physics will improve the existing fidelity levels without compromising performances.

5.3.2. Road Blocks

Through our experiences, we found it is common to have military checkpoints in MOUT simulations. Current checkpoints in virtual environment tend to be immovable or indestructible. Such invulnerable checkpoints are unrealistic and ineffective because the users will not be able to see the actual effect of attacks on the checkpoints. We introduce qualitative behaviours to some barricades and tires as shown in Figure 5-8. To achieve such behaviours with exact physics will be incur high computational overheads due to the complexities involving the interactions of the objects during collisions. Qualitative physics is able to overcome such limitations.

CHAPTER 5 – QUALITATIVE APPROACH FOR PHYSICS SIMULATION

(a) Barricades Set up

(b) Tires scattered

Figure 5-8: Barricades and Tires

These movable objects are used to form the defence layer of the checkpoints from onrushing car bombs. The barricades are useful as they allow researchers to experiment and visualize the set up of checkpoints. We did some experiments on the effectiveness of various barricades layouts at stopping onrushing vehicles as shown in Figure 5-9. Results show that single layer barricades do not provide adequate resistance to onrushing vehicles as compared to multiple layered barricades. It is significantly harder for vehicles to barge through when a few barricades are placed behind each other. Interestingly it can also be seen that barricades are stronger when 2 barricades are placed in a T-shape layout. The most effective layout, so far, is to place a T-shape layout with a tire behind it. The vehicles get stuck when they try to run over the barricades.

(a) Multiple Layered Barricade Layout

CHAPTER 5 – QUALITATIVE APPROACH FOR PHYSICS SIMULATION

(b) Stopping the Test Vehicle with the T-shape Layout

(c) Stopping the Test Vehicle with the T-shape Layout

Figure 5-9: Barricades Layout

CHAPTER 6: ARCHITECTURE FOR RAPIDLY RE-CONFIGURABLE SIMULATIONS

Apart from achieving low cost and high fidelity simulations, MOUT simulation systems are also expected to reconfigure rapidly with the frequent shifts in operational needs. We found that existing software architectures may not be optimized for re-configuration with a game engine based system approach. To this end, we describe our work on building agile and reconfigurable systems to construct game engine based MOUT virtual environments rapidly. This work leads us to propose the Architecture for Rapid Configuration (ARC) to support model interoperability and reusability for MOUT simulations. Along with ARC, we will explain how two other components, namely Game-based Object Standard Interfaces (GOSI) specifications and the ARC-based Objects Repository (ARCOR), work in tandem with the ARC. In our previous work, we have constructed *Twilight City*, a virtual training environment for MOUT. Our experiences show that the proposed architecture has transformed *Twilight City* into an agile simulation system that can build virtual environments at the required urgency.

6.1. Motivation

Our experience in constructing *Twilight City* shows that it is very time consuming to build the simulation environment from scratch due the large number of object models involved. In the meantime, we also noticed that although the behaviors of various objects seem different, they share a common structure in terms of how to respond to various events in the simulation, thus can be modeled using a generic framework. In particular, to increase realism of *Twilight City*, there is a need to increase the number of movable objects yet keeping the computational costs low. Thus, we used qualitative physics for modelling movable objects in *Twilight City*. In this chapter, we will

show how the proposed *ARC* helps to model the movable objects using qualitative physics. On the other hand, to facilitate the building of interoperable, reusable and composable models, *ARCOR*, a repository of models will be accumulated to aid rapid model development. Together with the *ARC* framework and *GOSI* specifications, the *ARCOR* will help to create a re-configurable test bed that offers flexibility and rapid model development at a low cost.

6.2. Overview

At the conceptual level, the *ARC* is designed to facilitate task abstraction and modularity in design. The *ARC* provides a framework to allow the various objects to operate in tandem. The *GOSI* specifications control the interaction between modules and acts as the linkages that hold the framework together. Finally, the *ARCOR* holds a rich repository of objects that can be activated to compose *ARC-based* simulation entities rapidly.

We adopt an object-oriented modelling approach to creating reusable and composable objects in MOUT simulations. With this approach, complex simulation models are broken down into some basic building blocks. For the purpose of this project, anything that exists individually within the virtual environment is known as an entity. Examples of entities are human, bots, movable objects, vehicles, etc. Entities contain components known as objects. Objects are responsible for the behaviors of the entities. In our test bed, objects are the superset from which actors and models are extended from via separate branches from the abstract base class, *Object*. Actors only store the current state of the simulated entity. Interaction models are the data exchange layers between logic models, actors and the virtual environment. Logic models should only exchange data via interaction models. With the relevant data, they will then perform reasoning and decision making. Thus, an entity within *Twilight City* will contain an

actor attached with logic models and interaction models working in tandem to provide realistic behaviors.

The class hierarchy in Figure 6-1 shows how the object models are extended from the base class. Weapons and human actors are extended from the base actor class. Models such as Path Finding and brain are also extended from the base model classes. Characters such as terrorists and soldiers are sub classes of the human actor class. Similarly, different weapons such as bombs for the terrorists and rifles for the soldiers can be produced from extending the weapon class. After the required objects are formed, the resulting entity is formed putting the objects together. For example, the soldier entity is built up by the rifle, human motion and soldier brain objects. Likewise, the terrorist entity is an aggregation of the bomb, human motion and terrorist brain objects.

Figure 6-1: Class Hierarchy of ARC based objects

6.2.1. Architecture for Rapid Configuration

Actor, logic models and interaction models are the building blocks of the ARC. ARC clearly defines the relationship between the actors, logic models and interaction models. *Entities* hold their objects in the ARC framework. The explicit definition of task allocation allows ease in model substitution and ensures modularity in the design. Being modular, it is easy to modify specific components of an ARC-based object without affecting the rest of the components.

Being able to access the game engine APIs, actors provide an entity with the capabilities to move around, interact with other actors, affect the environment, and do other useful game-related things. The high-level artificial intelligence processes such as physics reasoning and behaviors are contained within model classes. Models are attached to actor classes in a plug-and-play manner to provide the actors with the behaviors or reasoning techniques. The capabilities of each model should be distinct to ensure modularity. With modularity, we can compose the basic model classes into complex behavior classes. Models are further defined into the interaction models and the logic models. In figure 6-2, we illustrate ARC with a simple diagram.

Figure 6-2: Architecture for Rapid Configuration

ARC allows coordination between the logic and interaction models. The interaction models retrieve information from the actors and the environment for the logic models to perform their reasoning

techniques and decision making. The logic models control the actor and affect the simulation environment via the interaction models.

Being a modular framework, ARC allows the easy migration and swapping of models within an entity while minimizing the propagation effect sustained by the remaining models. The ease in model swapping provided by ARC can provide flexibility to the simulation system to switch their entity behaviors rapidly. With a common architecture, it will also be more convenient for the construction of interoperable models for sharing and reuse.

6.2.2. Game-based Objects Standard Interface

To be a rapidly re-configurable system, the frequent model swapping is expected. If there is no standardization of model data exchange methods, huge effort of time and effort will be incurred each time a model is migrated. Therefore, we propose the *Game-based Objects Standard Interfaces (GOSI)*, which contains standard interfaces specifying data exchange methods. The *GOSI* specifications will specify the interfaces that objects can follow. The objects will adopt interfaces according to their specific functions. As an example, Figure 6-3 shows the *GOSI* interfaces for the *actors*, *interaction* and *logic models* of movable boxes.

Figure 6-3: Game-Based Object Standard Interface compliant classes

AngularEffect and DisplacementEffect can be broken down into separate logic models. The two components perform distinct functions and can be composed to form the BoxPhysics model. When the BoxPhysics model is broken down into AngularEffect and DisplacementEffect logic models, the two logic models will again follow a set of GOSI interfaces.

Defining *GOSI* specifications will be our ongoing efforts to standardize data exchange. Presently, *GOSI* is largely working within the MOUT simulation domain. *GOSI* relies on the fact that general MOUT operational concepts are similar across projects [16]. For example, most MOUT simulation projects will need human behaviors, weapons, sensors and missions. By standardizing *object* interfaces according to their domain, scope and function, we can control information exchanges between models and achieve interoperability, reusability and composability. To share models across different game engines, we

can use wrappers to convert the game-specific code to suit another game engine.

6.2.3. ARC-Based Objects Repository

With *ARC* providing the architecture to fit the *entities* and *GOSI* to provide the linkages between the models, we need to set up a repository to store all the *ARC-Based Objects* for future re-use. From figure 6-1, it is seen that the *ARC-Based objects* are held within a class hierarchy. To facilitate the rapid retrieval of objects within *ARCOR*, we maintain the class hierarchy relationship of the objects instead of storing the objects with respect to the project that they are contributed from. The capabilities and limitations of the objects in the repository should be documented to provide a better understanding of the objects. To aid in the validation and verification process, the projects that the objects had been used for should be recorded as well. The more frequently used the models are, the greater is our confidence level with the models.

The recursive development flow with an *ARC-based* approach for MOUT simulations is shown in Figure 6-4. The simulation is made up of many *entities*. Each *entity* exhibits distinct behaviors as controlled by its *objects*. We compose *entities* by assembling the *objects* in an *ARC* framework. From Figure 6-4, we can see that after identifying the purpose of an *object*, the conceptual model for the *object* will be defined. Then the *ARCOR* is scanned for an *object* corresponding to the conceptual model. If an appropriate *object* is found, the *object* will be used to assemble the framework of the *ARC-based* entity. Otherwise, a new *object* needs to be built. This new *object* shall adopt an appropriate interface based on the *GOSI* specifications. Subsequently, we shall attempt to further break down the object into smaller components which can be composed together to form the original object. As shown by the dotted line in Figure 6-4, we will then do a recursive search for the composable components. The development

process for the *ARC-based objects* will be repeated recursively until the *entity* obtains all required objects. If the *object* cannot be further broken down, the *object* shall be developed from scratch. Thus, all *objects* are constructed either by composing existing *objects* or from scratch. All newly built *objects* will be added into the *ARCOR* for future re-use.

Figure 6-4: Development process for ARC-based objects

6.3. Example results

In this section, we describe how the proposed framework helps to extend our previous work on *Twilight City*. In *Twilight City*, we adopted an approach of Qualitative Physics (QP) to model the behaviours of movable objects. This approach will help to reduce the computational

cost of simulating the behaviours of movable objects and increase the fidelity level. We implemented a QP engine to simulate the behaviour of various movable objects. Figure 6-5 shows how the QP simulation on movable objects is achieved. This part of work had been presented in the 39th Annual Simulation Symposium [5].

Figure 6-5: Qualitative Physics Framework

Using *ARC*, we streamlined the development process of *QP* entities within *Twilight City*. We rebuilt the *QP* components in accordance to the *ARC* framework. The interactions between components captured within the dotted circle in Figure 6-5 were changed due to the influence of *ARC*. Instead of both the *event handlers* and *object behavior* communicating to the *Qualitative Physics Engine* separately, the *object behavior* can only communicate with the *QP engine* via the *event handlers*. The *Event Handlers* in Figure 6-6 are built using the *interaction models* from the previous project within the *ARCOR*. These interaction models are constructed based on the *GOSI* template to ensure composability and interoperability. Similarly, the *Object*

Behavior module is built up by composing the logic models stored in within the *ARCOR*. *Qualitative Physics Actors* are extended from the existing actors as well.

Figure 6-6: ARC- based Qualitative Physics

With the *ARC* framework and *ARCOR*, the development time for creating various movable objects in *Twilight City* was shortened from a projected 8 months to the actual 2 months. Figure 6-7 shows some examples of movable objects that we implemented in *Twilight City*.

(a) Movable Computers

(b) Tumbling Tires

(c) Falling chairs

Figure 6-7: ARC- based QP Movable Objects

There are many other benefits of using the *ARC* framework, e.g., the rapid generation of terrorist behaviors in *Twilight City*. The terrorist behaviors are extended from the generic human *entity* and are combined with existing models such as the shooting actions of the soldier models. When there is a need to change the behavioral patterns of terrorists, we can conveniently adapt to this change by extending the existing models in the *ARCOR*. At the current stage of our project, we have implemented more than 50 *ARC-based* models. We expect that the current model repository will greatly shorten the development time of various MOUT simulation set-ups and fulfil the goal of building a rapidly configurable simulation system.

CHAPTER 7: EXPERIMENTATION RESULTS

This chapter puts the capabilities of Twilight City to test with various methods and scenarios. We use subjective perception tests, automated testing and performance testing in our efforts to test and benchmark the effectiveness of Twilight City.

7.1. Interactive Voice Controls

It is important to test our voice commands that are incorporated into Twilight City. With a microphone, simple verbal commands are given and the performance of the module is evaluated. The testing should be done in a quiet room so that background noises are minimized.

7.1.1. Analysis and Comments

The voice control module works well within Twilight City. The speech control of many bots movement such as shooting, jumping or dancing is tested. However, two issues were identified during the testing phase.

Firstly, some of the voice commands are too complicated and it is difficult for the system to recognize them. The users may need to repeat the commands a couple of times. Using a more powerful microphone and simple commands with shorter syllabus later solved this problem. However, the commands cannot be too simple as well. Being too simple, the unwanted actions may be triggered by background noises that are falsely recognized as voice commands. Therefore, besides the use of a good microphone, the selection of the verbal commands used is important too. After rigorous testing with different command patterns, it is observed that the voice control module performs best with commands of two or three syllabuses.

The second issue is latency in performance of the module. There is a delay of around 3 seconds between the commands being issued and actions being performed. This turnaround time problem is investigated and improvements were made. The algorithm of the module includes a feedback module that displays the buttons being pressed during simulation. The structure of this module had too many unnecessary If-Else statements. After the streamlining of the code, the performance of the voice control module showed significant improvement. The performance of the voice control module is tested on different hardware specifications and it is observed that the module shows better response times with faster processors. Apart from that, the quality of microphone is important in the speech recognition time of the system as well. Results had also shown that there is a positive correlation in the amount of voice training and the performance of the voice control module

7.2. Evaluation of the Modifications made

The shattering glass effect is tested to demonstrate the hearing sense of the bots as shown in Figure 7-1. If the bots-hearing radius is within the noise radius of the shattering glass then the bots will turn to the noise and scan for activity in the area. This is tested in a room that consists of a soldier with its back facing the human player. If the human player shatters the glass, the bots will turn around and fire at the human player.

Figure 7-1: Shattered Glass in Twilight City

The raining effect shows a problem during testing. The raindrops are not the results of a particle system. Instead, the rainfall effect is a visualization trick of having raindrops in front of the bots so that they will have a field of view with a rainy weather. The raining effect only affects the field of view of the bots but not the entire virtual environment. The problem arises when the raining effect continues while the bots are actually under shelter. To solve this problem, a volume controlled by a script is used to disable the rainfall effect when the bots is inside the volume.

7.3. Benchmarking

During the testing phase, benchmarking analysis was carried out to evaluate the performance of Twilight City against the performance of the game levels created commercially by the Unreal Engine 2.0 developers. The benchmarking process is used as a quality assurance tool to ensure that this project have fully utilized the capabilities of the

Unreal Engine to build a virtual environment which will not lose out to those created by the game developers. Four commercial maps were short-listed. They are RobotFactory, UrbanStrike, MotherShip and Fallen City. The maps were chosen as they are of comparable size with Twilight City. The maps were compared in terms of image processing, rendering performance, rendering and speed of game play. The tool used for this analysis is UMark. UMark is the benchmarking software that allows gamers and hardware reviewers to easily configure and run benchmarks on Unreal Tournament 2004. UMark is not built upon its own benchmarking engine, it uses UT200x's benchmarking engine to run the tests. Running benchmarks with UMark is just like running UT2003.exe or UT2004.exe with benchmarking options passed to it. The scores were obtained by taking the result of a function involving the average frame rates and its variance as calculated from the difference between the minimum and maximum frame rates. The higher performance scores indicate a map running with higher frame rate at better consistency. We made use of the standard test values that comes with UMark. While UT2004 didn't include standard .ini files for benchmarking, UMark has its own files for UT2004, in which will hopefully become the de facto standard for UT2004 benchmarking. For example, when we choose "High Performance" or "High Image Quality", we are telling UMark to use the UMark standard .ini files. When "High Performance" is selected, it uses "MinDetail.ini" and "MinDetailUser.ini" configuration files to run the tests with. Respectively, when "High Image Quality" is selected, it uses "MaxDetail.ini" and "MaxDetailUser.ini". The performance scores of the different maps were processed and compared with Twilight City and displayed on Table 7-1. The scores were averages of benchmarking values from high performance, high image quality, normal settings and 32 bots.

Table 7-1: Benchmark Analysis

As observed from the benchmark statistics, *Twilight City* performed better than maps of similar sizes made commercially. The results not only prove that the potential of the Unreal Engine was maximized. It also provides a quality assurance on performance standards of *Twilight City*.

7.4. Mean Opinion Score Tests

To further evaluate the performance of *Twilight City*, we have installed it on some PCs (DELL with Pentium IV 2.4GHz CPU) in the Parallel and Distributed Computing Center (PDCC) at Nanyang Technological University and the Modeling & Simulation Department of the Defense Science and Technology Agency of Singapore. 26 subjects (aged between 20 to 40 years old) were asked to play with *Twilight City* for about half an hour (see Figure 7-2). Then the subjects’ opinions on various aspects of *Twilight City* were collected. As the system is meant mainly for military purpose, all the subjects have more than 2 years of military training experiences. The military experience of the human subjects will help to validate the accuracy of the behaviours simulation in *Twilight City*. Besides that, 16 of the subjects also had experience with military simulation systems before.

Figure 7-2: Mean Opinion Score Tests of Twilight City

It should be noted that the emphasis of these tests is not on the behavior of the AI bots that we have developed. Though the bots in *Twilight City* have demonstrated a certain level of tactical intelligence, we feel that the behaviors of the bots are still very limited and thus the bots are still not intelligent enough to collaborate or fight against real human players in more complex scenarios. Thus, the emphasis of these tests is on a subject's perception of the visual effects of the virtual environment and various artifacts that we have introduced. The subjects were asked to give their opinions on the following questions:

1. How do you feel the responsiveness of *Twilight City*?
2. Is *Twilight City* a good urban warfare simulation tool?
3. How do you feel the voice module (accuracy and responsiveness)?
4. How do you feel the behavior (realism and responsiveness) of the movable objects?

The subjects answer these questions by giving a score (5-Excellent, 4-Good, 3-Fair, 2-Poor, 1-Bad) to the performance of *Twilight City* on

each of these aspects. Figure 7-3 summarizes the results of these tests.

Figure 7-3: Mean Opinion Score Tests Results

The subjects are satisfied with the responsiveness of *Twilight City* and the behavior of the movable objects that we have introduced. They also feel that these movable objects greatly enhance the realism of *Twilight City* as compared to other FPS games they have played. However, subjects suggested that movable objects should be modelled to show damages after collisions. For example, a part of the box should be chipped off upon colliding with the wall. Though some subjects suggest that the voice module has noticeable delays and failed to produce correct results in some cases, the results show that the voice module is generally acceptable by most subjects. In general, the subjects feel that *Twilight City* is a good prototype system for MOUT simulations. The movable objects, various artifacts and the voice module are important to MOUT simulations. However, more voice commands need to be added, and more training of the voice module are also needed.

7.5. Frame Rate Analysis

For the purpose of quantitative analysis of the performances of qualitative physics, we can monitor the frame rate of the systems during runtime. We can extract this frame rate data from the Unreal Engine for the purpose of our tests. Although the frame rate can be

affected by a number of display factors, physics simulation impairs frame rates badly.

For comparison purposes, we implemented three set-ups for a particular scenario within *Twilight City*. In our tests, all other factors are kept constant and the only variation is the physics simulation methods. The first scenario is set up without any physics simulation to act as the control. The second scenario is built with the physics engine residing in Unreal Engine. The physics engine, *Karma*, drives the physics simulation through quantitative methods. The third set up is built from the qualitative methods discussed in this chapter.

As there are limited amount of physics simulation in the existing unreal environments, we need to construct our own test environment. *Twilight City* is utilized as the test bed for the purpose of this experiment. We introduced 132 movable objects within each of the environments using various physics simulation methods. Actions such as calling for air strike on collapsible structures and running down barricades are performed. The experiment was done on a computer with Intel T2500@2.GHz processor and 2GHz RAM. With each set-up, we did five runs and computed the average frame rate. The results of the frame rate analysis are shown in Figure 7-4.

When average frame rates fall below 30, lag is observed during the rendering. The default minimum desired frame rate in Unreal Engine is set at 35 by Epic Games. Thus, frame rates above 35 can be considered as acceptable.

Figure 7-4: Frame Rate Analysis Results

The results of the experiment show that qualitative physics incurs significantly lesser computational overheads as compared to quantitative physics. In fact, it is impractical to perform quantitative physics simulation of a large number of objects, as the load will drive the frame rates below the acceptable level of 35. It should be noted that quantitative simulations are more accurate than qualitative simulations. We should continue to utilize quantitative techniques when dealing with a small number of movable objects. The benefit of qualitative physics towards performance becomes more evident as the number of movable object increases. Therefore, it is particularly useful to apply qualitative techniques when the amount of simulated objects is large.

CHAPTER 8: CONCLUSIONS AND FUTURE WORK

This chapter provides a summary of the work done and the achievements of this project. The problems encountered during the implementation of Twilight City are discussed as well.

8.1. Summary

Chapter 3 described the design aspect of the Twilight City. Descriptions covered were on the overall architecture and the individual modules of Twilight City. This chapter described on the design of the virtual environment as a platform that allows simulation analysis to occur.

Chapter 4 touched on the implementation aspect of Twilight City. Directions and the requirements to setup the virtual environment framework were discussed and carried out. The main emphasis of the chapter is the implementation of simulation techniques that were used to develop and implement Twilight City as an efficient simulation tool. Various simulation techniques and game engine modifications were implemented to enhance the effectiveness of using a virtual environment used for MOUT simulation purposes.

Chapter 5 discussed a major contribution of our project towards MOUT simulations. Qualitative physics in Twilight City had acted as a successful proof of concept demonstrator of integrating the qualitative approach for future simulations. The reasoning techniques and the implementation were explained in this chapter.

In Chapter 6 we proposed an ARC framework to promote interoperability, reusability and composability among various models across different MOUT simulations. With ARC, we would be able to

rapidly re-configure our simulation testbed to meet the frequently changing requirements of the military simulation community.

Chapter 7 presented our experimentation and benchmarking tests on Twilight City. Twilight City performed well with the MOUT operations objectives implemented. Such tests increased the confidence level on the effectiveness of Twilight City and gave us a better understanding of its limitations as well.

In this project, we proposed the usage of a game engine based interactive virtual environment for Military Operations on Urbanized Terrain (MOUT) simulation. Our work revolved around building an interactive virtual MOUT test bed with voice control capabilities, architecture design, group coordination and game modifications. The Twilight City was produced as a product of this project. It provided a virtual environment for MOUT simulation, which came with special effects, custom animations, mutators and voice control modules. The architecture for rapid re-configuration was implemented within the Twilight City. Feedback from Artificial Intelligence (AI) researchers showed that the proposed interactive virtual environment provide better support in terms of MOUT simulation compared with the existing techniques. We incorporated qualitative physics for movable objects in Twilight City. This technique proved to be successful in reducing the computational overheads of physics simulations at acceptable realism levels. We proposed an architecture for rapidly re-configurable simulations to promote interoperability, reusability and composability among various models across different MOUT simulations. A model repository was developed which defines various commonly used object classes for MOUT simulations. With this repository, new models could be easily built by reusing or extending the existing models, which help to reduce the cost and time to build a new simulation environment.

The interactive virtual environments could support the definitions and authoring of a wide variety of behaviours. Along with the behaviours, virtual environments could be used for military training where the soldiers could be trained to be familiar with certain scenarios and location without having to be there physically. Furthermore, virtual environments allow easy and cheap storage of 3D representations of various locations. Success stories of simulation based on game engines includes [1], [7-10]. Due to the portability of the software, the simulation and training could be done on different systems on different parts of the world through network connections. This allowed teams to bridge their spatial issues. The virtual environments developed from Unreal Engine during the course of the project are of excellent quality, further reinforcing the relevance of Unreal Engine to future researchers. This approach had potential contributions in MOUT simulation, training. The current implementation provided a good framework for future work.

The interactive MOUT virtual environment had successfully shown its worth and capabilities, which allow researchers to easily build a MOUT infrastructure for use. As we are now in the process of integrating our work into DSTA military simulation systems, we can safely conclude that the results have met the purpose of the project.

8.2. Publications

We produced some papers to share our results. Through these publications, we know that our work will be able to benefit the community as a whole.

8.2.1. Journals

- Twilight City – A virtual environment for MOUT, Suiping Zhou, Shang-Ping Ting, Zhuoqian Shen, Linbo Luo (Accepted by International Journal of Computer and Applications)

- A Qualitative Approach to Behaviour Simulation in Virtual Environments, Shang-Ping Ting, Suiping Zhou (Under review by Simulation - Transactions of the Society for Computer Simulation International)
- Quartz: An Autonomous Navigation System for MOUT, Shang-Ping Ting, Suiping Zhou, (Invited for Computer Animation and Virtual Worlds Journal, impact factor: 1.091)

8.2.2. Conferences

- Qualitative Physics for MOUT, Suiping Zhou, Shang-Ping Ting, 39th Annual Simulation Symposium, April 2 - 6, 2006, Huntsville, AL, USA
- A Generic Model Framework for MOUT Simulations, Suiping Zhou, Shang-Ping Ting, 2006 International Conference on CYBERWORLDS, Nov 28-30, 2006, Lausanne, Switzerland
- An Architecture for Rapidly Re-configurable MOUT Simulation, Shang-Ping Ting, Suiping Zhou, 40th Annual Simulation Symposium, March 26 - 28, 2007, Norfolk, VA
- Quartz: An Autonomous Navigation System for MOUT, Shang-Ping Ting, Suiping Zhou, 20th International Conference on Computer Animation and Social Agents (CASA2007), 11-13 Jun, 2007, Hasselt, Belgium. (Ranked among 35 Best Papers, Invited for Computer Animation and Virtual Worlds Journal)

8.3. Future Work

Although the test-bed has fulfilled the basic requirements for researcher to work on multi-agent research on MOUT, there is still some room for improvement. The following sections propose a number

of recommendations that can be implemented to further improve the capabilities of the test bed and further propel the project into new heights, especially for NTU.

8.3.1. Interactive Virtual Environment for the Visually Impaired

An interactive virtual environment for the blind can be constructed to incorporate the speech function with the virtual environments. This map caters to the need of the visually impaired. A blind person can navigate through the map through speech and the computer will update him of his location upon request. The blind user can use the maps to understand the layout of a place without having to physically going to the place. Presently, a blind person can only understand the layout of a building by walking through the place or by feeling models of the place. However, models are not easily transferred and physically going to places merely to understand its layout is a tiring task. With the blind maps set up, the disabled can access the 3D environments anytime and anywhere they want.

8.3.2. Software Architecture

Research on software architecture of virtual environments should be considered in future. The current software architecture used is a simple way of implementing game objects. However, there is actually more to it when more and more objects are introduced. This would clog up the whole environment and slow down the process. Good software architecture is required to deal with this issue, as it should act like the brain of the program. Having good software architecture will improve the system's performance too, as it will be generic to allow modules to be connected to it without changing anything to it. Furthermore, software architecture also refers to implementations of projects that require a mixture of programming platforms. In such cases, we will need to set up the most efficient software structure in terms of processing and connection speeds.

For example, when a human needs to pick up an item, he needs to intuitively think of moving his arms down to the object’s location and then open his hands to grip the object. Picking up the object and moving his arm back follows this. All this action requires coordination between the arm, hand and the brain.

Having good software architecture will improve the system’s performance too, as it will be generic to allow modules to be connected to it without changing anything to it. Modules are like motor system, knowledge database, navigation system and etc. A simple AI architecture can be generated as shown in Figure 8-1. This architecture may not meet some specifications but is a basic idea to build on.

Figure 8-1: Simple AI Architecture

8.3.3. Improved Navigation System

There are various techniques that can be used in building a navigation system. To have an efficient and effective navigation system, an agent should find its path from point A to point B with the shortest time. It is always fastest to look up a path from a pre-calculated table. This would allow path finding of 10 to 200 times faster than the well-known A-Star algorithm [36]. With this method, the CPU usage for computing

paths is reduced tremendously. There are some disadvantages to path look-up tables too. Memory usage will have to be increased. However, many optimization techniques can be implemented to minimize this.

To achieve this improvement, the paths that are pre-computed are stored in a matrix [37]. This is also known as a path look-up matrix. To generate the look-up tables, the area-based path look-up algorithm is used. With this approach, the path look-up is done in two levels. This can be seen in Figure 8-2.

Figure 8-2: Terrain with 8 waypoints connected between 2 areas, but only 4 waypoints are sufficient for all inter-area connections

At the higher level, the terrain is seen as a set of portals, connected by clusters of waypoints or areas. To get from point A to B at that level, the portals for those points are determined. Then, the shortest path is determined between the nearby portals using a portal path look-up table. At the lower level, the portal path is translated into waypoints path. For each of these paths, the path in each area to the portal is retrieved. The portal is the waypoint that bridges the two paths forming a single path. This technique would allow small look-up tables,

one for the portal paths, the other for the within area paths, rather than one large look-up table. This would save a lot of memory too.

For a terrain of 2000 nodes (N), partitioned into 133 areas (P) connected by 90 portals (M), the memory consumption is about 463KB! This figure can be calculated from the following formula [36].

N/M (areas) $\times M^2$ (matrix size) $\times$ (per-area travel info) $+ P^2$ (connections) $\times$ inter-area travel info where per-area travel info and inter-area travel info are derived from some game content.

With this navigation system in place, we have an extremely fast path finding with relatively low memory cost. For this virtual environment, it is an important aspect to conserve memory and CPU usage. Furthermore, this technique is fairly easy to implement and is an enhancement to the current grid system, therefore it is recommended for future research work.

8.3.4. Test beds with Specific Behaviours

Through the use of modification of the game code, future research should involve creating test beds on Twilight City with specific behaviours such as flocking. The hostage rescue operation that was implemented is a good example of a test bed for simulation of rescue missions. More test beds such as those for airborne operations or ambush operations can be produced. These proposed test beds could be built from the existing development. As an interactive environment is already in place, the future developers only need to have an understanding of Unreal Script to create slight modifications to the currently available framework to produce the necessary mission frameworks. The test beds can be used in multi-agent research in other institutions especially in games AI. With the implementation of the test beds, the researcher can use the simulation results to perform analysis and evaluation of the operations involved. In the long term,

with a large package of scenarios, NTU can produce a truly complete simulation tool for all types of MOUT operations.

Figure 8-3: MOUT Airborne Operations as Future Developments

8.3.5. Steering Behaviour for Autonomous Agents

Efforts can be spent to investigate the steering behaviours of autonomous agents. However, problems evolved when more agents are required to follow the leader. The agents would bump into each other or the leader so often that the group movement or the objective of following the leader is lost.

Flocking is a computer model for the coordinated motion of groups (or flocks) of entities. Flocking represents typical group movement, as seen in bird flocks and fish schools, as combinations of simple steering behaviours for individual agents based on the position and velocities of nearby flock mates. Although individual flocking behaviours (sometimes called rules) are quite simple, they combine together to

give flocks very interesting overall behaviours, which would be extremely complicated to program. However, flocking behaviour can be implemented by utilizing 3 general rules described by Reynolds [41]:

- Separation: steer to avoid crowding local flock mates.
- Alignment: steer towards the average heading of local flock mates.
- Cohesion: steer to move toward the average position of local flock mates.

Figure 8-4: Flocking Behaviours

In Figure 8-4, which was extracted from the website of Reynolds [42], it shows the leader (grey), being followed by autonomous agents (green). The agents would not bump into each other and also move in a coordinated manner. This steering behaviour has already been used in multi-agent research in other institutions.

With the steering behaviour, the researcher can use this information to integrate with AI architecture as shown in Figure 8-5, which is extracted from [43]:

Figure 8-5: Hierarchy of Motion Behaviours in AI Architecture

It is thus recommended that future research to be done in this area.

8.3.6. 3D Virtual Worlds

With the increasing advancement in computer technology, there will be a strong demand for virtual environments of real locations for training, design or navigation purposes. Fantasy virtual worlds are in demand as well. Therefore, there is obvious tangible and economical value in virtual environments. Virtual environments can be designed to aid other areas of research or for exchange or monetary benefit. Thus, there is motivation to start developing virtual worlds. The framework and techniques learnt during this project can be used as a platform for future students to work on.

8.3.7. Conducting a Game AI Course for NTU

In view of the courses conducted in NTU, there is a lack of a course that allows creativity to be developed, especially in games. Early this year, the game scene in Singapore has been hyped up dramatically due to the support of the government. This is further justified when one of the Ministers promoted the World Cyber Games competition held early September 2004. Game AI is one of the courses that allow creativity and innovation for students as well as researchers to flourish. One example would be a course held in Rochester Institute of Technology, Australia [44].

CHAPTER 8 – CONCLUSION AND FUTURE WORK

The school conducted a game AI course that delves into the use of artificial intelligence in the creation of modern computer games. The focus is using existing tools for programming assignments in interactive fiction, first person shooter, real-time strategy, and simulations. The course included topics such as pattern matching, intelligent group movement, tactical reasoning, artificial life, and learning. It would be a great impact for the institutions, working hand in hand with commercial game companies to build powerful and dynamic environments for everyone.

REFERENCES

- [1] FM 90-10, Military Operations on Urbanized Terrain (MOUT). Headquarters, Department of the Army, Washington, D.C., 1979
- [2] FM 3-06, Urban Operations. Headquarters, Department of the Army, Washington, D.C., 2003
- [3] C. P. Stacey, Official History of the Canadian Army in the Second World War Volume III: The Victory Campaign: The Operations in North West Europe 1944-1945, 1960
- [4] J. R. Ballard, Fighting For Fallujah, A New Dawn for Iraq, 2006
- [5] "The Art of War", translated by Lionel Giles, 1910
- [6] R. Adobbati, A.N. Marshall, A. Scholer, S. Tejada, G.A. Kaminka, S. Schaffer, C. Sollitto, "GameBots: A 3D virtual world test bed for multiagent research", In Proceedings for the Second International Workshop on Infrastructure for Agents, 2001, Montreal, Canada
- [7] J. Akahani, K. Isbister, "Digital City Project: NTT Open Laboratory", NTT Communication Science Laboratories 2-4, 2000, Kyoto, JAPAN
- [8] L.B. McDonald, J. Weeks, T. Harris, "Security Forces Distributed Mission Training Technology Development", Proceedings of the 2000 Fall Simulation Interoperability Workshop, Sept 2000, Orlando, Florida
- [9] P. Prasithsangaree, J. Manojlovich, J.L. Chen, M. Lewis, "UTSAF: A Simulation Bridge between OneSAF and the Unreal Game Engine," IEEE International Conference on Systems, Oct 5-8, 2003, Washington, D.C., USA.

- [10] M. van Lent, R. McAlinden, P. Brobst, G. Barry, Silverman, K. O'Brien, C. Jason, "Enhancing the behavioral fidelity of synthetic entities with human behavior models". In Proceedings of the Thirteenth Conference on Behavior representation in Modeling and Simulation, pages 125–133, 2004
- [11] J. E. Laird, M. Assanie, B. Bachelor, N. Benninghoff, S. Enam, B. Jones, A. Kerfoot, C. Lauver, B. Magerko, J. Sheiman, D. Stokes, S. Wallace, "A Test Bed for Developing Intelligent Synthetic Characters", In Spring Symposium on Artificial Intelligence and Interactive Entertainment, AAAI, 2002
- [12] M. Lewis, J. Jacobson, "Game Engines in Scientific Research", Communications of the ACM, pages 27–31, Jan 2002
- [13] T. Sweeney, "Unreal Technology Features", Epic Games, Inc. 29 Jun, 1998
- [14] J. Jacobson, Using "CaveUT" to Build Immersive Displays With the Unreal Tournament Engine and a PC Cluster, Proceedings of the 2003 symposium on Interactive 3D graphics, Pages 221 – 222, 2003
- [15] Infiltration, <http://infiltration.sentrystudios.net/>
- [16] D. Arendash, "The Unreal Editor as a Web 3D Authoring Environment", Proceedings of the Ninth International Conference on 3D Web technology. Pages 119-126, 2004
- [17] B. Scott, C. Richard, "Training Effectiveness Evaluation of the Full Spectrum Command Game", Final report Jun 2002-Oct 2003, Jan, 2004
- [18] Karma, MathEngine Karma User Guide, 2002
- [19] V. Machado, B. Bredeweg, "Towards Interactive Tools for Constructing Qualitative Simulations", Fifteenth International Workshop on Qualitative Reasoning, 17-19 May, 2001, San Antonio, Texas

-
-
- [20] B. Bredeweg and K. Forbus, "Qualitative Modeling in Education", *AI Magazine*, Volume 24, Number 4, pages 35-46, 2003
- [21] P. Salles, B. Bredeweg, and R. Winkels, "Deriving Explanations from Qualitative Models", *Artificial Intelligence in Education: Knowledge and Media in Learning Systems*, IOS-Press/Ohmsha, pages 474-481, 1997, Japan, Osaka
- [22] K.D. Forbus, "Qualitative process theory", *Artificial Intelligence*, volume 24, number 1-3, 1984, pages 85-168.
- [23] M. Cavazza, S. Hartley, J-L.Lugrin, P. Libardi, M.L. Bras, "New Behavioral Approaches for Virtual Environments", *ICEC 2004*: 23-31
- [24] M. Cavazza, S. Hartley, J-L. Lugrin, M. L. Bras "Qualitative physics in virtual environments", *Intelligent User Interfaces*, pages 54-61, 2004,
- [25] M. Cavazza, S. Hartley, J.-L. Lugrin, M. Le Bras, "Alternative Reality: Qualitative Physics for Digital Arts", *Proceedings of the 17th International Workshop on Qualitative Reasoning 2003*, Brasilia, Brazil
- [26] M. Cavazza, S. Hartley, J.-L. Lugrin, M. Le Bras, "Alternative Reality: A New Platform for Digital Arts", *ACM Symposium on Virtual Reality Software and Technology*, pages 100-108, Oct, 2003, Osaka, Japan
- [27] P. L. Gustavson, "BOM Study Group Final Report", *Simulation Interoperability Workshop*, March 2001
- [28] S. Reichenthal: "The Simulation Reference Markup Language (SRML): A Foundation for Representing BOMs and Supporting Reuse", 02FSIW- 038, *Fall Simulation Interoperability Workshop*, Sept, 2002,Orlando, Florida,

- [29] S. M. Loh, R. Thiang, K. O. Chiam, "Behavioral Layering for Re-Use in Multiple Resolutions", The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC) , 21-25 Nov, 2003, Orlando, Florida
- [30] S. J. Thio, S. T. Kong, S. F. Tan, L. C. Yeo, "JEWEL – M&S Environment for the SAF", The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC), 1-7 Dec 2006, Orlando, Florida
- [31] E. Lewis and M. Barlow, "The Use of Games to Investigate Tactical Decision-making", SimTecT 05, 10-12 May, 2005, Sydney, Australia
- [32] K. D. Forbus, "Qualitative process theory", *Artificial Intelligence*, volume 24, number 1-3, pages 85-168, 1984
- [33] Animation Export Tools, Unreal Developer Network, <http://udn.epicgames.com/Two/ActorX>
- [34] F. Hill, "Attacking Asymmetrical Warfare Simulation Issues", Spring 2004 Simulation Interoperability, Apr 2004, Washington, D.C
- [35] S. TZU, 1982. The Art of War, trans. Samuel B. Griffith, Oxford University Press, New York, p. 78.
- [36] W. Sterren, "AI Game Programming Wisdom 2", pg 115-116, 2003.
- [37] S. Surasmith, "Preprocessed Solution for Open Terrain Navigation", "AI Game Programming Wisdom", Charles River Media, 2002.
- [38] J. Laird "Research in Human-Level A.I. Using Computer Games," Communications of the ACM, January, Vol. 45(1), pp. 32-35, 2002.

-
-
- [39] M. Köster, J. Teich: "(Self) reconfigurable Finite State Machines: Theory and Implementation". IEEE Automation and Test in Europe Conference and Exhibition, Paris, France. 2002.
- [40] S. S. Shah, E. W. Endsley, M. R. Lucas, D. M. Tilbury, "Reconfigurable logic control using modular Finite state machines: Design, verification, implementation, and integrated error handling". In Proceedings of the American Control Conference, 2002.
- [41] C. W. Reynolds, "Flocks, herds, and schools: a distributed behavioral model". Computer Graphics (ACM SIGGRAPH '87 Conference Proceedings), 21(4):25–34, July 1987.
- [42] C. W. Reynolds, "Leader Following Steering Behaviour", <http://www.red3d.com/cwr/steer/LeaderFollow.html>, 2004.
- [43] C. W. Reynolds, "Steering behaviors for autonomous characters", Game Developers Conference Proceedings, 2000.
- [44] J. D. Bayliss, "Artificial Intelligence for Computer Game Programming", Rochester Institute of Technology, <http://www.cs.rit.edu/~jdb/gameAI>, 2004.
- [45] S. P. Zhou, S. P. Ting, "Qualitative Physics for Movable Objects in MOUT", 39th Annual Simulation Symposium, April 2 - 6, 2006, Huntsville, AL, USA
- [46] S. P. Zhou, S. P. Ting, "A Generic Model Framework for MOUT Simulations", International Conference on CYBERWORLDS, Nov 28-30, 2006, Lausanne, Switzerland
- [47] S. P. Zhou, S. P. Ting, Z. Q. Shen, L. B. Luo, "Twilight City – A virtual environment for MOUT" (Accepted by International Journal of *Computer and Applications*)

- [48] S. P. Zhou, S. P. Ting, "A Qualitative Approach to Behaviour Simulation in Virtual Environments", (Under review by Simulation – Transaction of the Simulation Computer Society)
- [49] S. P. Ting, S. P. Zhou, "An Architecture for Rapidly Re-configurable MOUT Simulations", 40th Annual Simulation Symposium, 26 - 28 Mar, 2007, Norfolk, VA
- [50] S.P. Ting, S.P. Zhou, "Quartz: An Autonomous Navigation System for MOUT" , 2007 (Invited for Computer Animation and Virtual Worlds Journal)
- [51] S.P. Ting, S.P. Zhou, Quartz: An Autonomous Navigation System for MOUT, 20th International Conference on Computer Animation and Social Agents (CASA2007), 11-13 Jun, 2007, Hasselt, Belgium. (Ranked among 35 Best Papers, Invited for Computer Animation and Virtual Worlds Journal)
- [52] FM 90-10-1, An Infantry man's Guide to Combat in Built-up Areas
- [53] A. Al Ahuja, "Determining battlefield effects in an urban environment: MOUT terrain analysis - Military Operations in Urban Terrain", Military Intelligence Professional Bulletin, July-Sept 2003, 11 May. 2007
- [54] FM 34-130, Intelligence Preparation of the Battlefield, 8 July 1994
- [55] B. Kuipers, "Qualitative Reasoning: Modeling and Simulation with Incomplete Knowledge", The MIT Press, Cambridge, Massachusetts, 1994